

Universidad Internacional de la Rioja (UNIR)

ESIT

Máster Universitario en Inteligencia Artificial

**Agente conversacional para
búsqueda y recomendación
de ofertas laborales**

Trabajo Fin de Máster

presentado por: Héctor Gómez Loizaga

Dirigido por: Moisés Martínez Muñoz

Ciudad: Sevilla

Fecha: 18 de septiembre de 2019

Índice de contenidos

Índice de Ilustraciones	4
Índice de Tablas	3
Resumen	4
Abstract	5
1. Introducción	1
1.1 Descripción del problema	1
1.2 Motivación	2
1.3 Objetivos	2
1.4 Estructura del documento	3
2. Contexto y Estado del Arte	5
2.1 Procesamiento del lenguaje natural (NLP)	5
2.1.1 Recursos lingüísticos y representación del significado	6
2.1.2 Modelos del lenguaje	7
2.1.3 Aplicaciones y herramientas para PLN	8
2.2 Extracción automática de características	10
2.2.1 Tratamiento de datos no estructurados	10
2.2.2 Rastreo de datos (<i>web scraping</i>)	10
2.2.3 Reconocimiento de entidades (<i>NER</i>)	11
2.2.4 Clasificación de texto	12
2.3 Aprendizaje automático (ML)	13
2.3.1 Tipos de algoritmos de aprendizaje automático	14
2.3.2 Algoritmos de aprendizaje automático	15
2.4 Sistemas de recomendación	18
2.4.1 Sistemas de recomendación vs. sistemas de búsqueda	18
2.4.2 Técnicas de retroalimentación de información	19
2.4.3 Clasificación de sistemas de recomendación	19
2.4.4 Ventajas y contingencias	20
2.4.5 Implementaciones de sistemas de recomendación	21
2.5 Agentes conversacionales	23
2.5.1 Características de los agentes conversacionales	24
2.5.2 Agentes conversacionales dedicados vs. <i>chatbots</i>	24
2.5.3 Arquitectura de los agentes conversacionales	25
2.5.4 <i>Dialogflow</i>	26
3. Objetivos y metodología	27
3.1 Objetivos	27

3.1.1	Objetivo general	27
3.1.2	Objetivos específicos	27
3.2	Arquitectura del sistema	28
3.3	Metodología	29
4.	Identificación de Requisitos	31
4.1	Requisitos sobre el conjunto de datos (<i>dataset</i>)	31
4.2	Boletines de trabajo	32
4.3	Etiquetado automático del <i>dataset</i>	34
4.3.1	En función del nombre del boletín	34
4.3.2	<i>Clustering</i> en función de la descripción de la oferta laboral	35
4.3.3	Etiquetado manual	36
4.4	Descripciones de puestos laborales	39
5.	Desarrollo del trabajo	42
5.1	Procesado de datos	42
5.1.1	Secciones	43
5.1.2	Etiquetado	45
5.1.3	Clustering	45
5.2	Modelos de aprendizaje automático	46
5.2.1	Métricas de rendimiento	47
5.2.2	División del conjunto de datos	48
5.2.3	Clasificador bayesiano	48
5.2.4	Clasificación mediante redes neuronales	50
5.3	Selección de características y rendimiento de clasificadores	51
5.3.1	Analizador sintáctico	51
5.3.2	Comparativa de <i>datasets</i> y tipo de características	58
5.3.3	Reducción de la dimensionalidad	60
5.3.4	Mejora de la representatividad de las características	62
5.3.5	Clasificación mediante <i>word vectors</i>	69
5.4	Filtrado y recomendación	71
5.5	Agente conversacional	72
5.5.1	Estructura de la aplicación en <i>Python</i>	72
5.5.2	Control del flujo de diálogo	75
5.5.3	Entrenamiento del agente conversacional	80
5.5.4	Asignación del perfil laboral	82
5.5.5	Filtrado de ofertas laborales	85
5.5.6	Integración del agente conversacional	86
6.	Conclusiones y Trabajo Futuro	90
6.1	Conclusiones	90
6.2	Trabajo futuro	94

Bibliografía	95
Apéndice A	99
A.1 Módulo de pre-procesado: M_text_pre_processing.py	99
A.2 Módulo de análisis sintáctico: M_syntactic_analysis	100
A.3 <i>Datasets</i>	101
A3.1 Sobre el <i>dataset</i> de boletines de trabajo	101
A.3.2 Sobre el <i>dataset</i> de descripciones de perfiles laborales	110
A.4 Características (<i>features</i>)	116
A.4.1 Selección de características sobre el <i>dataset</i> completo	116
A.4.2 Selección de características individual por categorías	118
A4.3 Extracción de características	119
A.5 Agrupamiento (<i>clustering</i>)	122
A.5.1 030_Clustering_by_N.ipynb	122
A.6 Clasificadores	128
A6.1 040_Classifier_by_NB.ipynb	128
A6.2 040_Classifier_by_NN.ipynb	131
A6.3 050_Classifier_by_w2v.ipynb	135
Apéndice B	142
B.1 Configuración <i>Pycharm</i>	142
B.2 Módulos del proyecto	143
B.2.2 app.py	143
B.2.3 Inference.py	146
B.2.4 Filter.py	148
Apéndice: Artículo de investigación en español	149
Apéndice: Artículo de investigación en inglés	149

Índice de Ilustraciones

Ilustración 1. Distribución de categorías para etiquetado automático	36
Ilustración 2. <i>Clusters</i> para etiquetado automático	36
Ilustración 3. Distribución de categorías para etiquetado manual	37
Ilustración 4. <i>Clusters</i> para etiquetado manual	37
Ilustración 5. <i>Clusters</i> para <i>dataset</i> extendido	41
Ilustración 6. <i>Clusters</i> para <i>dataset</i> acotado	41
Ilustración 7. Importación del NLTK en <i>Python</i>	42
Ilustración 8. Muestra del <i>dataset</i> de boletines de trabajo	43
Ilustración 9. Atributos de los boletines de trabajos almacenados en listas	43
Ilustración 10. Formato original (web) de las descripciones de trabajo	44
Ilustración 11. Muestra del <i>dataset</i> de descripciones laborales	44
Ilustración 12. Precisión y función de pérdidas para modelos de redes neuronales	50
Ilustración 13. Pipeline de procesado del lenguaje natural	52
Ilustración 14. Ejemplo del proceso de etiquetado de <i>tokens</i> y segmentos para sintagmas nominales	52
Ilustración 15. Detección del par verbo transitivo-complemento directo	54
Ilustración 16. Rutina de segmentación según gramática en <i>Python</i>	54
Ilustración 17. Matriz de frecuencia de términos	57
Ilustración 18. Matriz de frecuencia de términos para categorías bien diferenciables	67
Ilustración 19. Expresión regular para obtención del rango salarial	71
Ilustración 20. Base de datos para filtrado y recomendación	71
Ilustración 21. Estructura del proyecto <i>job_consultant</i>	73
Ilustración 22. Gestión de peticiones HTTP recibidas desde el <i>webhook</i>	73
Ilustración 23. Configuración del puerto para el intercambio HTTP en la aplicación	74
Ilustración 24. Túnel creado con <i>ngrok</i>	74
Ilustración 25. Integración de motores de inferencia externos en <i>Dialogflow</i>	76
Ilustración 26. Configuración del <i>webhook</i> en <i>Dialogflow</i>	76
Ilustración 27. <i>Intents</i> del agente conversacional	76
Ilustración 28. Respuesta del <i>intent</i> de bienvenida	77
Ilustración 29. Toma de iniciativa del flujo de diálogo por parte del usuario	78
Ilustración 30. Frase para mantener las preferencias anteriores	79
Ilustración 31. Máquina de estados del diálogo del agente conversacional	79

Ilustración 32. Entidad de usuario	80
Ilustración 33. Frases de entrenamiento para el <i>intent consult</i>	81
Ilustración 34. Parámetros detectados para el <i>intent consult</i>	81
Ilustración 35. Respuesta estática para el <i>intent consult</i>	81
Ilustración 36. Respuesta formada por la combinación de dos frases estáticas	82
Ilustración 37. Regla para captar cualquier texto	82
Ilustración 38. Cancelación de contextos por código	83
Ilustración 39. Composición del resultado del motor de inferencia	83
Ilustración 40. Implementación de la extracción de características	84
Ilustración 41. Resultados del motor de inferencia	84
Ilustración 42. Frases de entrenamiento para el <i>intent job_salary</i>	85
Ilustración 43. Parámetros del <i>intent job_salary</i>	85
Ilustración 44. Resultado del módulo de filtrado	86
Ilustración 45. <i>iframe</i> para integración del agente conversacional	86
Ilustración 46. URL para depuración del agente conversacional	87
Ilustración 47. Configuración del entorno virtual de <i>PyCharm</i>	142

Índice de Tablas

Tabla 1. Fuentes de generación de datos _____	10
Tabla 2. Muestras por categoría para etiquetado manual _____	38
Tabla 3. Muestras por categoría para etiquetado semi-automático _____	40
Tabla 4. Comparativa de modelos bayesianos _____	49
Tabla 5. Gramáticas para el análisis sintáctico _____	54
Tabla 6. Características extraídas y frecuencia de aparición _____	55
Tabla 7. Representatividad de características extraídas _____	56
Tabla 8. Uso de sinonimia en la caracterización de las muestras _____	58
Tabla 9. Comparativa de resultados sobre el <i>dataset</i> original y el extendido _____	59
Tabla 10. Extracción de características sobre <i>dataset</i> completo vs. categorías individuales _____	62
Tabla 11. Métricas individuales por categoría _____	63
Tabla 12. Categorías con bajo porcentaje de acierto _____	63
Tabla 13. Métricas individuales por categoría tras cribado por mínimo de muestras _____	64
Tabla 14. Categorías con bajo rendimiento tras cribado por mínimo de muestras _____	64
Tabla 15. Rendimiento tras cribado por mínimo de muestras _____	65
Tabla 16. Reducción de característica para desacoplar categorías _____	68
Tabla 17. Comparativa matriz tf vs. tf-idf _____	68
Tabla 18. Rendimiento tras acotar el número de clases _____	69
Tabla 19. Comparativa matriz tf vs. word vectors _____	70

Resumen

El objetivo del presente trabajo es el desarrollo de la arquitectura y metodología utilizada por un agente conversacional para interactuar con un usuario y extraer la información necesaria que le permita actuar como orientador laboral. Se aplicarán técnicas de procesamiento del lenguaje natural orientadas a la extracción de información que será utilizada para la generación de características que puedan ser procesadas por algoritmos de aprendizaje automático. Es especialmente relevante que dichas características sean suficientemente discriminatorias entre los distintos perfiles laborales. El motor de inferencia se implementará como un clasificador que asignará el grupo o perfil laboral al que el usuario potencialmente pertenece y en función de información adicional sobre preferencias y formación del interlocutor, encontrará y mostrará un conjunto de ofertas laborales a las que el usuario esté capacitado para aplicar y que le puedan resultar de interés.

Palabras Clave: procesamiento del lenguaje natural, sistemas de recomendación, aprendizaje automático, extracción de características, agente conversacional

Abstract

The purpose of this project is developing the base architecture and methodology for a conversational agent conceived for extracting the key information from an end user to whom the system is due to give orientation for open job positions to apply. Natural language processing is used for getting the features that are later feed to a machine learning algorithm. It is crucial to find well-balanced features able to accurately split the different job profiles. The inference motor is implemented as an automatic classifier. This classifier is first predicting the user job profile and then is presenting the user with suitable job bulletins, which are filtered by using information about skills and preferences of the user.

Keywords: natural language processing, recommendation systems, machine learning, feature extraction, conversational agent

1. Introducción

La búsqueda de empleo ha sido asumida por la transformación digital en la que nos vemos inmersos[1]. Este proceso está aportando interesantes ventajas competitivas y operacionales en muchos sectores, de entre las cuales destacan la automatización de procesos y la consecuente minimización de costes, el incremento de la eficiencia y la productividad; la mejora de la comunicación interna y con el cliente, que a su vez facilita la adquisición de información sobre preferencias y hábitos de uso y consumo; y un notable aumento de la conectividad que ha posibilitado eludir algunas barreras geográficas y culturales. Hemos pasado de hacer cola en fábricas, acudir a la recepción de empresas para entregar nuestro currículum o revisar interminables y abrumadores boletines de trabajo a suscribirnos a portales web de empleo que nos envían avisos cada vez que se publica una oferta que se adapta a nuestro perfil laboral y pueden ser consultados en cualquier sitio y cualquier lugar desde nuestros dispositivos móviles. En este marco, este proyecto trata de investigar la viabilidad de aplicar técnicas de aprendizaje automático y de procesamiento del lenguaje natural para incrementar el nivel de automatización y la usabilidad de los portales de búsqueda de empleo.

1.1 Descripción del problema

En cada portal de empleo existe una interfaz distinta que actúa de intermediaria entre el usuario y los motores de búsqueda y filtrado de ofertas que implementa cada web. A su vez, si bien existen unas categorías genéricas relativamente bien definidas, cada portal de empleo utiliza ligeras variaciones en los criterios para agrupar las ofertas laborales en distintos perfiles profesionales. Estos factores pueden llegar a propiciar que se den dos situaciones aun utilizando los mismos datos (mismo usuario):

- Motores de búsqueda que filtran utilizando distintos criterios y/o trabajan sobre distintas categorías podría presentar distintas ofertas de trabajo.
- Interfaces distintas pueden llegar a dificultar al usuario aportar información precisa y completa, incurriendo en un sesgo a la hora de realizar el filtrado incluso aunque el motor de búsqueda fuese el mismo.

Para tratar de acometer las contingencias enumeradas, se investigarán varios enfoques encaminados cada uno a mejorar la precisión del motor de búsqueda y su usabilidad:

- Mediante técnicas de procesamiento del lenguaje natural y aprendizaje automático, se tratará de aprender de los datos tanto la definición de las distintas categorías laborales como los criterios de agrupación de las ofertas de trabajo en sus correspondientes perfiles laborales.
- El desarrollo de un interfaz más intuitivo, natural y asistido, reducirá la posible pérdida de información en el proceso de extracción de los distintos indicadores que se necesitan para definir el perfil laboral del usuario y filtrar ofertas en función de sus preferencias. El enfoque por el que se ha optado es utilizar un agente conversacional (chatbot) que se encargue de asegurarse de que el usuario no se deja información en el tintero.

1.2 Motivación

El procesamiento del lenguaje natural es sin duda una de las líneas de investigación y aplicación más complejas y con mayor trayectoria a futuro dentro del campo de la Inteligencia Artificial. Por ello, resulta especialmente atractivo profundizar en el uso de técnicas de procesado y aprendizaje para explorar y evaluar los beneficios que puede aportar la inclusión de este tipo de técnicas en algunos sistemas analíticos que utilicen enfoques más tradicionales. Por otro lado, la naturalidad en la comunicación que aporta el procesado mediante este tipo de técnicas, convierte la interacción con los dispositivos y/o algoritmos en un proceso más fluido e intuitivo, consiguiendo así que el usuario sea más propenso a aceptar los cambios que están siendo propiciados por la mencionada transformación digital

1.3 Objetivos

Se pretende invertir el proceso de extracción de información de tal forma que sea el sistema analítico el que tome la iniciativa. Para ello, se ha optado por utilizar un chatbot como interfaz para el usuario. En el marco de una conversación, el chatbot debe guiar y asistir al usuario para asegurarse de que este aporta información precisa y completa sobre la que trabajarán los distintos módulos del sistema analítico. Esta perspectiva está asociada a la

obtención de las siguientes ventajas:

- Estandarizar y naturalizar la interfaz de usuario permite maximizar las posibilidades de que el usuario encuentre una oferta laboral adecuada con independencia del portal de empleo que utilice para realizar la búsqueda.
- Mejorar la segmentación e identificación de perfiles laborales consigue que las ofertas de trabajo lleguen al usuario que realmente está interesado y cualificado para aplicar al puesto vacante. Con ello, las empresas aceptarán en sus procesos de selección a personal más apropiado e incrementarán la probabilidad de contratar a un trabajador que realmente encaje con los requisitos y expectativas de la empresa.

Tanto en la fase de entrenamiento como durante la fase de inferencia, el sistema debe ser capaz de cubrir principalmente tres tareas:

- Extracción de características para la correcta definición y/o diferenciación de los perfiles laborales.
- Categorización y asignación del perfil laboral más probable.
- Búsqueda y filtrado de ofertas laborales en función del perfil laboral asignado y de las preferencias del usuario.

1.4 Estructura del documento

A continuación se realiza una breve descripción del contenido de cada uno de los capítulos y anexos que incluye este documento:

- **Introducción:** se comienza presentando la cuestión que se pretende abordar en este trabajo, junto con las motivaciones que llevaron a escoger esta temática, los objetivos que se tratarán de alcanzar y la metodología a seguir durante el desarrollo del mismo.
- **Contexto y estado del arte:** en esta sección se documentan los ámbitos tecnológicos que sirven de base para el desarrollo del trabajo.
- **Objetivos y metodología:** en este capítulo se enumeran y analizan los objetivos que se pretenden alcanzar en el presente trabajo. Seguidamente, se detalla la arquitectura y metodología seguida para la construcción y despliegue del sistema analítico. A su vez, se justifica la elección de las herramientas y algoritmos utilizados en cada módulo.

- **Identificación de requisitos:** se identifican las principales características y condiciones que debe presentar el conjunto de datos sobre los que se construye y trabaja el sistema analítico a desplegar.
- **Desarrollo:** se detallan cada uno de los desarrollos realizados junto con los experimentos elaborados para verificar la funcionalidad y aplicabilidad de cada módulo del sistema. Dichos experimentos, se acompañan siempre de un análisis detallado que incide en las ventajas y contingencias inherentes a cada uno de los desarrollos.
- **Conclusiones y trabajo futuro:** se recogen en esta sección las conclusiones alcanzadas tras la realización del trabajo, junto con la evaluación del grado de completitud de cada uno de los objetivos y su aplicabilidad. Se evalúa en primer lugar el objetivo principal, para continuar luego con una valoración más detallada de cada uno de los subjetivos o partes constituyentes de la arquitectura del sistema (módulos). El capítulo, finaliza con una breve discusión sobre las posibles líneas de investigación futura que pueden continuar el trabajo expuesto en esta memoria.
- **Apéndices:** incluye los scripts, librerías y módulos de programación desarrollados para implementar cada uno de los módulos del sistema desarrollado.

2. Contexto y Estado del Arte

En esta sección, se documentan los ámbitos tecnológicos que sirven de base para el desarrollo del presente trabajo.

2.1 Procesamiento del lenguaje natural (NLP)

El procesamiento del lenguaje natural (*Natural Language Processing, NLP*) consiste en una serie de técnicas enfocadas al tratamiento y análisis del lenguaje natural. Su principal objetivo es que las computadoras puedan llevar a cabo tareas donde participa el lenguaje natural o humano [8]. En cuanto disciplina, se puede establecer su creación en los sesenta como parte del campo de la Inteligencia Artificial. Es sin duda un campo altamente interdisciplinar que abarca un amplio rango de tareas y los conocimientos asociados que se requieren para acometerlas: procesamiento de la señal (ingeniería electrónica y de telecomunicaciones), análisis sintáctico e interpretación semántica de palabras y frases (informática), morfología, fonología y pragmática (lingüística), estudio de los mecanismos cognitivos para la adquisición del lenguaje y cómo influye la sociedad en el uso de la lengua (sociólogos y psicólogos).

Las aplicaciones de procesamiento del lenguaje natural se distinguen de otros sistemas de procesamiento de datos por requerir el uso de conocimiento del lenguaje. Se pueden enumerar hasta seis aspectos o niveles sobre los que puede actuar dicho procesamiento[17]:

- Fonético: forma en que pronunciamos una determinada palabra.
- Morfológico: conocimiento acerca de la estructura de las palabras para su correcta delimitación (componentes significativos) y clasificación.
- Sintáctico: conocimiento sobre las relaciones estructurales entre palabras, utilizado para dividir las oraciones en sus componentes.
- Semántico: analiza el significado de la oración.
- Pragmático: estudia las oraciones y su significado en relación con los objetivos y las intenciones de los interlocutores.
- Discursivo: relaciona las distintas oraciones de un texto como unidades lingüísticas de mayor alcance.

Dicho conocimiento será necesario para el desarrollo de sistemas que sean capaces de realizar diverso tipo de tareas como las de análisis morfológico, análisis sintáctico, uso de gramáticas, análisis semántico y análisis pragmático del lenguaje.

Independientemente de la diversidad de idiomas y sistemas lingüísticos propios de cada lenguaje, un aspecto común que dificulta la computación del mismo es la ambigüedad, entendida como aquellas situaciones en las que la información se puede entender o interpretar de más de una forma. Esto no solo tiene porque deberse a la posibilidad de usar distintas palabras para expresar el mismo contenido (sinonimia), sino también al hecho de poder usar distinta sintaxis (estructura). De esta forma, se puede expresar el mismo contenido con estructuras distintas del lenguaje combinando distintas palabras, pero también se puede transmitir un mensaje diferente utilizando las mismas palabras si se agrupan u ordenan de forma distinta. Si unimos a esto, las propias diferencias dentro de cada idioma motivadas por el territorio, la incorporación de vocablos procedentes de dialectos propios de la región o el argot técnico asociado a las profesiones el panorama se complica [20].

2.1.1 Recursos lingüísticos y representación del significado

La implementación de las distintas tareas implícitas en el procesamiento del lenguaje natural, requiere de la utilización de una serie de recursos lingüísticos, de entre los cuales destacan:

- Diccionarios: repertorio en el que se recogen según un orden determinado las palabras o expresiones de una lengua acompañadas de su definición.
- Lexicón: se compone de la lista de morfemas de un idioma y la información básica sobre estos.
- Tesauros: incluye una lista de significados de palabras junto con las relaciones que guardan dichos significados. Estas relaciones pueden ser de sinonimia, antonimia, hiperonimia, hiponimia, meronimia y holonimia.
- Bases de datos de relaciones léxicas: contiene un conjunto de lemas, anotados con el posible conjunto de sentidos de cada palabra y las relaciones entre los sentidos.
- Corpus lingüístico: conjunto de textos suficientemente representativos de una lengua o de algún dominio concreto que se utilizan para el análisis lingüístico. Los corpus pueden estar anotados con algún tipo de información lingüística, lo cual favorece su uso para el entrenamiento de modelos de procesamiento del lenguaje natural.

Los recursos enumerados sirven de base para la creación de las distintas estrategias de representación del significado de las palabras orientadas a que puedan ser procesadas por un ordenador. Entre las más utilizadas destacan las siguientes estrategias:

Bases de datos relacionales: las palabras aparecen agrupadas por significado y las relaciones semánticas que se dan entre ellas. *Wordnet* [7], es un representativo ejemplo de este tipo de estrategias de representación del significado. A pesar de que este tipo de recursos funciona bastante bien, en muchas ocasiones se pierden matices sobre el contexto.

- A nivel computacional, no resulta directo obtener medidas de similitud a partir de estos datos. Además, requieren de gran esfuerzo para su recopilación y mantenimiento.

Representación discreta: partiendo de un vocabulario determinado, se representan las oraciones o textos mediante un vector donde cada elemento indica el número de veces que aparece cada palabra del vocabulario en dicho fragmento del documento (*term frequency*).

- Referenciada habitualmente como *bag-of-words*, este tipo de representación no evalúa el orden de aparición de las palabras (contexto) sino solo si aparecen o no en el texto. Suele derivar en vectores de datos muy dispersos.

Representación distribuida: consiste en la representación de una palabra mediante un vector denso. Dichos vectores se extraen mediante procedimientos estadísticos soportados por modelos de aprendizaje automático (redes neuronales) que evalúan un corpus y tratan de obtener para cada palabra los valores del vector (n-dimensional) de tal manera que dicho vector sea similar al de palabras que aparecen en contextos similares.

- Conocidos como *word vectors*, *word embeddings* o *word representations*, constituyen una representación manejable (computacionalmente) para expresar información semántica sobre similitud entre palabras.

2.1.2 Modelos del lenguaje

Principalmente, el problema de la modelización matemática para el análisis y tratamiento del lenguaje se aborda desde dos enfoques:

- Lógico: los expertos en lingüística establecen reglas para reconocer patrones estructurales y reflejan la estructura lógica del lenguaje. Hacen uso de lenguajes formales que permiten una representación inequívoca de los objetos, sus propiedades y las relaciones que guardan entre sí. Además, al estar basados en reglas permiten producir nuevo conocimiento mediante mecanismos de inferencia. Se basan en los estudios publicados por Chomsky en los años cincuenta [18].

- Probabilístico: basados en el análisis de colecciones de datos que conforman un corpus lingüístico. Se calculan las frecuencias de las unidades lingüísticas y se aplican técnicas de aprendizaje automático. La construcción de estos modelos se realiza principalmente mediante técnicas de entrenamiento supervisado, aunque también se está investigando la utilización de entrenamiento no supervisado con el objetivo de reducir el esfuerzo de tener que etiquetar los inmensos corpus sobre los que se entrena. Los modelos obtenidos pueden llegar a exhibir una elevada complejidad debido al alto número de variables y a que las muestras sobre las que se trabajan suelen ser muy dispersas.

2.1.3 Aplicaciones y herramientas para PLN

La interdisciplinariedad del campo conlleva implícitamente que exista una gran variedad de aplicaciones basadas en técnicas de procesamiento del lenguaje natural. Entre ellas podemos destacar:

- Agentes conversacionales (*chatbots*): programas que conversan con las personas a través del lenguaje natural. Los más avanzados permiten mantener conversaciones no estructuradas. Además, permiten interacción por texto o voz y son capaces de mantener el estado de la conversación para manejar aspectos tales como la gestión de los turnos de palabra y la generación de una conversación coherente.
- Traducción automática: su objetivo es la traducción automática de documentos o de conversaciones de un idioma a otro. Los métodos que aportan mejores resultados consisten en métodos estadísticos basados en aprendizaje automático. Con ellos, se analizan grandes conjuntos de datos y se realizan traducciones sin necesidad de contemplar en profundidad cuestiones gramaticales.
- Búsqueda de respuestas (*Question Answering, QA*): técnicas de recuperación de la información basadas en lenguaje natural enfocadas a responder preguntas complejas. Este tipo de sistemas requiere un buen desempeño en las siguientes tareas:
 - Extracción de información
 - Realizar inferencia para sacar conclusiones basadas en hechos conocidos
 - Sintetizar y resumir información de múltiples fuentes
 - Desambiguación del sentido de las palabras

De entre las numerosas herramientas y librerías enfocadas al desarrollo de aplicaciones de procesamiento del lenguaje [19] destacamos el *Natural Language Toolkit* y la *Stanford CoreNLP*.

- *Natural Language Toolkit (NLTK)* [9]: implementada en *Python* es sin duda una de las herramientas más utilizadas. La herramienta proporciona acceso a más de 50 corpus (incluido *Wordnet*) y recursos lingüísticos además de una amplia colección de librerías que resuelven tareas típicas del procesamiento del lenguaje tales como obtención de *tokens*, *stemming*, etiquetado morfosintáctico (*POS*), derivación, análisis sintáctico, análisis semántico, tareas de clasificación y análisis de sentimientos (*sentiment analysis*). Aunque está desarrollada principalmente para el procesamiento de la lengua inglesa, permite el uso de corpus en otros idiomas para el entrenamiento de etiquetadores morfosintácticos y adaptación del resto de funcionalidades de las librerías.
- *Stanford CoreNLP*: incorpora directamente modelos que han sido construidos en otros idiomas a parte del inglés (entre ellos el español), por lo que puede alcanzar mejores rendimientos al contemplar aspectos concretos del idioma en estudio. Implementado en Java, permite también una fácil integración con módulos programados en *Python* (pudiendo así embeber funcionalidades del NLTK).

Por otro lado, cabe destacar el interés demostrado por las grandes empresas de desarrollo tecnológico en copar el mercado con sus propias soluciones comerciales, las cuales ofrecen servicios completos, escalables y fáciles de integrar a través de las APIs disponibles y que normalmente son distribuidos desde sus plataformas *cloud*. Estos servicios pueden ser accedidos de forma gratuita durante un periodo determinado o con limitaciones en el ratio de uso, estando sus funcionalidades completamente disponibles bajo suscripción o contrato de paquetes completos.

- Google Cloud Natural Language
- Amazon Comprehend
- IBM Watson Natural Language Understanding
- Microsoft Language Understanding (LUIS)

2.2 Extracción automática de características

La extracción de características es un proceso de codificación y reducción que tiene el objetivo de reducir un conjunto de datos a características o atributos que capten con precisión la información contenida en los datos y que sean tratables por programas informáticos. Se describen en esta sección algunos de los métodos de uso más habitual para la extracción de características sobre documentos de texto.

2.2.1 Tratamiento de datos no estructurados

Una de las dificultades que tiene trabajar sobre documentos de texto es el hecho de que se trata datos que no incorporan una estructura implícita y que por tanto requieren de un procesado adicional para añadir información o estructura a los datos de partida, de tal forma que puedan ser gestionados de forma adecuada por los motores de inferencia [5][6]. Mediante el uso de herramientas y técnicas de *Big Data* [28], tales como *Hadoop* o bases de datos no relacionales como *NoSQL*, se puede convertir dicha información en información relevante. En el caso de documentos de texto, se hará prácticamente imprescindible el uso de técnicas de procesamiento del lenguaje natural.

Datos estructurados	Datos no estructurados
Tarjetas de identificación, cuentas bancarias, teléfonos, códigos postales, etc.	Correo electrónico, archivos de texto, presentaciones, sitios web, imágenes, archivos de audio y video, documentos generados por aplicaciones informáticas, etc.

Tabla 1. Fuentes de generación de datos

2.2.2 Rastreo de datos (*web scraping*)

El rastreo de datos (*web scraping*) es una técnica que permite navegar de forma automática por la red y extraer información de los portales web que se visitan. La extracción de información se basa en el conocimiento de las etiquetas *HTML* con las que se estructuran las distintas secciones y el contenido en las páginas web.

En sus inicios se utilizaba para recolectar información de consumo directo, tal como noticias, ofertas inmobiliarias, etc. Posteriormente, y especialmente debido a la proliferación de las redes sociales se comenzó a utilizar con fines más analíticos: ver las preferencias de los usuarios respecto a marcas y productos, analizar el contenido de blogs y foros, estudiar los precios en el mercado, localizar o controlar a la competencia (precios y productos), estudiar nichos del mercado, migrar datos de las webs, generar alertas, detectar cambios en una

web, alimentar una base de datos y cualquier otra información que sea útil a nuestros intereses. Al poder automatizarse el proceso, es una herramienta que permite la obtención de mucha información en poco tiempo y con relativamente poco esfuerzo [29].

Existen varias herramientas que permiten crear sin mucho esfuerzo un software que realice esta función y que recibe entre otras las denominaciones *spider*, *bot* o *crawler*. *Python* es sin duda el lenguaje mayormente empleado para rastrear mediante las implementaciones de varias de sus librerías: *Scrapy*, *BeautifulSoup* y *Selenium*. Hay que advertir que ciertas operaciones de rastreo pueden tener implicaciones legales. De hecho, muchas compañías poseen aplicaciones que detectan comportamientos no humanos en la navegación y evitan las visitas de los *bots*. Por su parte, los *bots* utilizan *proxies* para variar las IP de acceso, simular un comportamiento humano y evitar la detección. Como alternativa, algunas compañías habilitan APIs que permite acceder a datos sin hacer *scraping*, lo cual es beneficioso para las dos partes [30].

2.2.3 Reconocimiento de entidades (NER)

Conocido por sus siglas en inglés como *NER (Named Entity Recognition)*, el reconocimiento de entidades es una técnica que permite localizar e identificar todas las menciones textuales a una serie de entidades predefinidas. Estas entidades podrían ser organizaciones, personas, ubicaciones, fechas, instalaciones, etc. Cualquier instancia de estas entidades será localizada y etiquetada junto con su correspondiente entidad [31]. Las implementaciones disponibles están basadas en el uso de algoritmos de aprendizaje automático, gramáticas y corpus lingüísticos. En *Python* se puede acceder a una implementación pre entrenada incluida dentro del módulo correspondiente del NLTK.

Este tipo de herramientas juega un papel muy interesante en las tareas de recuperación de información (*information retrieval*), ya que permite obtener directamente los pares *clave-valor* evitando así tener que recopilar grandes cantidades de texto a las que se les tendría que aplicar posteriormente expresiones regulares. Dado que el significado de las palabras depende del contexto, los modelos no pueden resolver todas las ambigüedades contenidas en un texto dado. Las herramientas estadísticas y de aprendizaje automático resuelven en parte el problema pero necesitan de un gran volumen de datos de entrenamiento: entidades conocidas a priori [9]. Este aspecto fuerza a estos algoritmos a partir de un etiquetado manual de los datos, lo cual es un proceso costoso en tiempo e infraestructura dado el volumen necesario de datos a etiquetar para conseguir resultados aceptables.

2.2.4 Clasificación de texto

Como paso previo a las tareas de clasificación de texto necesitaremos obtener descriptores de la información contenida en el texto que sea manejable por los algoritmos. Dichas descripciones suelen estar basadas en la frecuencia de aparición de palabras en el texto.

- *Tf-idf (Term frequency – inverse document frequency)*: cuantificador utilizado para ponderar la relevancia de una palabra en un documento que forma a su vez parte de una colección de documentos. El valor asignado aumenta cuanto mayor sea la frecuencia de la palabra en un documento dado y disminuye cuando mayor sea la frecuencia de dicha palabra en el global de documentos. De esta forma se trata de penalizar las palabras genéricas que se repiten numerosamente en la colección de documentos. Matemáticamente se expresa con la siguiente fórmula:

$$tf\ idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

siendo $tf(t, d)$ la frecuencia de aparición del término t (palabra) en el documento d y $idf(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$, con D el número total de documentos y $1 + |\{d \in D : t \in d\}|$ el número de documentos donde aparece el término t , al que se suma la unidad para evitar la división entre cero si el término no aparece en la colección [32].

Una vez calculado este indicador para cada atributo en cada documento (matriz), se puede utilizar como entrada para algoritmos de aprendizaje automático, tales como redes bayesianas, redes neuronales, árboles de decisión, *clustering*, etc.

2.2.4.1 Latent Dirichlet Allocation (LDA)

La Asignación Latente de Dirichlet (*LDA*) es una técnica ideada para la detección de tópicos (categorías) en documentos de textos. Este método trata de modelar los documentos como un conjunto finito de tópicos asumiendo que estos siguen una distribución de Dirichlet. A su vez, cada tópico se asume que puede ser descrito por una distribución de palabras. El término latente proviene del hecho de que los tópicos, que no son observables (latentes), pueden ser determinados en función la distribución de palabras en los documentos y los propios documentos (conjunto de tópicos); ya que ambos elementos sí son observables. La cuantificación de las palabras se realiza mediante *bag-of-words*, siendo relevante si la palabra aparece o no en el documento, pero no la posición en que esta es observada. Se dice que es un modelo generativo porque los tópicos se “fijan” antes de generar o escoger las palabras que componen los documentos [33].

2.3 Aprendizaje automático (ML)

La disciplina del aprendizaje automático engloba una serie de métodos y algoritmos orientados a la extracción de conocimiento de forma autónoma a partir de datos. Los datos, representan los ejemplos que los algoritmos procesan para ajustar sus parámetros internos. De esta forma, durante el proceso de entrenamiento (ajuste) a medida que se procesa un mayor número de muestras el algoritmo adquiere mayor conocimiento a través de la experiencia [21]. A su vez, el entrenamiento se corresponde matemáticamente con la resolución de problemas de optimización, donde el procesado de cada muestra sirve al algoritmo para ajustar mejor sus parámetros buscando minimizar alguna métrica o medida del error cometido al realizar inferencias.

Enmarcada dentro de la rama de la inteligencia artificial, puede a su vez ser considerada como un campo de las ciencias de la computación en cuanto a su claro enfoque a la definición e implementación de algoritmos. Más concretamente, uno de los aspectos diferenciadores de este tipo de técnicas es el desarrollo de programas informáticos que puedan adaptarse a cambios o situaciones no vistas con anterioridad sin necesidad de ser programados explícitamente. La capacidad de adaptación la obtendrían a partir del aprendizaje obtenido durante el proceso de entrenamiento.

En cuanto a los objetivos, se pueden distinguir dos metas bien diferenciadas:

- Obtener el patrón o correspondencia existente entre los datos de entrada y de salida. Se trata por tanto de modelar la función que define como un sistema concreto transforma o procesa las entradas para generar unas determinadas respuestas. Para aquellos sistemas que exhiben una complejidad suficientemente elevada [22], no es viable encontrar la función analítica que modele el sistema en estudio. En estos casos es mucho más práctico encontrar una aproximación a dicha función; lo cual es lo que tratan de hacer los algoritmos de aprendizaje automático.
 - Este requisito se asocia típicamente a problemas de clasificación y regresión.
- Descubrir la estructura implícita en los datos. En este caso lo que se pretende es encontrar aquellas características que definen o describen los distintos elementos de un conjunto de datos.
 - La aplicación directa de este enfoque es la segmentación de los elementos presentes en los datos en distintas clases en función de sus características.

Dado que los algoritmos de aprendizaje automático están diseñados para aprender de los datos, resulta obvio deducir que si no se dispone de un conjunto de datos de calidad los algoritmos tendrán serias dificultades para obtener resultados aceptables. Podemos identificar principalmente dos requisitos sobre los datos de entrenamiento:

- Conjunto de datos abundantes: es imprescindible disponer un número elevado de muestras para que los algoritmos tengan la oportunidad de procesar suficientes ejemplos y completar con precisión el proceso de optimización: cubrir el total de la población que se pretende clasificar o mostrar la dinámica del sistema en todo su rango de actuación. De esta forma, el patrón obtenido podrá abarcar la diversidad de casos que puedan darse y permitirá realizar predicciones sobre datos no vistos durante el entrenamiento. Si se dispone de pocas muestras, las salidas propuestas por el modelo estarán completamente sesgadas.
- Variables representativas: la elección de las variables de entrada (subconjunto de los datos en 'crudo') para el algoritmo influye notablemente en su desempeño. Las características seleccionadas deben ser suficientemente predictores del sistema a modelar, es decir, deben ser aquellas variables que mejor describan y/o diferencien las respuestas del sistema.

Uno de los factores que ha propiciado el auge de este tipo de técnicas en la última década, sobre todo en la rama del *Deep Learning* [23], ha sido el reciente desarrollo de la capacidad de almacenamiento (disposición masiva de datos) y el importante incremento en la potencia de cálculo.

2.3.1 Tipos de algoritmos de aprendizaje automático

Podemos realizar una clasificación de los distintos tipos de algoritmos de aprendizaje automático en función de las técnicas para su implementación y de la metodología de entrenamiento utilizada:

- Aprendizaje supervisado: se dispone de un conjunto de datos etiquetados con la respuesta asociada. Durante el entrenamiento, en sucesivas iteraciones se procesa cada muestra y se compara con la salida esperada. La cuantía del error cometido en la predicción se evalúa a través de alguna métrica y se utiliza para reajustar los parámetros internos del sistema.
 - Aplicable a problemas de clasificación (asignar una entre n clases a cada muestra) y regresión (predicción de un valor numérico).

- Aprendizaje no supervisado: en este caso no se dispone de salida o respuesta esperada, sino que se procede a determinar la estructura interna de los datos. Estas técnicas trabajan con métricas de similitud entre los vectores de características de las muestras.
 - Aplicable a problemas de *clustering* (agrupar instancias muy similares en categorías) y detección de anomalías (detectar valores muy poco habituales).
- Aprendizaje semisupervisado: este tipo de algoritmos combina aprendizaje en base a etiquetas y clasificación en función de la estructura interna de los datos.
- Aprendizaje por refuerzo: los algoritmos de aprendizaje por refuerzo están orientados a la obtención de conocimiento del entorno con el que se interactúa y permiten tener en consideración las consecuencias futuras de realizar una acción en el momento de tomar la decisión [24]. Matemáticamente se modelan mediante los procesos de decisión de Markov:
 - El agente puede percibir un conjunto S de estados y realizar un conjunto A de acciones para interactuar con el entorno.
 - En cada instante de tiempo, el agente percibe un estado s_t y selecciona (en función a su política de actuación) una acción a_t que dará lugar al nuevo estado $s_{t+1}=a_t(s_t)$
 - El entorno responde a cada acción mediante una recompensa $r(s_t, a_t)$ que indica el beneficio que ha reportado al agente realizar dicha acción encontrándose en ese estado concreto.
 - El agente tratará de aprender la mejor política de acciones $\pi(s, a, r)$ en función de la combinación de estados y acciones disponibles junto con la recompensa esperada de tomar dichas decisiones.
 - Aplicable a la implementación de agentes inteligentes

2.3.2 Algoritmos de aprendizaje automático

Presentamos a continuación una breve descripción de los principales algoritmos desarrollados en el campo del aprendizaje automático.

- Regresión Logística: modelos que tratan de explicar y predecir una característica cualitativa a partir de los datos de otras variables conocidas (cuantitativas o cualitativas) que actúan como variables explicativas. La característica que se quiere explicar suele seguir modelos binomiales [25]. Se basa en el uso de la función *logit*, logaritmo de la función impar (*log-odds*). Al tomar esta función valores de salida en el

rango $[0,1]$ puede interpretarse como la probabilidad de que un dato pertenezca a una clase determinada dadas sus características.

- Uso en problemas de clasificación.
- Máquinas de Vector Soporte (SVM): modelo geométrico que busca encontrar un plano que separe espacialmente las clases a categorizar. En este proceso (problema de optimización), se realiza una transformación del espacio de características que aumenta (enriquece) la dimensión inicial con la intención de facilitar la separación de las clases. Cuando las clases no sean linealmente separables, la ecuación del plano puede sustituirse por alguna función no lineal (*kernel*).
 - Uso en problemas de clasificación y regresión.
- Árboles de decisión (*decision trees*): este tipo de técnicas funciona segmentando sucesivamente el espacio de variables predictores en una serie de regiones. En cada división, el algoritmo selecciona la variable más predictiva de la variable objetivo en función de alguna métrica de pureza (máxima información). Continúa realizando divisiones hasta alcanzar el criterio de parada. Los nodos intermedios (ramas) representan la toma de decisiones. Los nodos finales (hojas) conforman la predicción. Se puede controlar el sobreajuste del modelo mediante la definición de una profundidad máxima para el árbol o un número mínimo/máximo de muestras que deben quedar en cada segmento al realizar la división. Este tipo de modelos presentan una característica analítica muy interesante, ya que observando las variables escogidas para realizar las divisiones se puede evaluar fácilmente la relevancia de los atributos de entrada.
 - Uso en problemas de clasificación y regresión.
- Bosques Aleatorios (*Random Forests*): los *Random Forest* surgen para solucionar la baja capacidad predictiva que exhiben los modelos basados en árboles de decisión. Para ello, se construyen n modelos de árboles de decisión que serán promediados para obtener la predicción. Cada uno de estos modelos no trabaja sobre el total de variables de entrada, sino sobre un subconjunto de ellas seleccionado de forma aleatoria. Con ello, se consigue reducir la varianza del resultado a la vez que se controla el sobreajuste del modelo.
 - Uso en problemas de clasificación y regresión.
- Red bayesiana : modelo probabilístico basado en el teorema de Bayes, que relaciona la probabilidad de dos eventos utilizando la dependencia condicional que existe entre ellos. Se trata de un método muy empleado en la clasificación de textos [26] porque ofrece un análisis cualitativo y cuantitativo de los valores que intervienen en el problema. En lo cualitativo, se puede representar como se relacionan los atributos de forma causal o mediante la correlación existente entre los ellos. Cuantitativamente,

da una medida de probabilidad de la importancia de las variables en el problema. Este tipo de técnicas es capaz de aportar buenos resultados incluso cuando los datos son muy dispersos, existe ruido en las muestras o cuando no se dispone de información completa (*missing*) sobre los atributos en todos los ejemplos.

- Uso en problemas de clasificación y regresión (discretizados).
- Redes Neuronales (*Neural Networks*): modelos basados en el paradigma conexionista que construye una red de unidades lógicas interconectadas (“neuronas”) a través de las cuales se propaga la información. Las unidades de procesamiento están organizadas según una arquitectura de capas: una capa de entrada, con una unidad por cada atributo de entrada; una o varias capas intermedias (ocultas) que procesan y propagan la información; y una capa de salida, con unidades que codifican la predicción calculada por el modelo. Individualmente, cada neurona aplica la función de activación (función matemática) al sumatorio de entradas ponderadas con un peso: $y(x) = f(\sum_{i=1}^n w_i X_i)$. La salida de cada neurona es a su vez la entrada de las neuronas de la siguiente capa. Este tipo de modelos muestran gran poder de expresividad, permitiendo captar patrones muy complejos (altamente no lineales). Debido a la flexibilidad que le aporta el elevado número de parámetros internos que lo forman, pueden usarse para resolver un gran rango de problemas de distinta tipología. Es uno de los modelos con mayor potencia predictiva.
 - Uso en problemas de clasificación y regresión.

2.4 Sistemas de recomendación

“Los sistemas de recomendación son una serie de mecanismos y técnicas aplicadas a la recuperación de información para intentar resolver el problema de sobrecarga de datos en Internet. Estos ayudan a los usuarios a elegir los objetos que pueden ser de utilidad o de su interés, pudiendo estos objetos ser de cualquier tipo, tales como libros, películas, canciones, páginas web, blogs.” [\[11\]](#)

2.4.1 Sistemas de recomendación vs. sistemas de búsqueda

Un aspecto relacionado con los sistemas de recomendación es la “recuperación de información” o “búsqueda de información” (*information retrieval*). Ambos conceptos se asemejan en el hecho de que intentan proporcionar información relevante al usuario pero se distinguen por ciertas singularidades [\[12\]](#):

- Objetivo: Los sistemas de búsqueda seleccionan los ítems de la base de datos que coinciden con la pregunta (*query*) mientras que los sistemas de recomendación eliminan la información irrelevante de flujos de entrada de información o reúnen información relevante de diferentes repositorios de acuerdo al perfil del usuario.
- Base de datos: Los sistemas de búsqueda trabajan con bases de datos relativamente estáticas mientras que los sistemas de recomendación trabajan con información dinámica.
- Tipos de usuarios: En los sistemas de búsqueda no se tiene por qué tener información sobre los usuarios que lo utilizan mientras que en los sistemas de recomendación se necesita saber o tener información sobre los usuarios.
- Ámbito social: Los sistemas de recomendación están interesados en aspectos sociales de modelado y privacidad del usuario mientras que los sistemas de búsqueda no.
- Frecuencia de uso: Los sistemas de búsqueda están enfocados a un uso concreto y puntual por parte del usuario mientras que los sistemas de recomendación están diseñados para un uso prolongado y de forma repetitiva.
- Representación de las necesidades de información: En sistemas de búsqueda, la información requerida se expresa en forma de pregunta (*query*), mientras que en los sistemas de recomendación la información es descrita en los perfiles del usuario.

2.4.2 Técnicas de retroalimentación de información

Un sistema de recomendación debe proporcionar un mecanismo para recopilar la mayor cantidad posible de información relacionada al perfil de los usuarios con el fin de descubrir sus gustos e intereses por determinados objetos y posteriormente generar recomendaciones atractivas para el usuario. Este proceso se llama “retroalimentación”. El proceso de retroalimentación puede ser clasificado de acuerdo a dos tipos [13]: retroalimentación implícita y la explícita.

- Retroalimentación implícita: este proceso consiste en evaluar los objetos, sin las intervenciones de los usuarios, es decir, a través de la captura de la información obtenida de las acciones realizadas por los usuarios en la aplicación. Este tipo de técnicas aprovechan del comportamiento del usuario para comprender los intereses y preferencias del usuario.
- Retroalimentación explícita: estas técnicas recopilan información a través de un proceso de encuesta, en el que el usuario valora los contenidos mediante la asignación de puntuación. La Retroalimentación explícita proporciona a los usuarios un mecanismo para expresar de manera inequívoca su interés en los objetos.

2.4.3 Clasificación de sistemas de recomendación

Los sistemas de recomendación se suelen clasificar de acuerdo al tipo de información que utilizan para realizar las recomendaciones. Principalmente, se pueden distinguir tres grupos [14]:

- Basados en contenidos: tratan de recomendar productos similares a los que le han gustado a un usuario determinado en el pasado. Estos productos han sido previamente valorados por el usuario. Para saber que un ítem es parecido a otro se buscan “palabras clave” del ítem que calificó el usuario. Esto puede presentar un problema ya que si siempre se recomiendan ítems similares a los vistos anteriormente se puede llegar a lo que se conoce como la sobre-especialización. Una de las posibles soluciones es añadir cierta aleatoriedad a las recomendaciones del sistema.
- Basados en filtrado colaborativos: tienen como objetivo conocer las preferencias del usuario y hacer recomendaciones sobre la base de datos de los usuarios y la comunidad. El sistema recomienda ítems de otros usuarios con “gustos” similares a los suyos. Para ello, el sistema de recomendación calcula la similitud entre usuarios.

- Híbridos: explotan características de los sistemas basados en contenido y colaborativos, debido a la naturaleza complementaria de ambos tipos.

Mencionamos también, otras clasificaciones más específicas de sistemas de recomendación que debido a sus características pueden considerarse particularizaciones de los tres grupos recién presentados:

- Sistemas basados en ontologías o esquemas de conceptos
- Sistemas adaptables al contexto
- Sistemas basados en redes de confianza
- Recomendaciones demográficas (*Demographic*)
- Recomendaciones basadas en el conocimiento (*Knowledge based*)
- Recomendaciones basadas en la utilidad (*Utility based*)

2.4.4 Ventajas y contingencias

Los sistemas de recomendación ofrecen una amplia gama de ventajas tanto a los usuarios que utilizan internet para buscar o compartir información, como a las instituciones que utilizan la web para ofrecer sus servicios y tener presencia en el mercado.

- Personalización del cliente: ofrecen recomendaciones personalizadas y acertadas
- Marketing efectivo: se presenta como una forma atractiva y eficiente de interacción al crear una experiencia de compra personalizada para cada cliente.
- Retención del cliente: permite generar informes que ayudan a saber por qué y qué productos se están vendiendo y quién los está comprando. Esto sirve de base para la elaboración de estrategias enfocadas a la retención de clientes.
- Incrementa el volumen de ventas: facilitar el proceso de búsqueda hace que la tendencia de compra sea mayor.
- Satisfacción del usuario: Mejoran la satisfacción de los usuarios, porque le ayudan descubrir nuevos contenidos de su interés de una forma fácil y con un mínimo esfuerzo.
- Ahorro de tiempo: Reducción de tiempo por parte de los usuarios para encontrar productos que le gusten o le parezcan interesantes.

Por otro lado, el diseño, implementación y mantenimiento de este tipo de sistema implica una serie de contingencias:

- Sobrecarga de datos: el acceso al gran volumen de datos disponibles en internet requiere de mecanismos y algoritmos de clasificación que permitan optimizar las búsquedas y el acceso a estos contenidos eficientemente.
- Mecanismo eficiente de retroalimentación: la mayoría de implementaciones están basadas en retroalimentación explícita. Esto puede causar inconvenientes a los usuarios, ya que típicamente les resulta tedioso tener que valorar los contenidos.
- Arranque en frío: cuando el sistema de recomendación no dispone de suficiente información acerca de un usuario o contenido, es difícil que pueda realizar recomendaciones acertadas.
- Coste: asociado al mantenimiento del sistema. Es necesario pensar en modelos de negocio que generen suficientes ingresos y para cubrir estos costes.
- Diseño de los sistemas de recomendación: la interfaz del sistema de recomendación tiene que ser atractiva y debe mejorar la interacción con el usuario. Es fundamental captar la atención de los usuarios.

2.4.5 Implementaciones de sistemas de recomendación

En último lugar mencionamos algunas herramientas o webs que disponen de sistemas de recomendación:

- Fab: sistema híbrido, basado en contenido y filtrado colaborativo, que recomienda al usuario páginas de Web. El sistema modela el perfil del usuario basándose en las calificaciones que el usuario hace de las páginas y compara los perfiles para determinar similitudes entre usuarios ofreciendo una recomendación colaborativa.
- Filmaffinity: sistema de recomendación de películas, documentales, cortometrajes, medimetrajes y series de televisión muy popular. Se basa en el filtrado colaborativo.
- Amazon: sistema de recomendación utilizado por la tienda virtual Amazon basado en un algoritmo de filtrado colaborativo *item-to-item* [15], que a diferencia de los sistemas de filtrado colaborativos tradicionales, es escalable e independiente del número de clientes y del número de ítems en el catálogo de productos. Realiza recomendaciones en tiempo real, opera sobre un conjunto de datos masivos, y genera recomendaciones de alta calidad. Al contrario que otros sistemas, este sistema de recomendación se enfoca en la búsqueda de productos similares en vez de usuarios similares. Por cada compra y valoración explícita que los usuarios realizan sobre un producto, el sistema busca los productos similares y los añade a los catálogos de productos asociados para recomendar.

- Google news: este sistema genera recomendaciones usa tres tipos de técnicas:
 - Filtrado colaborativo usando agrupación (*clustering*) *MinHash*: agrupamiento probabilístico que asigna un par de usuarios a un mismo grupo con probabilidad proporcional a la coincidencia entre el conjunto de elementos que estos usuarios han votado a favor.
 - Indexación Semántica Latente Probabilística (*Probabilistic Latent Semantic Indexing, PLSI*): indexación automática de documentos basada en un modelo de clases de semántica latente [16]. Se realiza indexación de palabras que tienen relación semántica.
 - *Covisitation*: método basado en ítems que calcula qué productos tienden a mirar los usuarios después de haber visto un determinado artículo.

Este sistema de recomendación funciona para los usuarios que se registran y que han permitido de forma explícita el registro de su historial web. El sistema construye los perfiles para las noticias de interés de los usuarios en función de la navegación pasada de los usuarios.

- Google AdSenses: Esta herramienta de Google permite insertar en las páginas web de las diferentes aplicaciones de Google (gmail, docs, etc.) publicidad acorde con los gustos y preferencias de cada usuario particular.

2.5 Agentes conversacionales

Los agentes conversacionales son sistemas diseñados para comunicarse con los usuarios mediante el uso del lenguaje natural. Si bien la interacción suele establecerse principalmente a través de mensajes de voz (audio) o mediante la escritura (texto), también se puede hacer uso de otras modalidades de la comunicación integrando gráficos, imágenes o animaciones, que pueden contribuir a la construcción de entornos altamente interactivos.

El interés por este tipo de sistemas está presente prácticamente desde los inicios de la disciplina de las ciencias de la computación (principios de 1940). Tanto es así que en 1950 Alan Turing ideaba el “*Test de Turing*” (originalmente nombrado como “*Imitation Game*”), un método que permitía evaluar si una máquina podía ser calificada como inteligente [35]. En esta prueba, se sometía a aislamiento visual a un juez humano (interrogador) que mediante el establecimiento de una conversación debía determinar si el agente conversacional con el que mantiene el diálogo es un humano o una máquina. Recogiendo esta semilla, en 1990 Hugh Loebner en colaboración con el Centro de Estudios del Comportamiento de Cambridge crean el premio Loebner, en el que los participantes presentan sus diseños de agentes conversacionales para tratar de superar el mencionado test.

De entre los primeros desarrollos, destacamos el realizado por Joseph Weizenbaum que en 1966 creó ELIZA mientras trabajaba en el MIT. El algoritmo que utilizaba realizaba búsqueda de palabras clave en las respuestas del usuario. Más tarde, en 1972, K. Colby desarrolló PARRY, utilizado para evaluar las conversaciones de personas que padecían paranoia. En 1995 R.Wallace implementó ALICE, con la que ganaba el premio Loebner en 2000, 2001 y 2004. En las últimas ediciones Bruce Wilcox con ROSE y S. Worswick con MITSUKO han acaparado los primeros puestos.

Hoy en día los agentes conversacionales son empleados en ámbitos tan diversos como el entretenimiento, la educación, psicología, idiomas, ventas, marketing, etc. Entre sus principales aplicaciones podemos destacar su uso en sistemas de atención al cliente así como en la asistencia al proceso de compra, donde se presentan como interfaz para el motor de recomendación.

2.5.1 Características de los agentes conversacionales

Los agentes conversacionales tienen que estar diseñados, entrenados y programados para interpretar la intención de las conversaciones, entender las preguntas y respuestas del interlocutor y decidir la acción que se debe realizar en cada momento. La resolución de este tipo de tareas no triviales conlleva que este tipo de sistemas estén basados habitualmente en el uso de técnicas de procesamiento del lenguaje natural (*NLP*) y que apliquen métodos y algoritmos de aprendizaje automático.

En el contexto de una conversación, el agente deberá modelar y gestionar los comportamientos que se dan en los diálogos entre humanos:

- Turnos de palabra: alternancia de intervenciones.
- Actos del habla: representan las acciones involucradas en el mensaje transmitido.
- Puntos de coincidencia: fundamentos o creencias que comparten los participantes.
- Estructura de la conversación: patrones observables en los diálogos.
- Información implícita: conocimiento y contexto compartido por los interlocutores.

2.5.2 Agentes conversacionales dedicados vs. *chatbots*

En base al tipo de conversación que sean capaces de mantener y el objetivo con el que se hayan implementado, podemos distinguir entre dos tipos de agentes conversacionales:

- Agentes conversacionales dedicados: diseñados para el desarrollo de una tarea concreta y por tanto restringidos a un dominio de aplicación. Generalmente, esto implica que están enfocados a la resolución de tareas sencillas, para las cuales es suficiente mantener una conversación corta que requiera pocas interacciones con el usuario. En esta categoría encontramos asistentes personales como Alexa (Amazon), Siri (Apple), Google Assistant, etc.
- Chatbots: también conocidos como *bots* conversacionales. Permiten entablar conversaciones extensas con múltiples interacciones, basadas en un diálogo no necesariamente estructurado y que pueden versar sobre cualquier materia.

2.5.3 Arquitectura de los agentes conversacionales

Un agente conversacional basado en voz presentará los siguientes módulos [8]:

- Reconocimiento automático del habla
 - Función: generación de texto (palabras) a partir de una señal acústica.
 - Técnicas: modelado acústico-fonético, gramáticas de estados finitos y modelos probabilísticos.
- Comprensión del lenguaje natural
 - Función: implementa tareas de procesamiento del lenguaje natural enfocadas a extraer el significado (semántica) de las frases.
 - Requisitos: cuando el agente puede realizar tareas en múltiples dominios, primero tiene que detectar la intención del usuario para poder así escoger el *frame* que le permite realizar una correcta interpretación.
 - Técnicas: plantillas (*frames* [36]) con pares atributo-valor que el agente rellena tras examinar el texto de entrada. Es habitual el uso de gramáticas semánticas.
- Gestión del diálogo
 - Función: determinar la acción a realizar y/o el concepto a transmitir en el turno de palabra actual como respuesta a la petición realizada por el usuario.
 - Requisitos: mantener el estado y flujo de la conversación. Requerir información al usuario cuando esta sea necesaria para generar la respuesta.
 - Técnicas: máquina de estados finitos donde los estados son las preguntas a realizar al usuario, las cuales tratan de obtener los valores para los huecos de la plantilla (*frame*) en uso. Los arcos se corresponden con las acciones a realizar en función de la respuesta obtenida del usuario. Como alternativa, se pueden utilizar reglas de producción, que permiten obtener el valor de varios de los huecos de la plantilla en un mismo turno en función de qué reglas estén activas.
- Generación del lenguaje natural
 - Función: escoger los conceptos a transmitir y planificar la forma de expresarlos [37].
 - Técnicas: patrones de pares adyacentes (saludo-saludo, pregunta-respuesta, etc.) en función de los actos del habla asociados. Se procede primero a la generación de una frase des lexicalizada (con huecos) que se rellena a continuación con valores que serán inferidos en función de la información extraída del usuario.
- Conversión del texto al habla
 - Función: sintetizar en formato acústico el mensaje a transmitir.
 - Técnicas: síntesis de formantes o reglas, síntesis articulatoria y síntesis basada en concatenación (resultado más natural).

2.5.4 Dialogflow

Dialogflow es la plataforma de Google de desarrollo de productos orientados a facilitar la interacción entre humanos y computadoras basada en aprendizaje automático y técnicas de procesamiento del lenguaje natural. Consta de una serie de elementos y funcionalidades que le permiten la creación, configuración y entrenamiento de agentes conversacionales de forma cómoda e intuitiva:

- **Intención (*intents*):** un *intent* sirve para relacionar una serie de mensajes (frases) que el usuario puede emitir con alguna tarea que el agente está capacitado para resolver.
 - Contextos: los contextos se utilizan tanto para guiar el flujo de la conversación como para desambiguar el significado de las frases. De este modo, se puede conseguir que determinadas reglas de producción (*intents*) solo se activen cuando se den ciertas circunstancias (contexto).
 - Eventos: sucesos externos que pueden utilizarse para activar un *intent*.
 - Frases de entrenamiento: conjunto de textos con los que reacciona un *intent*.
 - Acciones: identificador de la acción o tarea asociada al *intent*.
 - Parámetros: entidades detectadas en el texto del usuario.
 - Respuestas: texto que se presenta al usuario cuando la regla del *intent* se ha verificado. Este texto puede ser puramente estático, incluir los valores de los parámetros extraídos de la entrada del usuario o ser generados por alguna aplicación externa (mediante el uso de *Fulfillments*).
- **Entidades (*entities*):** *Dialogflow* se basa en *NER* ([2.2.3 Reconocimiento de entidades \(NER\)](#)) para resolver los pares atributo-valor de las plantillas (*intents*) que utiliza.
- **Fulfillment:** API que habilita la comunicación entre *Dialogflow* y cualquier aplicación externa mediante el uso de *webhooks*. Permite generar las respuestas del agente mediante el uso de motores de inferencia o reglas de producción externas.
- **Integrations:** permite integrar de forma ágil el agente conversacional con los servicios de mensajería, asistentes y aplicaciones de chat más populares: Facebook Messenger, Slack, Twitter, Kik, Line, Skype, Telegram, Twilio and Viber, Google Assistant, Amazon Alexa, Microsoft Cortana, etc.

El registro de históricos de conversación y la sección de analíticas son las principales herramientas que provee *Dialogflow* para la evaluación y seguimiento del rendimiento del agente. Destacar también que *Dialogflow* dispone de una serie de agentes pre-construidos y entrenados que resuelven gran variedad de tareas habituales para este tipo de sistemas, tales como configurar alarmas, realizar pedidos de productos, reservas de restaurantes, etc.

3. Objetivos y metodología

En este capítulo se enumeran y analizan los objetivos que se pretenden alcanzar el presente trabajo. Seguidamente, se detalla la arquitectura y metodología seguida para la construcción y despliegue del sistema analítico. A su vez, se justifica la elección de las herramientas y algoritmos utilizados en cada módulo

3.1 Objetivos

3.1.1 Objetivo general

El objetivo principal del presente trabajo es investigar la viabilidad del uso de técnicas de procesamiento del lenguaje natural y de aprendizaje automático para naturalizar y asistir el proceso de búsqueda de trabajo a través de medios digitales. Implícitamente, se trata de construir una herramienta de búsqueda de empleo con interfaz implementada a través de un agente conversacional que identifique el perfil laboral del usuario y filtre la base de datos de ofertas laborales en función del perfil asignado y las preferencias del usuario.

3.1.2 Objetivos específicos

Los objetivos específicos descomponen el objetivo principal en tareas contenidas que pueden ser planificadas y abordadas con mayor eficiencia. Se enumeran a continuación las metas que se pretenden abarcar:

- Extracción de características sobre datos no estructurados. Se trata de abordar la gestión de datos que no presentan una estructura implícita y que por tanto presentan una mayor complejidad a la hora de extraer información con valor. Son archivos que por lo general pueden ser almacenados y administrados sin que el sistema tenga conocimiento del formato de los mismos y que suele suponer un alto porcentaje de la cantidad de datos generados por las empresas (hasta un 80%)[\[5\]](#) [\[6\]](#).
- Comparar el rendimiento de distintas técnicas de clasificación automática y evaluar los requisitos para ser aplicadas.
- Usar técnicas de filtrado y recomendación para refinar la clasificación de categorías y

para ayudar a obtener resultados más precisos.

- Guiar el proceso de extracción de información y naturalizar la comunicación con el usuario mediante el uso de agentes conversacionales como interfaz para el sistema.

3.2 Arquitectura del sistema

Para la construcción del sistema, se ha considerado interesante utilizar una arquitectura modular. Cada módulo ha de resolver alguna tarea bien definida y acotada y generar una salida que pueda ser utilizada de forma transparente por cualquier otro módulo de procesamiento. El alcance y ámbito de cada módulo viene habitualmente determinado por los objetivos específicos que se desean cubrir. En este sentido podemos distinguir cuatro módulos:

- Módulo de extracción de características
 - Cometido: tratamiento de datos no estructurados para obtención de indicadores predictores para el problema a resolver
 - Técnica: procesamiento del lenguaje natural
 - Herramienta: NLTK en *Python*
 - Entrada: texto con descripción de tareas a realizar en cada trabajo
 - Salida: frecuencia de palabras relevantes en cada texto
- Módulo de inferencia
 - Cometido: asignación del perfil laboral más probable
 - Técnica: clasificación automática mediante aprendizaje automático
 - Herramienta: *Keras* y *Scikit-Learn* en *Python*
 - Entrada: frecuencia de palabras relevantes en cada texto
 - Salida: categoría laboral
- Módulo de filtrado y recomendación
 - Cometido: selección en base de datos de ofertas relevantes para el usuario
 - Técnica: filtrado con claves de búsqueda
 - Herramientas: *Python*
 - Entrada: categoría laboral asignada y preferencias del usuario
 - Salida: conjunto de ofertas laborales

- Módulo de interfaz conversacional
 - Cometido: guiar al usuario para asegurar que aporta información precisa y completa para realizar la búsqueda
 - Técnica: modelo de conversación basado en *frames* y gestión del diálogo mediante máquinas de estados finitos
 - Herramientas: *Dialogflow*
 - Cometido: tratamiento de datos no estructurados para obtención de preferencias del usuario
 - Técnica: expresiones regulares y reconocimiento de entidades
 - Herramientas: *Python* y *Dialogflow*
 - Entrada: texto de usuario
 - Salida:
 - Módulo de inferencia: texto introducido por el usuario
 - Módulo de filtrado y recomendación: preferencias del usuario
 - Usuario: pregunta para requerir información, texto informativo y conjunto de ofertas laborales encontradas

3.3 Metodología

Las técnicas de aprendizaje automático, por lo general, se basan en algoritmos diseñados para captar patrones de los datos; emulando así el proceso de aprendizaje que el ser humano realiza cuando adquiere conocimiento de la experiencia. Se comenzará acudiendo a repositorios de datos y seleccionando algún *dataset* que se adapte a las necesidades del trabajo. Tras evaluar su representatividad, se ponderará la posibilidad de ampliar el conjunto de datos mediante técnicas de *web scraping*.

El siguiente paso consistirá en extraer características de nuestra fuente de datos no estructurada. Para ello, se utilizarán técnicas de procesamiento del lenguaje natural mediante analizadores morfosintácticos y un posterior filtrado aplicando gramáticas para extraer los constituyentes gramaticales que puedan ser más predictivos de nuestro conjunto de datos. Se comparará el grado de representatividad que se obtiene del uso de sintagmas nominales, sintagmas verbales o la combinación de ambos. Se contabilizará la frecuencia de aparición de los constituyentes gramaticales obtenidos y se seleccionarán los más frecuentes (matriz de frecuencia) como características de entrada para el motor de inferencia. También se probarán distintos enfoques orientados a refinar el proceso de selección de características para conseguir que sean lo más representativas posible y tratar

de reducir el solapamiento entre categorías que sean similares. La posibilidad utilizar distintas palabras para expresar la misma idea o contenido, nos emplazará a considerar la sinonimia a la hora de extraer las características del texto de entrada. Para el desarrollo de estas tareas se han utilizado rutinas programadas en *Python* combinadas con módulos del *Natural Language Toolkit* (NLTK).

La información extraída de la descripción de cada profesión será utilizada para alimentar los algoritmos de aprendizaje automático (motores de inferencia) encargados de catalogar el perfil laboral del usuario. En este aspecto, se evaluará la viabilidad de diferentes algoritmos de aprendizaje automático para entrenar un clasificador. Reservaremos un 80% de los datos para el entrenamiento de los modelos y un 20% para la evaluación de los mismos. Se utilizarán con profusión las librerías *Keras* y *Scikit-learn* desarrolladas en *Python*.

Una vez obtenido el perfil laboral del usuario, el sistema de recomendación encontrará en su base de datos las ofertas laborales disponibles para dicho perfil. Para evitar presentar información abundante e innecesaria al usuario, se realiza un filtrado atendiendo a información adicional extraída del usuario, tal como la horquilla salarial, la ciudad en la que busca incorporarse, etc. Si bien esta información también podría extraerse mediante técnicas de aprendizaje automático, se comprobará que al ser información muy concreta, en la mayoría de las ocasiones bastará aplicar expresiones regular al texto de entrada.

En todo momento, el agente conversacional (*chatbot*) se presentará como el punto de entrada y salida para el usuario. Implementado con *DialogFlow*, aportará al usuario una interfaz fluida y cómoda que propiciará que el usuario tenga una mejor predisposición al uso del sistema. Nos centraremos en la definición de la estrategia de gestión del diálogo y en la verificación de que el usuario ha aportado la información requerida mediante el uso de máquinas de estado finitos y el empleo de técnicas basadas en *frames*. También se prestará atención a las estrategias para conseguir mayor variabilidad en la formulación de las preguntas, lo cual aportará más naturalidad a la conversación.

En definitiva se pretende acometer el ciclo completo de desarrollo de una aplicación basada en técnicas de inteligencia artificial, partiendo de la extracción de características desde los datos en crudo, pasando por el entrenamiento y evaluación de un motor de inferencia basado en técnicas de aprendizaje automático, y finalizando con la creación de un interfaz que naturaliza la interacción con el usuario entregando un resultado que logre aportar algún beneficio al interlocutor; en este caso, asistirle en la búsqueda de ofertas laborales que se adapten a su perfil y preferencias.

4. Identificación de Requisitos

En esta sección, se identifican las principales características y condiciones que debe presentar el conjunto de datos sobre los que se construye y trabaja el sistema analítico a desplegar.

4.1 Requisitos sobre el conjunto de datos (*dataset*)

Como se ha mencionado al describir el estado del arte de las técnicas de aprendizaje automático, para que los modelos puedan ser capaces de aprender los patrones implícitos en los datos y puedan usar este conocimiento para generalizar o predecir tendencias es fundamental que los datos utilizados para construir el modelo sean suficientemente abundantes y representativos (diversos) del problema que se pretende modelar. A menudo, encontraremos que el proceso de recopilación de datos puede convertirse en un factor altamente determinante para que los algoritmos seleccionados puedan obtener niveles de rendimiento aceptables. En problemas de clasificación, principalmente podremos notar alguno de los inconvenientes enumerados a continuación:

- El algoritmo no será capaz de predecir una categoría si no disponemos de un número suficiente de ejemplos o muestras de dicha categoría. El número mínimo de ejemplos depende de dos factores:
 - La naturaleza de los datos: un patrón más complejo requerirá de más muestras para que el algoritmo sea capaz de en primer lugar determinar la base del patrón y en segundo lugar distinguir detalles particulares o anomalías del patrón base (evitar *overfitting*).
 - El alcance del clasificador: cuantas más categorías tenga que distinguir el clasificador, necesitará un mayor número de ejemplos de cada categoría para conseguir evitar solapamiento o confusión en las predicciones. Se puede observar fácilmente como partiendo de un clasificador que predice de forma aceptable 20 clases, si reducimos el número de clases a predecir a 10, el clasificador incrementará notablemente su rendimiento ya que estamos simplificando el problema a resolver.

- Si la mayoría de las muestras de las que disponemos solo describen o representan parte del patrón o dinámica oculta en los datos, el algoritmo tenderá a generar una salida sesgada que apunte la mayoría de sus predicciones hacia el patrón que más ha visto durante la fase de entrenamiento. También podría suceder, que al no poder captar adecuadamente los indicadores correctos de cada categoría, se produzca solapamiento en las predicciones y confunda clases que tengan algunas características comunes.

Como punto de partida, sería adecuado disponer de al menos en torno a 80-120 muestras por categoría para que el modelo pudiera predecir las categorías con suficiente fiabilidad.

4.2 Boletines de trabajo

Tras una búsqueda exhaustiva por distintos repositorios de datos de uso habitual para investigación o desarrollo de proyectos de ciencia de datos (*data science*), se ha seleccionado el *dataset* “*Data Science for Good: City of Los Angeles*¹” publicado en la plataforma web *Kaggle* [2].

El *dataset* consiste en un conjunto 683 boletines de ofertas de trabajo en formato texto y en inglés que presentan las siguientes secciones:

- Job name: nombre del puesto de trabajo, fecha de incorporación e identificador en base de datos.
- Annual salary: horquilla o rango salarial para el puesto vacante
- Duties: competencias, habilidades y tareas asociadas al desarrollo del trabajo ofertado
- Requirement/minimum qualification: requisitos mínimos para acceder al proceso de selección. Principalmente en forma de experiencia laboral y formación.
- Process notes: documentos y plazos de obligado cumplimiento
- Where to apply: medio y/o punto de recogida de candidaturas
- Application deadline: fecha límite para recepción de curriculums
- Selection process: indicaciones sobre cómo se realiza el proceso de selección y en qué consiste
- Notes: información adicional que pueda resultar de interés.

Para nuestro propósito, la información considerada relevante es la siguiente:

¹ <https://www.kaggle.com/c/data-science-for-good-city-of-los-angeles>

- Nombre del puesto de trabajo
 - Se puede extraer del primer encabezado del documento o del propio nombre del documento de texto.
- Descripción del puesto de trabajo
 - Contenido en la sección “*Duties*”.
 - De aquí trataremos de extraer los indicadores que nos permitan clasificar cada oferta de trabajo en distintos perfiles laborales.
- Formación, experiencia mínima y demás requisitos
 - Sección “*Requirement/minimum qualification*”.
 - Esta información puede usarse para filtrar las ofertas de trabajo una vez obtenido el perfil profesional del usuario.
- Salario mínimo y máximo
 - Sección “*Annual salary*”.
 - Información útil para el filtrado de ofertas.
- Fecha de incorporación
 - Sección “*Job name*”.
 - Información útil para el filtrado de ofertas.

El primer inconveniente que encontramos en el presente *dataset* es que las muestras no están etiquetadas, es decir, cada oferta laboral no está asociada a una categoría; la cual nos interesaría que se correspondiera con su perfil profesional: contable, comercial, ingeniero, etc. Dado que esto es un requisito fundamental para poder entrenar el clasificador que asigna la categoría laboral, en la siguiente sección se detallan los enfoques que se han utilizado para tratar de producir un etiquetado coherente de las muestras de forma automática.

4.3 Etiquetado automático del *dataset*

4.3.1 En función del nombre del boletín

El primer enfoque que se ha utilizado para tratar de agrupar en categorías las distintas ofertas laborales se basa en que habitualmente, el nombre de la categoría laboral aparece directamente en el nombre de la oferta de trabajo o en su defecto aparece algún término relacionado. Partiendo de este supuesto se procede de la siguiente forma:

1. Se examinan los nombres de las ofertas de trabajo contando cuantas veces se repite cada palabra.
2. Se establece un umbral consistente en el número de veces que se repite una palabra.
3. Se seleccionan las palabras que superan el umbral fijado y se eliminan aquellas que no aporten información (adverbios, pronombres, etc.), así como aquellas que son consideradas demasiado genéricas (*'principal'*, *'senior'*, etc.).
4. Se recorren todos los nombres de las ofertas de trabajo comparando cada palabra del nombre con las palabras más comunes encontradas en el paso anterior. Se asignará siempre la categoría de la palabra más común de entre las que coincidan.

Se muestra a continuación el resultado para distintos umbrales (*threshold*) y la selección tras el filtrado:

```
Most common words(th = 15 ): 33
['AIRPORT', 'ANALYST', 'AND', 'ASSISTANT', 'ASSOCIATE', 'BUILDING',
'CHIEF', 'COMMUNICATIONS', 'CONSTRUCTION', 'DIRECTOR', 'ELECTRICAL',
'ELECTRICIAN', 'ENGINEER', 'ENGINEERING', 'INSPECTOR', 'MAINTENANCE',
'MANAGER', 'MECHANIC', 'OF', 'OFFICER', 'OPERATOR', 'POLICE', 'PRINCIPAL',
'SENIOR', 'SERVICES', 'SPECIALIST', 'SUPERINTENDENT', 'SUPERVISOR',
'SYSTEMS', 'TECHNICIAN', 'UTILITY', 'WATER', 'WORKER']
```

```
Most common words(th = 20 ): 22
['ANALYST', 'AND', 'ASSISTANT', 'ASSOCIATE', 'CHIEF', 'DIRECTOR',
'ELECTRICAL', 'ENGINEER', 'ENGINEERING', 'INSPECTOR', 'MANAGER',
'MECHANIC', 'OF', 'OFFICER', 'OPERATOR', 'POLICE', 'PRINCIPAL', 'SENIOR',
'SERVICES', 'SPECIALIST', 'SUPERVISOR', 'TECHNICIAN']
```

```
Selected job groups:(th = 20 ): 18
['ANALYST', 'ASSISTANT', 'ASSOCIATE', 'CHIEF', 'DIRECTOR', 'ELECTRICAL',
'ENGINEER', 'ENGINEERING', 'INSPECTOR', 'MANAGER', 'MECHANIC', 'OFFICER',
'OPERATOR', 'POLICE', 'SERVICES', 'SPECIALIST', 'SUPERVISOR', 'TECHNICIAN']
```

Se puede observar como en el listado aparecen demasiadas palabras genéricas que no resultan fáciles de eliminar de forma automática, ya que en algunos casos presentan mayor

frecuencia de aparición que las palabras que son realmente descriptivas de un perfil laboral (frecuencia de *'manager'* > frecuencia *'engineering'*) y en otros aparecen con menor frecuencia (frecuencia de *'director'* < frecuencia *'engineering'*).

```
('INFORMATION', 10), ('ELECTRIC', 10), ('TREATMENT', 10), ('CLERK', 11),
('PORT', 11), ('TRANSPORTATION', 11), ('EQUIPMENT', 11), ('WASTEWATER',
11), ('DRAFTING', 12), ('INVESTIGATOR', 12), ('ENVIRONMENTAL', 12),
('MECHANICAL', 13), ('REPRESENTATIVE', 13), ('STREET', 14), ('SERVICE',
14), ('BUILDING', 15), ('AIRPORT', 16), ('ELECTRICIAN', 16), ('SYSTEMS',
16), ('WATER', 16), ('COMMUNICATIONS', 17), ('MAINTENANCE', 18), ('WORKER',
18), ('CONSTRUCTION', 18), ('SUPERINTENDENT', 19), ('UTILITY', 19),
('ASSOCIATE', 20), ('SPECIALIST', 22), ('ELECTRICAL', 22), ('INSPECTOR',
23), ('ANALYST', 24), ('POLICE', 24), ('OF', 24), ('DIRECTOR', 25),
('TECHNICIAN', 25), ('MECHANIC', 26), ('OPERATOR', 26), ('OFFICER', 28),
('SERVICES', 28), ('ENGINEERING', 28), ('AND', 30), ('PRINCIPAL', 30),
('MANAGER', 32), ('ASSISTANT', 33), ('CHIEF', 34), ('ENGINEER', 38),
('SUPERVISOR', 93), ('SENIOR', 97)
```

El efecto de esta situación, es que las categorías aglutinarán ejemplos propios de su categoría junto con muestras que podrían pertenecer a cualquier otra categoría. Esto provocará que el clasificador no consiga aprender correctamente los patrones de los datos.

4.3.2 *Clustering* en función de la descripción de la oferta laboral

El segundo enfoque utilizado consiste en extraer características² (*features*) de la descripción de cada oferta laboral (campo *'Duties'* del documento de texto) y utilizar algoritmos de agrupamiento [3] (*clustering*) para asignar las categorías de forma automática.

Las técnicas de aprendizaje supervisado, tratan de modelar la función o patrón de transformación que el sistema en estudio aplica a las entradas para obtener unas salidas dadas. En contraposición, las técnicas de *clustering* consisten en determinar la estructura existente en los datos y utilizar esta información para agrupar muestras similares dentro de la misma categoría.

Se procede a etiquetar las muestras utilizando los 18 grupos obtenidos por frecuencia de palabras y añadiendo una categoría adicional (*'Other'*) para aquellos ejemplos que no contengan ninguna de las palabras seleccionadas. La distribución que se obtiene, deja claro que los grupos apenas cubren la diversidad de la muestra, ya que casi todos los ejemplos son clasificados en la categoría *'Other'* y el resto presentan un número demasiado bajo de ejemplos.

² Los detalles de cómo se realiza la extracción de características se especificarán en el capítulo 5.

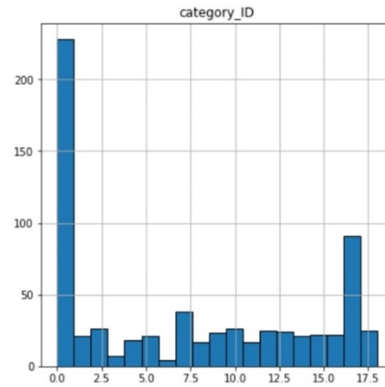


Ilustración 1. Distribución de categorías para etiquetado automático

Si bien es sabido que *Spherical k-Means Clustering* [4] suele presentar mejores resultados para clasificación de texto, se ha utilizado la versión estándar del algoritmo *k-Means* (basada en distancias euclidianas en vez de en la disimilitud del coseno) para obtener una primera aproximación y evaluar la viabilidad del enfoque.

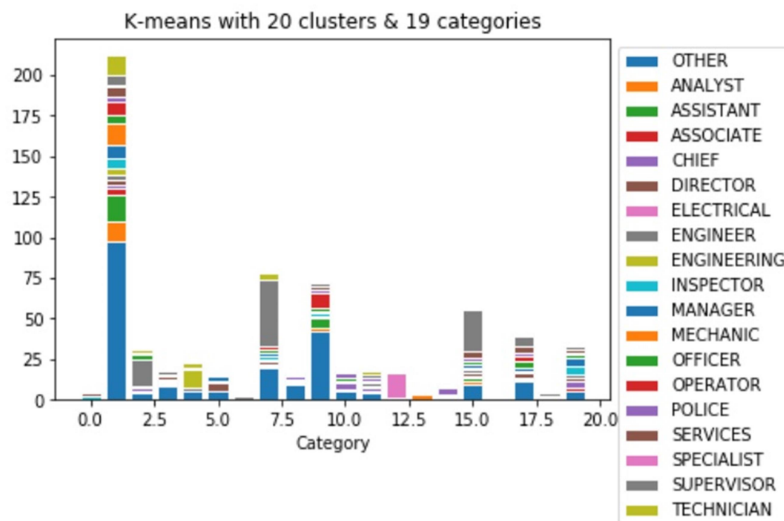


Ilustración 2. Clusters para etiquetado automático

Se puede observar como la mayoría de los *clusters* encontrados presentan una mezcla de muestras de demasiadas categorías, no pudiendo hallar agrupaciones que claramente engloben a alguna de las categorías. Ante esta circunstancia, se decide proceder a un etiquetado manual de las muestras.

4.3.3 Etiquetado manual

Tras analizar y comparar los perfiles laborales que típicamente suelen utilizar los portales de empleo, se opta por la utilización de las siguientes categorías:


```
{'NONE': 0, 'AGRICULTURE, CATTLE AND FISHING': 1, 'BANKING, ACCOUNTING AND INSURANCE': 2, 'BUILDING AND CONSTRUCTION': 3, 'BUSINESS ADMINISTRATION': 4, 'CHEMIST, BIOLOGY AND ENVIRONMENT': 5, 'COMPUTER AND PROGRAMMING': 6, 'CUSTOMER SERVICE, SECRETARIAT AND CLERKS': 7, 'EDUCATION AND FORMATION': 8, 'ENGINEERING AND TECHNICIANS': 9, 'HEALTH, SOCIAL SERVICES AND CARE': 10, 'HUMAN RESOURCES': 11, 'LANGUAGES': 12, 'LAWYERS': 13, 'LEISURE AND ENTERTAINMENT': 14, 'MARKETING, COMUNICATION AND PUBLISHING': 15, 'MECHANICS AND MAINTENANCE': 16, 'MULTIMEDIA, DESIGNERS AND ARTISTS': 17, 'OTHERS': 18, 'PRODUCTION, QUALITY AND R&D': 19, 'SECURITY': 20, 'SHOPPING, LOGISTICS AND STOCK': 21, 'SPORTS AND BEAUTY ': 22, 'TOURISM, HOTEL AND CATERING': 23, 'TRADE AND SALES': 24, 'TRANSPORT': 25, 'WORKERS AND OPERATORS': 26}
```

La distribución obtenida tras el etiquetado se muestra en la siguiente imagen:

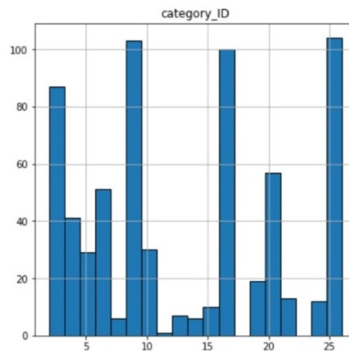


Ilustración 3. Distribución de categorías para etiquetado manual

Nos gustaría observar que el algoritmo de *clustering* al menos es capaz de consolidar los ejemplos de las cuatro categorías más pobladas. Sin embargo, se puede apreciar como de nuevo el agrupamiento parece bastante irregular.

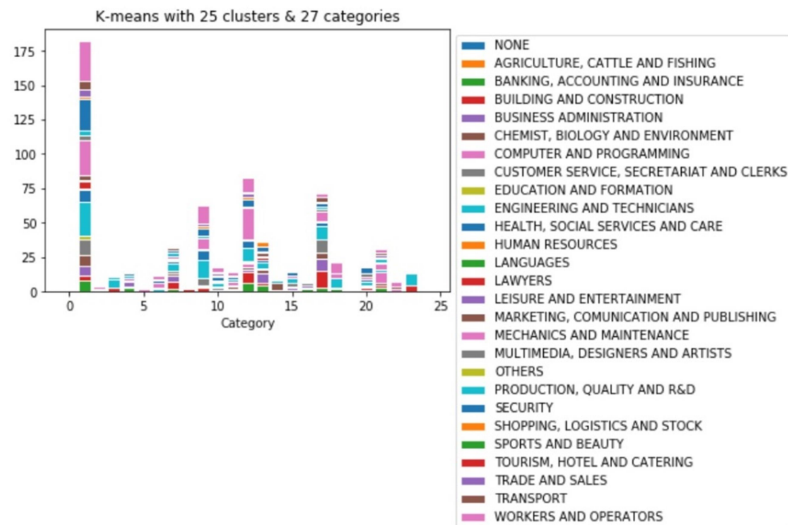


Ilustración 4. Clusters para etiquetado manual

Parece que la causa principal que dificulta el proceso de *clustering* es el reducido número de ejemplos que tenemos para cada categoría (tabla 2). Además, con el objetivo de conseguir aglutinar más ejemplos para cada clase durante el etiquetado se han mezclado algunas

categorías a priori similares. Esto puede provocar que las muestras sean aún menos representativas y confundan al algoritmo de agrupamiento.

Categoría	Número de muestras
'NONE'	0
'AGRICULTURE, CATTLE AND FISHING'	0
'BANKING, ACCOUNTING AND INSURANCE'	36
'BUILDING AND CONSTRUCTION'	51
'CHEMIST, BIOLOGY AND ENVIRONMENT'	41
'BUSINESS ADMINISTRATION'	29
'COMPUTER AND PROGRAMMING'	14
'CUSTOMER SERVICE, SECRETARIAT AND CLERKS'	37
'EDUCATION AND FORMATION'	6
'ENGINEERING AND TECHNICIANS'	103
'HEALTH, SOCIAL SERVICES AND CARE'	30
'HUMAN RESOURCES'	1
'LANGUAGES'	0
'LAWYERS'	7
'LEISURE AND ENTERTAINMENT'	6
'MARKETING, COMUNICATION AND PUBLISHING'	10
'MECHANICS AND MAINTENANCE'	89
'MULTIMEDIA, DESIGNERS AND ARTISTS'	11
'OTHERS'	0
'PRODUCTION, QUALITY AND R&D'	19
'SECURITY'	57
'SHOPPING, LOGISTICS AND STOCK'	13
'SPORTS AND BEAUTY '	0
'TOURISM, HOTEL AND CATERING'	0
'TRADE AND SALES'	12
'TRANSPORT'	22
'WORKERS AND OPERATORS'	82

Tabla 2. Muestras por categoría para etiquetado manual

Junto con la dificultad de agrupar de manera razonable las muestras, se ha comprobado que utilizando estos datos no era posible entrenar un clasificador automático que consiguiera rendimientos aceptables. Ante esta dicotomía se ha optado por buscar datos que puedan ser más representativos y ayuden a crear un modelo más consistente.

4.4 Descripciones de puestos laborales

Para la obtención de datos adicionales se han utilizado técnicas de ‘*web scraping*’ sobre un portal web que contiene más de 12000 descripciones de empleos³. Las descripciones se corresponderían con la información que estábamos extrayendo del campo ‘*Duties*’ de los boletines de trabajo. Por tanto, dicha información será utilizada para construir un clasificador fiable. No obstante, a la hora de interactuar con el usuario se filtrarán las ofertas contenidas en los boletines de trabajo etiquetadas a mano; pudiendo previamente refinar el etiquetado manual en función de las categorías que finalmente se utilicen de acuerdo a la disponibilidad de los datos encontrados.

La extracción de datos ha sido semi-guiada, consistiendo en los siguientes pasos:

1. Se obtienen todos los hipervínculos con enlace a las páginas de descripciones laborales.
2. Se proponen varias claves de búsqueda como candidatos a formar las distintas categorías del *dataset*. Para ello:
 - Se toma una lista de palabras (‘*account*’, ‘*clerk*’, ‘*construction*’, etc.).
 - Se obtiene la lista de enlaces donde aparece esas palabras.
 - Se seleccionan aquellos candidatos que consiguen agrupar un número mínimo de ejemplos y se añaden a una lista.
3. Se crea una carpeta por cada categoría seleccionada. En cada carpeta se crea un documento de texto por cada ejemplo asignado a la categoría:
 - Se abre cada enlace almacenado en la lista.
 - Se detectan las etiquetas *html* que contienen el nombre de la profesión y la descripción de la misma.
 - Se crea un documento de texto con el nombre de la profesión y el campo ‘*Dutie*’ siguiendo la estructura de los boletines de trabajo utilizados hasta el momento. Se ha mantenido la estructura original para posteriormente poder utilizar el mismo script de extracción de características.

A pesar de disponer de más de 12000 descripciones, tan solo conseguimos encontrar 13 categorías con más de 70 muestras (tabla 3). Esto pone de manifiesto lo complicado que es poder llegar a distinguir un gran número de categorías dada la poca diversidad de los datos. Se han añadido al *dataset* otras 5 categorías que a pesar de disponer de un número

³ <https://dot-job-descriptions.careerplanner.com/>

reducido de muestras, parecen suficientemente distintivas como para poder ser identificadas con facilidad.

Categoría	Número de muestras
'CLERK'	273
'COLOR_PAINTER'	85
'DIRECTOR_MANAGER'	115
'DRIVER'	53
'EDUCATION_TEACHING_TRAINING_INSTRUCTOR'	78
'ENGINEER'	173
'GUARD_OFFICER_POLICE_SECURITY'	75
'HEALTH_NURSE'	48
'MAINTENANCE'	55
'MECHANIC'	142
'OPERATOR'	304
'PHOTO'	75
'PRINTER'	125
'SALES'	190
'SHOPPING'	44
'STOCK'	46
'TECHNICIAN'	154
'WORKER'	236

Tabla 3. Muestras por categoría para etiquetado semi-automático

Si bien el agrupamiento del total de las categorías aún es bastante mejorable, se observa como los perfiles laborales comienzan a agruparse de forma más concentrada.

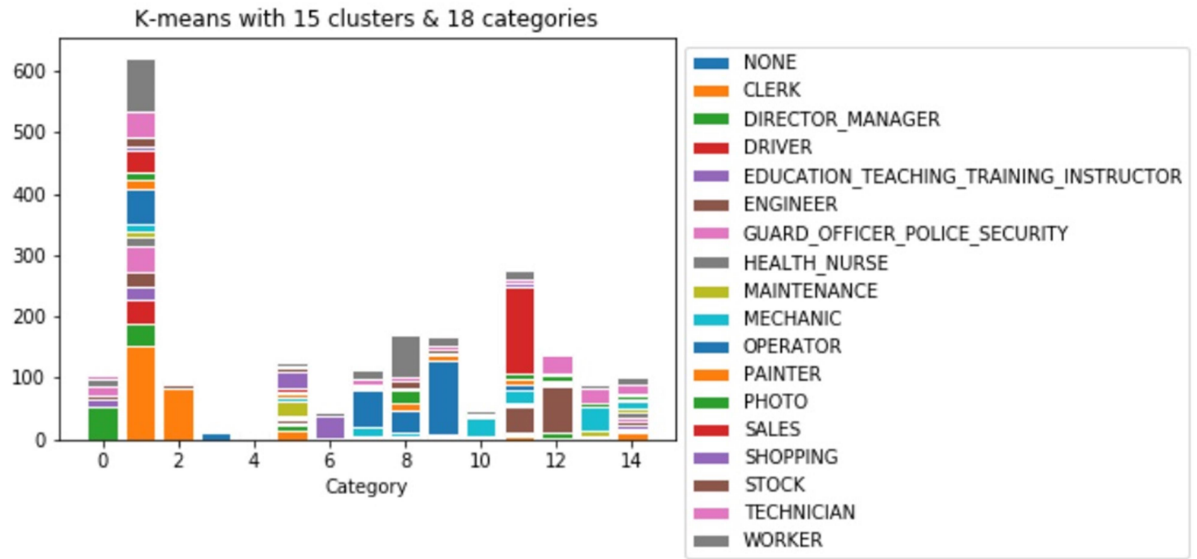


Ilustración 5. Clusters para dataset extendido

Para el caso en que seleccionamos 8 categorías tras realizar un cribado que se justificará en las siguientes secciones, podemos observar como el resultado del *clustering* es bastante aceptable. Lo cual, veremos que va en línea con rendimientos aceptables del clasificador automático que se presentará también en las siguientes secciones.

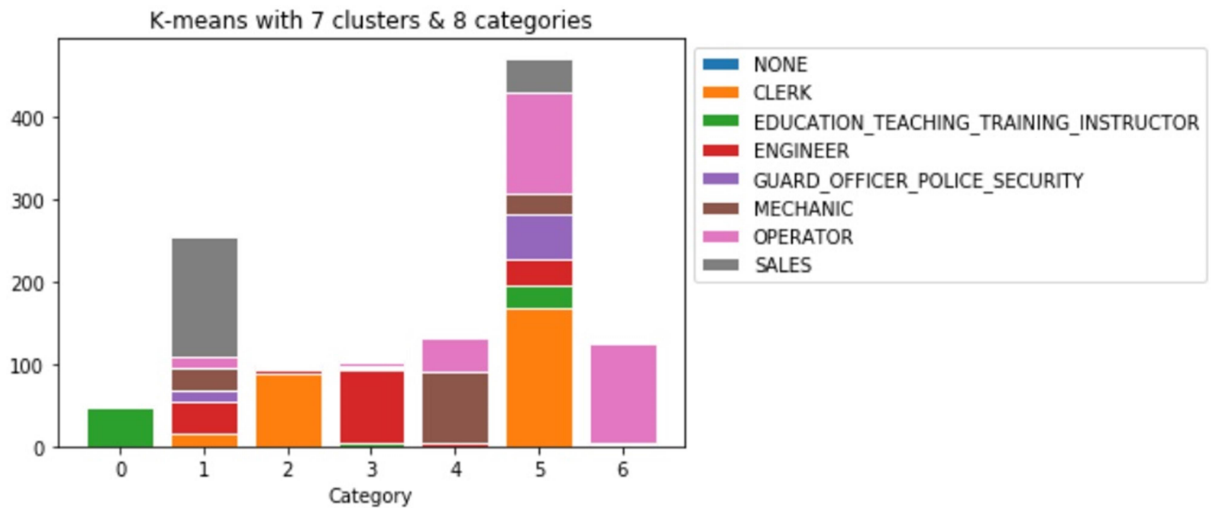


Ilustración 6. Clusters para dataset acotado

5. Desarrollo del trabajo

En este apartado, se detallan cada uno de los desarrollos realizados junto con los experimentos elaborados para verificar la funcionalidad y aplicabilidad de cada módulo del sistema. Dichos experimentos, se acompañan siempre de un análisis detallado que incide en las ventajas y contingencias inherentes a cada uno de los desarrollos.

Todas las rutinas de procesamiento de datos y algoritmos de aprendizaje automático han sido programados en *Python* haciendo uso de *Jupyter notebook*. Mediante su interfaz web, esta API nos permite crear scripts que facilitan intercalar texto, imágenes, tablas y gráficas de forma ágil e intuitiva. De esta forma, resulta muy cómodo y directo seguir el flujo de datos y comprender cómo se han realizado las implementaciones paso a paso. El entorno de desarrollo (IDE) puede descargarse desde la página web oficial (<https://jupyter.org/>). La mínima versión de las librerías necesaria para ejecutar las rutinas son las siguientes:

```
Library versions
Numpy: 1.16.3           Pandas: 0.23.4
Scikit-Learn: 0.20.3   Keras: 2.2.4
TensorFlow: 1.13.1
```

5.1 Procesado de datos

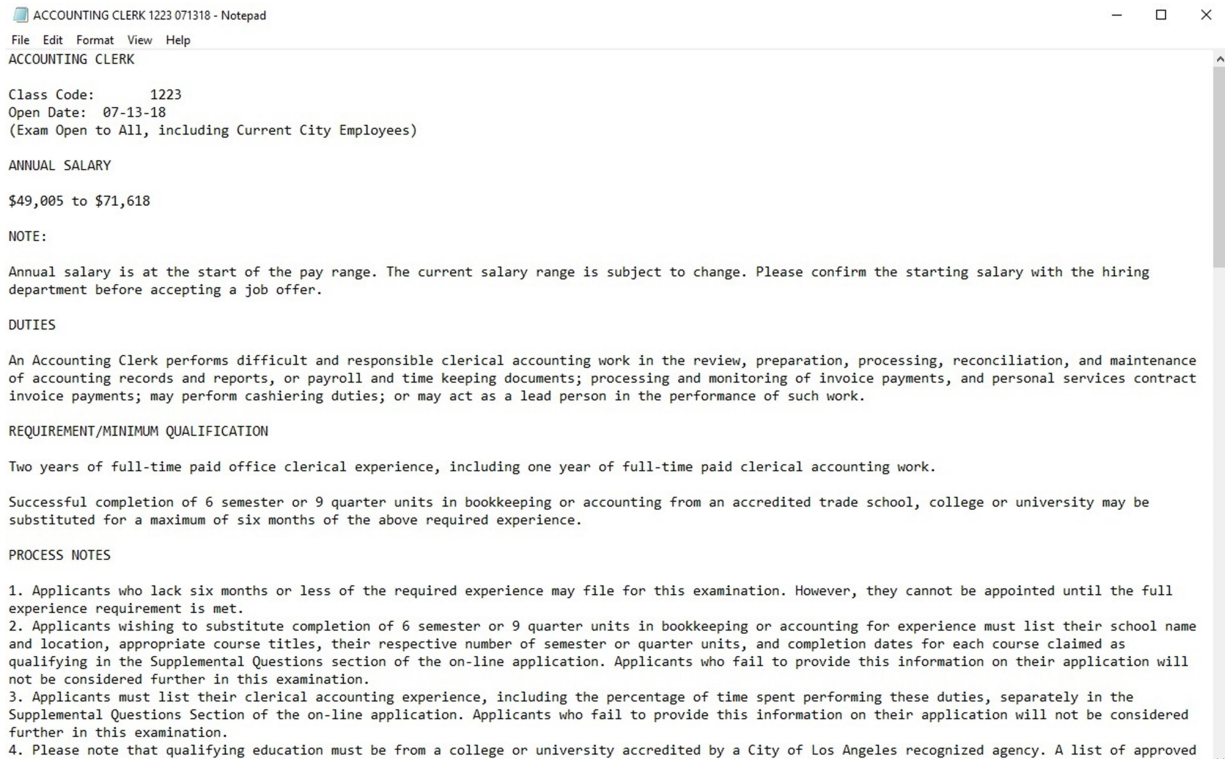
En este y en los siguientes apartados, tratamos la importante tarea de extraer valor de los datos. La herramienta sobre la que basaremos los desarrollos de esta sección es el *Natural Language Toolkit* (NLTK) trabajando sobre el corpus de *Wordnet* [7].

```
#NLTK
import nltk #Natural Language Toolkit
from nltk.corpus import wordnet as wn #Wordnet corpus
```

Ilustración 7. Importación del NLTK en *Python*

5.1.1 Secciones

Para cada boletín de trabajo, se obtiene el nombre de la oferta laboral y el texto de las secciones 'Duties', 'Requirement' y 'Salary'. Partimos de documentos como el que se muestra a continuación:



```

ACCOUNTING CLERK 1223 071318 - Notepad
File Edit Format View Help
ACCOUNTING CLERK

Class Code:      1223
Open Date:      07-13-18
(Exam Open to All, including Current City Employees)

ANNUAL SALARY

$49,005 to $71,618

NOTE:

Annual salary is at the start of the pay range. The current salary range is subject to change. Please confirm the starting salary with the hiring department before accepting a job offer.

DUTIES

An Accounting Clerk performs difficult and responsible clerical accounting work in the review, preparation, processing, reconciliation, and maintenance of accounting records and reports, or payroll and time keeping documents; processing and monitoring of invoice payments, and personal services contract invoice payments; may perform cashiering duties; or may act as a lead person in the performance of such work.

REQUIREMENT/MINIMUM QUALIFICATION

Two years of full-time paid office clerical experience, including one year of full-time paid clerical accounting work.

Successful completion of 6 semester or 9 quarter units in bookkeeping or accounting from an accredited trade school, college or university may be substituted for a maximum of six months of the above required experience.

PROCESS NOTES

1. Applicants who lack six months or less of the required experience may file for this examination. However, they cannot be appointed until the full experience requirement is met.
2. Applicants wishing to substitute completion of 6 semester or 9 quarter units in bookkeeping or accounting for experience must list their school name and location, appropriate course titles, their respective number of semester or quarter units, and completion dates for each course claimed as qualifying in the Supplemental Questions section of the on-line application. Applicants who fail to provide this information on their application will not be considered further in this examination.
3. Applicants must list their clerical accounting experience, including the percentage of time spent performing these duties, separately in the Supplemental Questions Section of the on-line application. Applicants who fail to provide this information on their application will not be considered further in this examination.
4. Please note that qualifying education must be from a college or university accredited by a City of Los Angeles recognized agency. A list of approved

```

Ilustración 8. Muestra del *dataset* de boletines de trabajo

La rutina "010_Dataset_bulletins_reader" almacena el contenido seleccionado en una variable tipo lista ([figura 9](#)).

```

list_job_att=[]      #Attribute list for each job // [0]=Duties List,[1]=Requirements List,[2]=Salary List

1  index=2
2  print(job_filenames[index])
3  print('\nDUTIES:',list_job_att[0][index])
4  print('\nREQUIREMENTS',list_job_att[1][index])
5  print('\nSALARY',list_job_att[2][index])

ACCOUNTING RECORDS SUPERVISOR 1119 072718.txt

DUTIES: ['An Accounting Records Supervisor assigns, reviews and evaluates the work of a group of clerical employees engaged in the preparation, processing, and maintenance of a wide variety of accounting or financial records; or supervises accounting clerical personnel serving as cashiers or tellers; applies sound supervisory techniques in building and maintaining an effective work force; and fulfills equal employment opportunity responsibilities.\n']

REQUIREMENTS ['Two years of full-time paid experience as an Accounting Clerk, or in a class at the level of Senior Administrative Clerk (formerly Senior Clerk) which provides experience in office clerical accounting or cashiering.\n']

SALARY ['$55,332 to $80,930 and $65,145 to $95,254\n']

```

Ilustración 9. Atributos de los boletines de trabajos almacenados en listas

Tal y como se comentó en el la sección [4.3 Etiquetado automático del dataset](#), la escasez y falta de representatividad de los datos hizo necesario la ampliación de los datos referentes a las descripciones laborales; utilizadas para entrenar el clasificador. Para este cometido, se han empleado técnicas de *web scraping* según la metodología descrita en la sección [4.4. Descripciones de puestos laborales](#). La rutina “012_Web_scraping” convierte las descripciones publicadas en la web de origen ([figura 10](#)) al formato utilizado en los boletines de trabajo ([figura 11](#)). De esta forma “010_Dataset_descriptions_reader” puede reutilizar las funciones del primer script para generar la lista de atributos ‘Duties’.

Ilustración 10. Formato original (web) de las descripciones de trabajo

Ilustración 11. Muestra del dataset de descripciones laborales

5.1.2 Etiquetado

Como primera aproximación al etiquetado de los boletines de trabajo, se trató de asignar las categorías profesionales en función de las palabras más frecuentes presentes en los nombres de los propios boletines (sección [4.3.1 En función del nombre del boletín](#)). Tras constatar que la distribución obtenida no era suficientemente representativa se realizó un etiquetado manual. Ambos procedimientos han sido programados en la rutina “011_Dataset_group_by_name” y sus resultados se almacenaron en los diccionarios “dict_job_group” y “dict_job_group_tagged”. Estos diccionarios incluyen el nombre de la categoría y un identificador numérico.

```
{'NONE': 0, 'CLERK': 1, 'DIRECTOR_MANAGER': 2, 'DRIVER': 3,
'EDUCATION_TEACHING_TRAINING_INSTRUCTOR': 4, 'ENGINEER': 5,
'GUARD_OFFICER_POLICE_SECURITY': 6, 'HEALTH_NURSE': 7, 'MAINTENANCE': 8,
'MECHANIC': 9, 'OPERATOR': 10, 'PAINTER': 11, 'PHOTO': 12, 'SALES': 13,
'SHOPPING': 14, 'STOCK': 15, 'TECHNICIAN': 16, 'WORKER': 17}
```

En el caso del *dataset* obtenido mediante *web scraping*, las categorías se conocen en el momento de la extracción y son asignadas directamente en la rutina “010_Dataset_descriptions_reader”.

5.1.3 Clustering

Las técnicas de agrupamiento (*clustering*) han sido utilizadas en este trabajo con dos objetivos:

- Realizar una segmentación automática de los documentos en categorías.
- Verificar si los algoritmos propuestos para el etiquetado automático aportan resultados consistentes.

La rutina de código “030_Clustering_by_N” utiliza la implementación del algoritmo de *clustering k-means* presente en la librería de *Python Scikit-Learn*. Este algoritmo procesa el vector de características de cada instancia y utiliza una medida de similitud (distancia euclidiana, distancia de Manhattan, etc.) para agrupar las muestras. El número *n* de categorías ha de ser definido previo al entrenamiento del algoritmo. El algoritmo se compone de dos fases. En la primera se escoge aleatoriamente una muestra como centroide de cada categoría. A partir de este punto, de forma iterativa el algoritmo asigna las muestras al grupo cuyo centroide esté más cercano y recalcula los centroides hasta que la solución converja.

Para tratar de hacer el análisis más intuitivo el resultado se muestra en forma de gráfica con la frecuencia de asignación de las muestras a los n grupos propuestos. El análisis visual ha servido para corroborar el reducido valor obtenido en las métricas de evaluación (coeficiente de *Silhouette* y el índice de *Calinski-Harabaz*) de la agrupación resultante. Por este motivo, se descartó esta técnica para el etiquetado automático (ver [4.3.2 Clustering en función de la descripción de la oferta laboral](#)).

5.2 Modelos de aprendizaje automático

En todo problema que se pretende abordar mediante el uso de técnicas de aprendizaje automático se deben cubrir una serie de etapas, las cuales resumimos en los siguientes puntos:

1. Recolección de datos
 - Siendo fundamental que los datos sean representativos y diversos.
2. Preparación de datos
 - Selección de variables más predictivas, evitar correlaciones, reducción de dimensionalidad y división de los datos en conjunto de entrenamiento y validación.
3. Elección del modelo
 - Prueba de distintos modelos/algoritmos de aprendizaje automático.
4. Entrenamiento
 - El modelo procesa los datos y compara la salida generada con la etiqueta de cada muestra (aprendizaje supervisado) para ajustar su configuración interna.
5. Evaluación
 - Utilización de métricas de rendimiento para comprobar la precisión y ratio de acierto del modelo.
6. Ajustes finales
 - Ajuste de parámetros para mejorar el rendimiento.

En el desarrollo del presente trabajo se van a utilizar y comparar los resultados de dos tipos de modelos de aprendizaje automático: las redes bayesianas y las redes neuronales. Los primeros se han seleccionado como técnica preferente debido a su demostrada eficacia y extendido uso para la clasificación de texto [26]. Los segundos se utilizarán por ser uno de los modelos más potentes y flexibles, que por tanto son siempre buenos candidatos a utilizar para tratar de obtener rendimientos elevados. Si bien, cabe mencionar que este tipo de

modelos no suelen destacar cuando se trabaja con un conjunto de datos muy disperso.

Las rutinas “040_Classifier_by_NB” y “040_Classifier_by_NN” se encargan de entrenar y evaluar cada uno de los modelos respectivamente. En los respectivos scripts se detalla el proceso de ajuste de parámetros y evaluación de rendimientos.

5.2.1 Métricas de rendimiento

Para problemas de clasificación multiclase, la exactitud (*accuracy*) se calcula mediante fórmulas derivadas de la correspondiente a la clasificación binaria; mostrada a continuación:

$$accuracy = \frac{VP + VN}{VP + VN + FP + FN}$$

siendo *VP* verdaderos positivos, *VN* verdaderos negativos, *FP* falsos positivos y *FN* falsos negativos. Adicionalmente, se utilizan distintas técnicas de promediado (*micro/macro average*) sobre otros indicadores habituales tales como “*precision*”, “*recall*” y “*f-measure* o *f1*” [27].

$$precision = \frac{VP}{VP+FP} \quad recall = \frac{VP}{VP+FN} \quad f1 = \frac{2 \cdot precision \cdot recall}{precision+recall}$$

Respecto a la evaluación de redes neuronales, la función de pérdidas (función a optimizar) aporta información sobre el posible margen de mejora que aún tiene el modelo. La función pérdidas más adecuada la evaluación de problemas de clasificación multiclase es la *Categorical Cross-Entropy*, que resulta de la combinación de una función *Softmax* y otra de tipo *Cross-Entropy*:

$$f(s)_i = \frac{e^{s_i}}{\sum_j^C e^{s_j}} \quad CE = -\sum_i^C t_i \log(f(s)_i)$$

siendo *C* el número total de categorías a clasificar, *s_j* la puntuación que el modelo asigna a la pertenencia a cada categoría y *t_i* la categoría real a la que pertenece la muestra. Al ser *t_i* un vector que contiene uno en la posición de la categoría a real y cero en el resto, podemos reescribir la ecuación como sigue:

$$CE = -\log\left(\frac{e^{S_p}}{\sum_j^C e^{S_j}}\right)$$

donde *S_p* es la probabilidad para la clase positiva. A medida que el modelo mejora la pérdida

se reduce. No obstante, es preciso tener controlado esta medida ya que pérdidas muy bajas pueden ser indicio de que el modelo ha caído en sobreajuste (*overfitting*). En estas situaciones, el modelo reduce su capacidad de generalización (acertar datos nunca vistos) ya que lo que acaba haciendo es memorizar los datos. Para evitar estas situaciones se utilizan técnicas de regularización.

5.2.2 División del conjunto de datos

Antes de proceder al entrenamiento debemos dividir los datos disponibles en tres conjuntos distintos:

- Conjunto de ajuste (80%): datos para la fase de entrenamiento
 - Entrenamiento (80%): el modelo procesa los datos de entrada tratando de predecir la salida asociada. A partir de la diferencia entre la salida esperada (real) y la predicha, el modelo va ajustando sus parámetros internos para tratar de aprender el patrón inherente a los datos.
 - Validación (20%): datos que el modelo no procesa directamente y que son utilizados para realizar predicciones con el objetivo de determinar cómo evoluciona la precisión durante la fase de entrenamiento. Con ello, se puede apreciar cómo afectan al rendimiento distintas configuraciones del modelo o tratamiento de los datos de entrada.
- Conjunto de test o evaluación (20%): datos no vistos por el modelo sobre los que se realizan predicciones. Si el rendimiento es adecuado el modelo estará listo para entrar en producción. Un modelo con precisión por debajo de 50% no tiene capacidad predictiva; lo cual implica que no tendrá la habilidad de acertar muestras o ejemplos nunca vistos durante la fase de entrenamiento. Para que un modelo pueda entrar en producción con suficiente fiabilidad deberíamos rondar rendimientos entorno al 85-90%. Modelos con rendimientos inferiores deberán ser acompañados de mecanismos de evaluación y entrenamiento progresivo hasta que su rendimiento alcance niveles aceptables.

5.2.3 Clasificador bayesiano

Para la implementación del clasificador bayesiano se han utilizado los distintos modelos disponibles en la librería *Scikit-Learn*, junto con rutinas auxiliares de código programados en

Python. En la mencionada librería encontramos cuatro implementaciones que son variantes adaptadas o particularizadas para distintos casos de uso. Todas estas implementaciones se acogen a la suposición de independencia condicional entre los eventos a clasificar (*naive*), la cual se utiliza para simplificar la complejidad computacional del algoritmo:

- Gaussian NB: asume que la verosimilitud presenta una distribución gaussiana.
- Multinomial NB: variante para datos que siguen una distribución multinomial. Esta implementación está directamente enfocada a la clasificación de texto en función de la matriz de frecuencia de palabras. Aunque también puede dar resultados aceptables al trabajar con la matriz *tf-idf*.
- Complement NB: adaptación de la implementación *Multinomial NB* para conjuntos de datos desbalanceados.
- Bernoulli NB: asume que los datos se acogen a una distribución multivariante de Bernoulli.

Aunque la variante *Multinomial NB* está directamente enfocada a la clasificación de texto, en todas las pruebas se evalúan siempre los cuatro modelos, tomando en cada ocasión aquel que aporte mejor rendimiento. La matriz de frecuencia de términos (ver [2.2.4 Clasificación de texto](#)), será la entrada a procesar por los algoritmos. Además del procesado directo de la matriz, se han programado dos variantes: una en la que se reduce la frecuencia de aparición de cada palabra para que siga una distribución binomial (aparece (1) /no aparece (0)) y otra en la que cada columna es escalada en el rango [0,1] teniendo en cuenta el valor máximo encontrado en el conjunto de todas las muestras.

Features	Modelo	Procesado de la matriz de frecuencia	Rendimiento validación	Rendimiento test
N: 298	Multinomial NB	Distribución binomial	0.6763	0.6821
N: 298	Multinomial NB	Escalado de características	0.6731	0.6897
N: 298	Multinomial NB	Ninguno	0.7019	0.7103

Tabla 4. Comparativa de modelos bayesianos

A continuación se muestra la salida del algoritmo de evaluación, el cual muestra el rendimiento sobre el conjunto de validación, test, los resultados de una validación cruzada (*cross-validation*) sobre cinco subdivisiones y el tiempo de entrenamiento.

```
NB type: MULTINOMIALNB
NB accuracy on VALIDATION 0.7019
NB accuracy on TEST 0.7103
NB accuracy on cross-validation [0.6786 0.7125 0.6883 0.6667 0.6486]
```

5.2.4 Clasificación mediante redes neuronales

Para la clasificación mediante redes neuronales se ha utilizado *Keras*, siendo esta una API de alto nivel construida en *Python* para el *framework Tensorflow* desarrollado por Google. Se puntualiza que las funciones de activación utilizadas en las redes neuronales fuerzan el escalado de las variables de entrada en el rango $[0,1]$. Por ello, esta opción se ha aplicado por defecto sobre la matriz de frecuencias para las pruebas con este tipo de modelos.

```

TRAINING TIME: 6.8746

VALIDATION dataset
Accuracy: 0.6915 Loss: 1.5132
TEST dataset
Accuracy: 0.6738 Loss: 1.4188

```

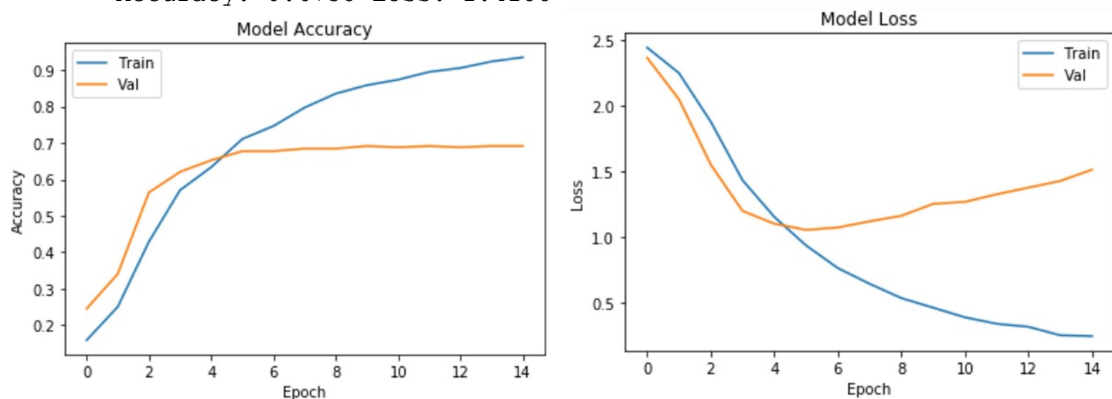


Ilustración 12. Precisión y función de pérdidas para modelos de redes neuronales

La arquitectura de la red se compone de 6 capas cuyo número de neuronas se adapta en función del número de *features* de entrada (*inputs*) y el número de categorías a clasificar (*outputs*). La capa de entrada incluye siempre una neurona por cada variable de entrada ($n_{L1}=inputs$). Las capas intermedias (ocultas) realizan primero una suave expansión del número de neuronas de entradas para luego converger suavemente hacia el número de neuronas en la capa de salida ($n_{L2}=1.2 \cdot inputs$, $n_{L3}=0.6 \cdot inputs$, $n_{L4}=0.4 \cdot inputs$, $n_{L5}=\max(0.2 \cdot inputs, 1.4 \cdot outputs)$), el cual se corresponde con el número de categorías a clasificar ($n_{L6}= outputs$). Con el objetivo de reducir el posible sobreajuste (*overfitting*), se añaden capas intermedias de regularización. Por otro lado, las funciones de activación son de tipo “Relu” en capas intermedias y “Softmax” en la capa de salida, lo cual consigue que el algoritmo de como resultado la probabilidad de que cada clase esté asociada a la muestra que se procesa. La inicialización de los pesos sigue la distribución propuesta por, habiéndose optado por el método de optimización “Adam” con función de pérdida “Categorical Cross-Entropy” y “Early-Stopping” sobre el conjunto de validación como criterio de parada en el entrenamiento. Se adjunta en cada prueba también la matriz de confusión, la cual se utiliza para la evaluación individual del rendimiento en la predicción de cada una de las categorías.

5.3 Selección de características y rendimiento de clasificadores

La calidad y precisión en el proceso de extracción y selección de características determina en gran medida (especialmente cuando se parte de datos no estructurados) el rendimiento alcanzable por los modelos de clasificación. Es por ello, que en esta sección se describen los distintos procesos de selección de características junto con los resultados obtenidos por los clasificadores en cada uno de los enfoques implementados. El análisis de los resultados obtenidos en cada fase nos guiará hacia las modificaciones que se realizarán en cada etapa posterior con el objetivo de incrementar la calidad de las predicciones producidas por los clasificadores

5.3.1 Analizador sintáctico

El análisis sintáctico está encaminado a determinar la función y relación que guardan las palabras o grupos de palabras presentes en una oración. Dicho análisis nos permitirá interpretar correctamente el significado de los documentos que se analizan. Más concretamente, nos asistirá en la tarea de detectar los constituyentes sintácticos o sintagmas⁴ que poseen significado directo: sintagma nominal y sintagma verbal. El núcleo de los sintagmas nominales son los nombres o sustantivos, los cuales hacen referencia a una entidad concreta. Por su parte, el núcleo de los sintagmas verbales son los verbos, empleados para designar estados, procesos o relaciones entre entidades.

La mayoría de técnicas empleadas para el análisis de texto (en particular para detección de tópicos en documentos) se basan en la frecuencia de aparición de los constituyentes sintácticos. Es por ello, que el análisis sintáctico aquí desarrollado estará enfocado a filtrar los textos para separar nombres y verbos del resto de categorías gramaticales (adjetivos, adverbios, determinantes, conjunciones, interjecciones, preposiciones y pronombres) para proceder a su posterior conteo.

⁴ Según la RAE, “palabra o conjunto de palabras que se articula en torno a un núcleo y que puede ejercer alguna función sintáctica”. <https://www.rae.es/>

El procesado de datos que se va a realizar se corresponde con la secuencia de tareas presentada en la [figura 13](#). [9]:



Ilustración 13. Pipeline de procesamiento del lenguaje natural

1. Pre-procesado: consiste en eliminar una serie de símbolos (tales como saltos de línea: '\n') que los procesadores de texto suelen incluir en los documentos para su correcta visualización.
 - Función de usuario implementada en *Python*
2. Obtención de *tokens*: se identifican las palabras (*tokens*) que conforman las oraciones. Las formas gramaticales que aparezcan contraídas se desarrollan y se separan (ej. 'del' pasaría a ser 'de' + 'el').
 - La función `word_tokenize(sentence)` del NLTK devuelve una lista con los *tokens* de cada palabra de la frase de entrada
3. Etiquetado morfosintáctico (POS): consiste en identificar las diferentes partes de la oración y asignar la categoría gramatical a cada palabra presente en la frase.
 - La función `pos_tag(tokens)` del NLTK devuelve una lista de tuplas (token,POS) para cada *token* de la lista de entrada
4. Análisis sintáctico (*parser*): se utiliza una gramática (conjunto de reglas formales) para procesar los *tokens* y obtener una división en las distintas subestructuras o segmentos (*chunks*) agrupados según las especificaciones de las reglas de la gramática utilizada. En la [figura 14](#) se presenta un ejemplo del proceso de etiquetado a nivel de *tokens* y de segmentos.

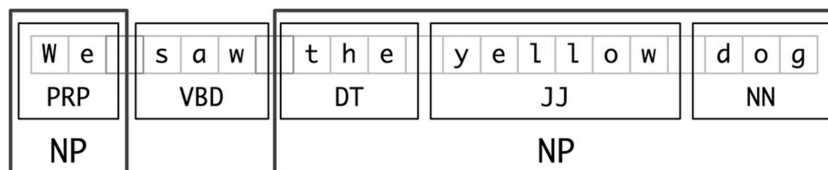


Ilustración 14. Ejemplo del proceso de etiquetado de *tokens* y segmentos para sintagmas nominales

Fuente: Bird, S., Klein, E., & Loper, E. (2009)

La gramática utilizada para el ejemplo anterior incluiría al menos la siguiente regla para la detección de un sintagma nominal (*Nominal Phrase*):

NP: <DT>?<JJ.*>*<NN.*>+

La cual se lee como cualquier secuencia de *tokens* que comience con un determinante (<DT>) opcional (<DT>?), seguido de cero o más (<JJ.*>*) adjetivos (<JJ>) de cualquier tipo (<JJ.*>; JJ, JJR, JJS), finalizando con uno o más (<NN.*>+) nombres (<NN>) de cualquier tipo (<NN.*>; NN, NNP, NNS, NNPS).

A continuación se enumeran todas las etiquetas utilizadas por el analizador del NLTK:

CC coordinating conjunction; CD cardinal digit; DT determiner; EX existential there; FW foreign word; IN preposition/subordinating conjunction; JJ adjective; JJR adjective, comparative; JJS adjective, superlative; LS list marker, 1); MD modal could, will; NN noun, singular; NNS noun plural; NNP proper noun, singular; NNPS proper noun, plural; PDT predeterminer; POS possessive ending parent's; PRP personal pronoun I, he, she; PRP\$ possessive pronoun my, his, hers; RB adverb; RBR adverb, comparative; RBS adverb, superlative; RP particle; TO, to go 'to'; UH interjection; VB verb, base form; VBD verb, past tense; VBG verb, gerund/present participle; VBN verb, past participle; VBP verb, sing. present, non-3d; VBZ verb, 3rd person sing. Present; WDT wh-determiner; WP wh-pronoun; WP\$ possessive wh-pronoun; WRB wh-abverb

La forma más típica de presentar el resultado del análisis sintáctico es mediante árboles sintácticos. Esta es la forma utilizada por el NLTK. Opcionalmente se puede utilizar el formato IOB⁵.

Concretando a nuestro caso de estudio, analizando las descripciones de los perfiles laborales se observó que la enumeración de las competencias y habilidades profesionales se presentaba habitualmente a través de un verbo transitivo y su correspondiente complemento directo sobre el que recae la acción del sujeto. Mediante el uso de gramáticas como las descritas en el ejemplo anterior y un procesado posterior se consiguió extraer los pares (verbo, complemento). En la [figura 15](#) se muestran varios ejemplos obtenidos de los boletines de trabajo:

⁵ I (inside, etiquetas no iniciales del segmento), O (outside, etiquetas que no pertenecen al segmento), B (begin, etiqueta inicial del segmento)

```
('works', 'as an apprentice in a specific metal trade, ')
('receives', 'on-the-job training ')
('assisting', 'a journey-level Sheet Metal Worker, Structural Steel Shop
Worker, Blacksmith, Welder, or Auto Body Builder and Repairer; ')
('attends', "approved classroom instruction during off duty hours.")
```

Ilustración 15. Detección del par verbo transitivo-complemento directo

Este proceso de extracción define con bastante rigurosidad las atribuciones que se exigen a cada perfil laboral. Sin embargo, no es demasiado flexible ni representativa para su tratamiento por la mayoría de algoritmos de aprendizaje automático. El motivo es que cuanto más compleja sea la regla para obtención de segmentos, más variabilidad tendremos en la segmentación y por tanto, vamos a encontrar menos instancias de cada regla. Menos instancias implica menos ejemplos de cada patrón para el entrenamiento de los algoritmos, lo cual nos conducirá a un rendimiento más bajo en las predicciones. Este es probablemente el motivo por el que la mayoría de los algoritmos que se utilizan para determinar el tópicos de un texto se basan directamente en el conteo de palabras, en vez de en la extracción de estructuras gramaticales más complejas. Partiendo de estas pesquisas, se ha decidido utilizar el análisis sintáctico para obtener nombres y verbos, por ser estos los constituyentes sintácticos que contienen la mayor parte del significado de las frases. Las gramáticas utilizadas se muestran a continuación ([tabla 5](#)):

Gramática filtrado de nombres	Gramática para filtrado de verbos
NP: {<N.*>}	VP: {<TO> <V.*>}, {<V.*> <RB> <V.*>} {<V.*> <V.*> <TO>}, {<V.*> <IN>} {<V.*> <RP>}, {<V.*> <VBN>} {<V.*> <VBG>}, {<V.*>+}, {<V.*>}, <TO>{, }<RB> <V.*>{

Tabla 5. Gramáticas para el análisis sintáctico

- o La función `grammar_chunk(grammar, POS)` devuelve en forma de árbol los segmentos encontrados según la gramática propuesta. Se basa en las funciones del NLTK `RegexParser(grammar)` y `parse(POS)`. La primera sirve para construir un analizador sintáctico (`parser`) basado en la gramática indicada. La segunda utiliza el `parser` para segmentar los POS de entrada.

```
def grammar_chunk(grammar, POS_tag):          #Grammar chunking

    cp=nlk.RegexpParser(grammar)              #Built a parser using the defined grammar
    tree_phrase=cp.parse(POS_tag)             #Get chunks from each sentence based on POS

    return tree_phrase
```

Ilustración 16. Rutina de segmentación según gramática en Python

Seguidamente, la función *chunk_lookup(tree_phrase)* se encarga de devolver los constituyentes sintácticos de cada árbol encontrado en el análisis anterior.

5. Obtención del lema: con el objetivo de evitar que las distintas variaciones que puede presentar una palabra (género, número, tiempo verbal) computen como distintas instancias, se procede a obtener la raíz de la palabra (lema) antes de realizar la selección de características.
 - La función *chunk_lemma(chunk,key)* recorre cada una de las palabras del segmento (*chunk*), obtiene su lema en función de que sea nombre o verbo (*key* toma valores 'n' o 'v') utilizando la función del NLTK *lemmatize(chunk_split[i],key)* y devuelve el segmento compuesto por los lemas de las palabras de entrada.
6. Obtención de características: en último lugar se construye un diccionario con cada una de las palabras extraídas acompañada de su frecuencia de aparición en los documentos. Se excluyen palabras muy comunes y verbos no transitivos. El total de características obtenidas para cada *dataset* queda recogido en la siguiente tabla:

Dataset	Características	Nombres	Verbos
ds_B	676 documentos, 2006 nombres, 788 verbos. Ratio n/v=2.5	('work', 905) ('building', 413), ('technique', 323), ... ('transportation', 63), ('computer', 62), ('customer', 61), ... ('docent', 1), ('legislator', 1), ('county', 1),	('maintain', 389) ('sound', 301), ('engage in', 271), ... ('prepare', 54) ('coordinate', 53), ('act as', 51), ... ('call from', 1), ('provide by', 1), ('arise', 1)
ds_D	2271 documentos, 6548 nombres, 4673 verbos. Ratio n/v=1.4	('machine', 2026) ('equipment', 1815), ('record', 939), ... ('traffic', 63), ('lever', 63), ('hopper', 63) ... ('bonderite', 1), ('badminton', 1), ('tint', 1)	('use', 2441) ('determine', 624), ('accord', 527), ... ('secure', 56), ('carry', 56) ('expose', 55), ... ('call from', 1), ('provide by', 1), ('arise', 1)

Tabla 6. Características extraídas y frecuencia de aparición

Donde *ds_B* se corresponde con el *dataset* de boletines de trabajo y *ds_D* con el *dataset* de las descripciones de perfiles profesionales obtenido mediante *web scraping*.

Atendiendo al ratio entre nombres y verbos ([tabla 6](#), *ratio n/v*), se puede apreciar como a medida que tenemos más muestras somos capaces de encontrar más ejemplos para las reglas más complejas (verbos) y por tanto la representatividad de los datos mejora. También resulta bastante directo notar como las palabras más repetidas son a menudo bastante genéricas ([tabla 6](#), 'machine', 'equipment', 'record'). Lo cual redundaría en que dichas características se repitan en categorías completamente diferentes sin aportar información que nos permita diferenciarlas. En las siguientes etapas se planteará algún mecanismo de filtrado que nos permita obviar este tipo de palabras genéricas.

A continuación seleccionamos aquellas características que aparezcan un número de veces igual o mayor a un umbral y calculamos el porcentaje que representan sobre el total de características de su mismo tipo:

Dataset	Umbral	Atributo	Features	Porcentaje	Atributo	Features	Porcentaje
ds_B	10	Nombre	345	17.2	Verbo	71	9.0
	20	Nombre	205	10.2	Verbo	44	5.6
	30	Nombre	145	7.2	Verbo	25	3.2
ds_D	10	Nombre	1638	25.0	Verbo	475	10.2
	20	Nombre	1027	15.7	Verbo	283	6.1
	30	Nombre	775	11.8	Verbo	201	4.3

Tabla 7. Representatividad de características extraídas

Si bien la representatividad de los verbos apenas ha variado de un *dataset* a otro, la de los nombres sí que ha efectuado una notable mejoría. Aprovechamos para reflexionar brevemente sobre el impacto que puede tener el ratio nombre a verbo (*n/v*) a la hora de seleccionar características. Idealmente, a la hora de seleccionar nuestras características, podría ser interesante mantener el ratio *n/v* que se observa en el *dataset*, ya que esto podría implicar ser más fidedignos a la distribución interna de los datos.

Por otro lado, dado que las relaciones de sinonimia posibilitan el empleo de distintas palabras para expresar el mismo contenido, se va a considerar esta circunstancia a la hora de evaluar el número de veces que las características seleccionadas aparecen en cada documento. Distinguiamos principalmente dos tareas:

- Extracción y selección de características
 - Se aplica *pipeline* descrito en este apartado al total de documentos para obtener las características (*features*) descriptivas del *dataset*.

- Caracterización de los documentos (conteo, [figura 40](#))
 - Aplica el *pipeline* a cada documento para filtrar las palabras que coinciden con la categoría sintáctica escogida. Después procede de la siguiente forma:
 1. Se busca coincidencia entre las palabras filtradas y alguna de las palabras que forman el conjunto de características del *dataset*. Se incrementa la columna de la *feature* que coincida.
 2. Si la palabra no se encuentra entre las *features*, se evalúa el conjunto de sinónimos de cada *feature* para comprobar si la palabra fuera alguna variante de las *features* del *dataset*. En caso de encontrar coincidencia, se incrementa el conteo de la *feature* original.
 3. En caso de no encontrar ninguna coincidencia en alguno de los dos grupos anteriores, la palabra no computa en ninguna columna y se incrementa el campo de características no vistas (*unseen*); provisto para la generación de estadísticas sobre el *dataset*.

Esto nos dará como resultado una matriz con la frecuencia de cada característica para cada documento, que habitualmente se conoce en la literatura como *term frequency* (ver [2.2.4 Clasificación de texto](#)).

Jobs	ID	accounting	work	analysis	preparation	maintenance	control	record	report	principle	performs	review	processing
ACCOUNTANT	1513	2	1	1	1	1	1	1	1	1	1	0	0
ACCOUNTING CLERK	1223	3	2	0	1	1	0	1	1	0	1	1	2
ACCOUNTING RECORDS SUPERVISOR	1119	2	2	0	1	1	0	2	0	0	0	1	1
ADMINISTRATIVE ANALYST	1590	0	1	2	0	0	0	0	1	0	1	0	0
ADMINISTRATIVE CLERK	1358	0	2	0	0	0	0	0	0	0	1	0	1
ADMINISTRATIVE HEARING EXAMINER	9135	0	1	0	0	0	0	0	0	0	0	0	0
ADVANCE PRACTICE PROVIDER CORRECTIONAL CARE	2325	0	0	0	0	0	0	0	0	0	0	0	0
AIR CONDITIONING MECHANIC	3774	0	0	0	0	0	2	0	0	0	0	0	0
AIR CONDITIONING MECHANIC SUPERVISOR	3781	0	2	0	0	1	0	0	0	1	0	1	0
AIRPORT AIDE	1540	0	0	0	0	0	0	0	0	0	0	0	0
AIRPORT CHIEF INFORMATION SECURITY OFFICER	1404	0	0	0	0	1	0	0	0	0	0	0	0
AIRPORT ENGINEER	7256	0	2	0	0	1	0	0	0	1	1	0	0
AIRPORT GUIDE	0845	0	2	0	0	0	0	0	0	0	0	0	0
AIRPORT INFORMATION SPECIALIST	1783	0	1	0	0	0	0	0	0	0	0	0	0
AIRPORT LABOR RELATIONS ADVOCATE	9210	0	0	0	0	0	0	0	0	0	0	0	0
AIRPORT MANAGER	7260	0	1	0	0	1	0	0	0	1	0	0	0
AIRPORT POLICE CAPTAIN	3228	0	1	0	0	0	1	0	0	0	0	0	0
AIRPORT POLICE LIEUTENANT	3227	0	1	0	0	0	1	0	0	0	0	0	0
AIRPORT POLICE OFFICER	3225	0	1	0	0	0	0	0	1	0	0	0	0
AIRPORT POLICE SPECIALIST	3236	0	0	0	0	0	0	0	0	0	0	0	0
AIRPORT SUPERINTENDENT OF OPERATIONS	7268	0	1	0	0	2	0	0	0	0	0	1	0
AIRPORTS MAINTENANCE SUPERINTENDENT	3331	0	2	0	0	3	0	0	0	0	0	0	0
AIRPORTS MAINTENANCE SUPERVISOR	3336	0	2	0	0	4	0	0	0	1	0	0	0

Ilustración 17. Matriz de frecuencia de términos

Si atendemos a las dos tareas recién mencionadas, podemos notar como la sinonimia solo se ha considerado a la hora de caracterizar los documentos, pero no en el momento de realizar la selección de características. Esta decisión viene justificada por el siguiente matiz: aunque dos palabras puedan usarse con el mismo significado, se ha considerado que en el ámbito de algunas profesiones sea más común el uso de alguna de sus variantes (que por

hábito costumbre o tradición, forman parte del argot) y que por tanto identificar la variante que más se use podría ayudar al clasificador a diferenciar mejor dos perfiles laborales, especialmente aquellos que sean relativamente similares. No obstante, si al caracterizar los documentos no encontramos ninguna coincidencia, siempre obtendremos más información si evaluamos la posibilidad de que exista alguna relación de sinonimia.

Fijando el umbral a 20 obtenemos las siguientes estadísticas sobre la matriz de frecuencia:

Dataset	Características del dataset	Media de características en documentos	Media de sinónimos en documentos	Media de características no presentes en documentos	Porcentaje de uso
ds_B	Nombres: 205	22.3	0.0	8.9	10.9
ds_B	Nombres: 205	22.3	1.1	7.8	11.4
ds_B	Verbos: 44	4.4	0.0	2.9	10.0
ds_B	Verbos: 44	4.4	0.8	2.2	11.7
ds_D	Nombres: 1027	40.9	0.0	9.3	4.0
ds_D	Nombres: 1027	40.9	2.3	7.0	4.2
ds_D	Verbos: 283	10.7	0.0	5.2	3.8
ds_D	Verbos: 283	10.7	1.7	3.5	4.4

Tabla 8. Uso de sinonimia en la caracterización de las muestras

donde se ha definido el porcentaje de uso como:

$$\% \text{ uso} = (\text{features} + \text{sinónimos}) / \text{features del dataset} * 100$$

Si bien observamos que el porcentaje de uso no aumenta demasiado (no más de un 1%), se adoptará como medida preventiva para aquellos casos en los que la descripción sea muy escueta o no contenga muchas palabras de las que aparecen directamente en las *features* seleccionadas del *dataset*. También podemos notar que al dispararse el número de características seleccionadas, el porcentaje de uso decrece considerablemente, lo cual nos sugiere tratar de buscar alguna estrategia de reducción de la dimensionalidad.

5.3.2 Comparativa de *datasets* y tipo de características

En esta sección se evalúa el rendimiento de los clasificadores para cada uno de los enfoques utilizados en la extracción y selección de características. Dado que el clasificador bayesiano ingenuo (NB) es bastante más rápido de entrenar, se realizarán las pruebas utilizando este modelo. Una vez se detecte la mejor forma de seleccionar las características

se optimizará utilizando redes neuronales (NN) además del modelo bayesiano.

Comenzamos comparando los resultados obtenidos cuando se trabaja con características basadas en nombres (N) y cuando se utilizan como atributos los verbos (V).

Dataset	Features	Sinonimia	Etiquetado	% de uso	Rendimiento validación	Rendimiento test	Modelo
ds_B	V: 44	No	Manual:22	10.0	0.2174	0.1961	NB
ds_B	V: 44	Sí	Manual:22	11.7	0.2087	0.1863	NB
ds_D	V: 283	No	Dirigido:19	3.8	0.4588	0.4923	NB
ds_D	V: 283	Sí	Dirigido:19	4.4	0.4588	0.4879	NB
ds_B	N: 205	No	Manual:22	10.9	0.4957	0.4412	NB
ds_B	N: 205	Sí	Manual:22	11.4	0.513	0.4412	NB
ds_D	N: 1027	No	Dirigido:19	4.0	0.5549	0.6044	NB
ds_D	N: 1027	Sí	Dirigido:19	4.2	0.5522	0.6066	NB

Tabla 9. Comparativa de resultados sobre el *dataset* original y el extendido

Analizando los resultados, se puede destacar la identificación de dos requisitos mínimos para un correcto funcionamiento de los algoritmos de clasificación: disponer de un número suficiente de muestras y poder ser capaces de extraer un número mínimo de características. El primer requisito queda reflejado en la mejoría que se observa al pasar del *dataset* de boletines de trabajo (*ds_B* con 676 muestras) al *dataset* de descripciones de trabajo (*ds_D* con 2271 ejemplos): las pruebas con verbos mejoran un 30% y las pruebas con nombres un 15%. El segundo requisito se extrapola de la primera prueba realizada sobre *ds_B* con verbos, en la que al disponer solo de 44 *features* el modelo no es capaz de captar los patrones en los datos de entrenamiento. Si bien pudiera inicialmente pensarse que se debe al tipo de característica utilizada (verbos), observamos que en las pruebas con *ds_D* la clasificación basada en verbos aumenta el rendimiento relativo considerablemente.

Trabajando sobre el mismo número de muestras, mediante uso de nombres se consiguen rendimientos mayores que cuando las predicciones están basadas en verbos. Sin embargo, ha de anotarse que esto viene también condicionado por el hecho de que de forma natural, en promedio se pueden extraer muchos más nombres que verbos de los textos; ya que también puede apreciarse como cuando el número de *features* es similar (*ds_D* con 283 verbos y *ds_B* con 205 nombres) el rendimiento mediante el uso de verbos puede superar al que se obtiene con los nombres.

Respecto al etiquetado, se ha optado por la asignación manual para *ds_B* porque el

etiquetado automático no conseguía distribuir las etiquetas de forma coherente (ver sección [4.3 Etiquetado automático del dataset](#)). Para *ds_D* se ha utilizado la asignación dirigida (semi-automática) ya que se tuvo que hacer uso de ella para la obtención de los datos (sección [4.4. Descripciones de puestos laborales](#)).

En último lugar se puede corroborar que el uso de la sinonimia apenas mejora los resultados que se obtienen. No obstante, se va a mantener su uso debido a los motivos esgrimidos en la sección anterior.

En su conjunto, estas reflexiones no conducen a adoptar las siguientes decisiones de cara a trabajar sobre ellas en las siguientes etapas:

- Se continuarán las pruebas sobre el *dataset* de descripciones laborales (*ds_D*) por disponer de más número de muestras y estar éstas además mejor etiquetadas.
- Nos basaremos en la extracción de características basadas en nombres por ser estos más abundantes y diversos, factor que favorece una mayor representatividad de los documentos y reduce el posible solapamiento entre las distintas categorías en comparación con el uso de verbos. Además, se ha constatado que la combinación de nombres y verbos se estanca en un porcentaje intermedio, no siendo por tanto una estrategia más favorable.

5.3.3 Reducción de la dimensionalidad

Hasta el momento el mayor rendimiento obtenido ronda el 60%. Sin embargo, este nivel de precisión no es suficiente para que el sistema analítico pueda realizar inferencias de forma adecuada en un entorno de producción. Por ello, nos disponemos a investigar algunos enfoques que ayuden a mejorar la representatividad de las características extraídas.

En primer lugar se va a tratar de conseguir una reducción de la dimensionalidad. Se busca desarrollar un proceso que obtenga resultados similares a las técnicas de PCA (*Principal component analysis*) [10]. Hasta el momento, el proceso de extracción y selección de características se ha realizado sobre el total de documentos. Podría entonces darse la situación de que palabras poco comunes para una categoría sean seleccionadas porque al considerar todos los documentos en su conjunto aparecen un elevado número de veces. Esto estaría provocando que palabras realmente relevantes o concretas para cada categoría acaben siendo desplazadas por algunas más genéricas, haciendo que su peso en la

categorización se diluya; lo cual dificulta la tarea de clasificación del modelo de aprendizaje automático. La estrategia que se va a seguir para tratar de regularizar la selección de características es aplicar el umbral de frecuencias categoría a categoría, resultando el siguiente procedimiento:

1. Se agrupan los documentos por categorías de acuerdo al etiquetado de las muestras
2. Para cada categoría, se aplica el proceso de extracción y selección de características de acuerdo a una frecuencia mínima de aparición
 - Se establece un máximo de características seleccionables por categoría
3. Una vez recorridas todas las categorías se compone una lista resultante de la unión de las características extraídas de cada categoría individual
 - Si al unir las *features* de las distintas categorías se repitiera alguna palabra, esta solo se anotará una sola vez

Se comprueba como al seleccionar *features* individualmente por categorías se extraen palabras muy representativas. No obstante se debe fijar el umbral para el mínimo de ocurrencias entre 20 y 25, ya que para algunas profesiones no se encuentran demasiadas palabras. En esos casos, utilizar valores más altos para el umbral deriva en que el conjunto de palabras sea muy limitado. Se muestran a continuación las características seleccionadas para las categorías secretario/a (*clerk*), fotógrafo/a (*photo*) y conductor/a (*driver*).

CLERK

Most common features: 60

```
['record', 'customer', 'information', 'order', 'data', 'type', 'clerk',
'report', 'compiles', 'file', 'machine', 'prepares', 'material', 'payment',
'account', 'form', 'charge', 'credit', 'telephone', 'number', 'bill',
'computer', 'card', 'department', 'item', 'insurance', 'company',
'merchandise', 'personnel', 'mail', 'service', 'rate', 'amount',
'computes', 'worker', 'stock', 'date', 'document', 'receives', 'equipment',
'check', 'examines', 'bank', 'review', 'schedule', 'application',
'verifies', 'post', 'calculator', 'issue', 'production', 'work', 'time',
'repair', 'sale', 'request', 'duty', 'answer', 'change', 'cost']
```

PHOTO

Most common features: 36

```
['film', 'plate', 'machine', 'equipment', 'camera', 'print', 'material',
'mask', 'color', 'paper', 'position', 'type', 'photograph', 'data',
'image', 'specification', 'design', 'photo', 'computer', 'tape', 'record',
'copy', 'processing', 'solution', 'handtools', 'negative', 'line', 'order',
'place', 'examines', 'pattern', 'production', 'printing', 'control',
'customer', 'exposure']
```

DRIVER

Most common features: 8

```
['truck', 'drive', 'driver', 'vehicle', 'customer', 'record', 'machine',
'delivery']
```

El número de características obtenidas depende directamente del número de muestras que se dispone para cada categoría. En el ejemplo, se dispone de 273 muestras de la categoría *clerk*, 75 de *photo* y 53 para *driver*. La tabla que se presenta a continuación recoge la comparativa entre las pruebas realizadas con características obtenidas sobre el *dataset* (*ds_D*) completo frente a aquellas obtenidas individualmente sobre cada categoría; usando un umbral de 20 para la frecuencia mínima y de 60 para el máximo de *features* por categoría.

Selección dataset	Features	Frecuencia mínima	Máximo categoría	% de uso	Rendimiento validación	Rendimiento test	Modelo
Completo	N: 1027	20	Sin límite	4.2	0.5522	0.6066	NB
Individual	N: 316	20	60	10.0	0.6181	0.6352	NB

Tabla 10. Extracción de características sobre dataset completo vs. categorías individuales

Por encima del leve incremento en el rendimiento (apenas un 3%) resalta haber conseguido reducir el número de características desde 1027 a 316. Por tanto, se ha conseguido realizar un filtrado que reduce la dimensión de atributos de entrada y aumenta la representatividad de dichos atributos. Colateralmente, los tiempos de entrenamiento y ejecución de los modelos de aprendizaje mejoran considerablemente, es especial, cuando se usan redes neuronales (ya que el modelo bayesiano es normalmente bastante rápido).

Se menciona en último lugar que en el caso de emplear verbos como constituyentes sintácticos para la caracterización de los documentos, al presentarse estos en un número mucho menor que los nombres, el algoritmo de reducción de dimensionalidad produce una merma considerable en el número de *features* obtenidas por cada categoría. Tras aplicarlo, la mayoría de categorías aportan apenas 10/15 características, haciendo inviable el uso de esta metodología sobre este tipo de atributos.

5.3.4 Mejora de la representatividad de las características

Vamos ahora a continuar el análisis de los resultados utilizando los modelos de redes neuronales (NN). El motivo radica en que podemos trabajar sobre la matriz de confusión (ver ver [5.2.4 Clasificación mediante redes neuronales](#)) para obtener el acierto que el modelo obtiene sobre cada una de las categorías de forma individual. Para nuestras 19 categorías:

```
{'NONE': 0, 'CLERK': 1, 'COLOR_PAINTER': 2, 'DIRECTOR_MANAGER': 3,
'DRIVER': 4, 'EDUCATION_TEACHING_TRAINING_INSTRUCTOR': 5, 'ENGINEER': 6,
'GUARD_OFFICER_POLICE_SECURITY': 7, 'HEALTH_NURSE': 8, 'MAINTENANCE': 9,
'MECHANIC': 10, 'OPERATOR': 11, 'PHOTO': 12, 'PRINTER': 13, 'SALES': 14,
'SHOPPING': 15, 'STOCK': 16, 'TECHNICIAN': 17, 'WORKER': 18}
```

Obtenemos las siguientes métricas ([tabla 11](#), y acudir a [5.2.1 Métricas de rendimiento](#)):

	precision	recall	f1-score	support
1	0.74	0.74	0.74	61
2	0.64	0.64	0.64	22
3	0.50	0.62	0.55	34
4	0.56	0.60	0.58	15
5	0.58	0.55	0.56	20
6	0.64	0.55	0.59	51
7	0.72	0.50	0.59	26
8	0.25	0.25	0.25	8
9	0.20	0.08	0.11	13
10	0.44	0.55	0.49	22
11	0.65	0.67	0.66	91
12	0.41	0.41	0.41	17
13	0.72	0.50	0.59	36
14	0.73	0.72	0.73	46
15	0.40	0.40	0.40	10
16	0.12	0.12	0.12	8
17	0.22	0.19	0.20	32
18	0.46	0.66	0.54	56

Tabla 11. Métricas individuales por categoría

Atendiendo a la métrica *f1*, que pondera *precision* y *recall*, podemos notar como varias categorías no alcanzan el 50% de precisión. Por debajo de este porcentaje, el modelo realizará predicciones totalmente aleatorias o sesgadas.

Identificador	Nombre	Métricas <i>f1</i>	Muestras
8	HEALTH_NURSE	0.25	48
9	MAINTENANCE	0.11	55
10	MECHANIC	0.49	142
12	PHOTO	0.41	75
15	SHOPPING	0.40	44
16	STOCK	0.12	46
17	TECHNICIAN	0.20	154

Tabla 12. Categorías con bajo porcentaje de acierto

Las categorías 8, 9, 15 y 16 tienen en común no disponer de un número suficiente de muestras. Esto nos va a servir para estimar el umbral para el mínimo de muestras que se necesitan para poder llegar a clasificar una categoría. La clase 12 tiene el mismo número de

ejemplos que la clase 7 (*guard_officer_police_security*) y sin embargo el rendimiento es menor. Esto puede deberse a que las características seleccionadas para esta clase sean menos representativas debido al contenido del texto de las descripciones de dicha categoría. Lo contrario sucede con la categoría 4 (*driver*) de la que tan solo se dispone de 53 muestras y solo se han seleccionado 8 *features*: las características extraídas para esta categoría son altamente distintivas. Las categorías 10 y 17 parecen tener suficiente número de muestras, por lo que la causa del bajo rendimiento podría ser distinta (se analizará más adelante). El número de muestras para cada categoría puede consultarse en la sección [4.4 Descripciones de puestos laborales](#). Tomamos 80 como límite para el mínimo de muestras por categorías y mantenemos las categorías *driver* y *guard_officer_police_security* por el hecho de que se obtuvo buen rendimiento a pesar el bajo número de muestras. Ahora, con 14 categorías:

```
{'NONE': 0, 'CLERK': 1, 'COLOR_PAINTER': 2, 'DIRECTOR_MANAGER': 3,
'DRIVER': 4, 'EDUCATION_TEACHING_TRAINING_INSTRUCTOR': 5, 'ENGINEER': 6,
'GUARD_OFFICER_POLICE_SECURITY': 7, 'MECHANIC': 8, 'OPERATOR': 9,
'PRINTER': 10, 'SALES': 11, 'TECHNICIAN': 12, 'WORKER': 13}
```

Obtenemos las siguientes métricas ([tabla 13](#)):

	precision	recall	f1-score	support
1	0.77	0.84	0.81	57
2	0.83	0.48	0.61	21
3	0.57	0.64	0.61	36
4	0.60	0.33	0.43	9
5	0.86	0.67	0.75	18
6	0.76	0.62	0.69	56
7	0.60	0.50	0.55	18
8	0.56	0.71	0.63	35
9	0.74	0.78	0.76	80
10	0.60	0.58	0.59	31
11	0.69	0.78	0.73	51
12	0.34	0.29	0.32	34
13	0.61	0.67	0.64	55

Tabla 13. Métricas individuales por categoría tras cribado por mínimo de muestras

Analizamos las categorías que han obtenido menores rendimientos.

Identificador	Nombre	Métricas <i>f1</i>	Muestras
4	<i>DRIVER</i>	0.43	53
7	<i>GUARD_OFFICER_ POLICE_SECURITY</i>	0.55	75
12	<i>TECHNICIAN</i>	0.32	154

Tabla 14. Categorías con bajo rendimiento tras cribado por mínimo de muestras

Las categorías 4 y 7 eran las que habíamos mantenido a pesar de disponer de un número reducido de muestras. Recordemos que al realizar la división de los datos entre los conjuntos de entrenamiento y verificación (test), se realiza un barajado aleatorio de los ejemplos para tratar de escoger una muestra más representativa de la población en estudio. Esto implica que los resultados sobre categorías con pocos ejemplos son altamente dependientes del barajado realizado en el experimento, ya que podrían caer muy pocas muestras en el alguno de los conjuntos (entrenamiento o verificación). En la primera prueba realizada en este apartado las categorías *driver* y *guard* se verificaron (columna *support*) con 15 y 26 ejemplos respectivamente. En esta última prueba, se han verificado con 9 y 18 ejemplos. Esto confirma, que a pesar de que se hayan obtenido características bien descriptivas para ambas categorías, se sigue dependiendo de un mínimo número de ejemplos para asegurar un buen rendimiento en cualquier circunstancia.

La precisión global alcanzada por el modelo NB es ahora de un 72%.

Selección dataset	Features	Mínimo muestras	Máximo categoría	% de uso	Rendimiento validación	Rendimiento test	Modelo
Individual	N: 301	80	60	10.3	0.6604	0.7207	NB

Tabla 15. Rendimiento tras cribado por mínimo de muestras

Nos centramos ahora en la categoría *technician*, que a pesar de tener 154 muestras sigue exponiendo un rendimiento muy bajo. En primer lugar, hemos de recordar que aunque la selección de características se realiza de forma individual por categorías, la lista global de *features* seleccionadas se compone uniendo cada selección individual. Al realizar esta unión las palabras repetidas solo computan una vez, pero esto no implica que pueda haber subconjuntos de palabras que sean descriptivas de varias categorías. Si esta circunstancia se da, cuando se realiza la caracterización de las muestras sucederá que algunos ejemplos de categorías distintas quedarán solapadas y consecuentemente serán más difíciles de diferenciar para el clasificador. Vamos a comparar la categoría *technician* con algunas de las categorías con las que podría estar potencialmente más solapada: *engineer*, *mechanic* y *worker*.

TECHNICIAN

```
[ 'equipment', 'test', 'system', 'specification', 'instrument', 'machine', 'material', 'data',
  'repair', 'part', 'handtools', 'record', 'patient', 'film', 'procedure', 'operation', 'tool',
  'engineering', 'computer', 'control', 'component', 'personnel', 'production', 'knowledge',
  'design', 'use', 'program', 'device', 'measure', 'prepares', 'installs', 'information',
  'report', 'performs', 'power', 'processing', 'maintenance', 'adjusts', 'work', 'result',
  'installation', 'customer', 'worker', 'blueprint', 'pattern', 'unit', 'activity', 'method',
  'circuit', 'problem', 'plant', 'sample', 'heart', 'cable', 'instruction', 'technician',
  'order', 'type', 'inspects', 'performance' ]
```

MECHANIC

```
[ 'equipment', 'part', 'test', 'handtools', 'machine', 'repair', 'instrument', 'system',
  'tool', 'engine', 'installs', 'power', 'unit', 'replaces', 'specification', 'adjusts',
  'component', 'mechanic', 'assembly', 'control', 'aircraft', 'pump', 'vehicle', 'assembles',
  'motor', 'device', 'operation', 'machinery', 'gauge', 'precision', 'valve', 'blueprint',
  'duty', 'inspects', 'disassembles', 'water', 'engineering', 'examines', 'malfunction',
  'master', 'title', 'worker', 'meter', 'clean', 'bolt', 'mechanism', 'pressure', 'remove',
  'truck', 'wire', 'design', 'service', 'defect', 'press', 'maintains', 'material',
  'installation', 'record', 'position', 'gas' ]
```

ENGINEER

```
[ 'equipment', 'design', 'system', 'engineering', 'product', 'engineer', 'plan', 'test',
  'specification', 'operation', 'material', 'production', 'activity', 'control', 'construction',
  'personnel', 'research', 'directs', 'data', 'cost', 'facility', 'method', 'problem',
  'maintenance', 'requirement', 'project', 'installation', 'conduct', 'duty', 'title',
  'knowledge', 'worker', 'process', 'water', 'part', 'plant', 'procedure', 'development',
  'study', 'machinery', 'master', 'work', 'machine', 'repair', 'customer', 'power', 'report',
  'program', 'type', 'gas', 'drawing', 'field', 'confers', 'safety', 'oil', 'prepares',
  'analysis', 'standard', 'area', 'performance' ]
```

WORKER

```
[ 'machine', 'part', 'equipment', 'worker', 'plant', 'metal', 'place', 'task', 'hand',
  'performs', 'material', 'tool', 'combination', 'handtools', 'area', 'remove', 'position',
  'work', 'conveyor', 'truck', 'section', 'clean', 'power', 'crop', 'water', 'duty', 'record',
  'container', 'surface', 'tank', 'press', 'cut', 'specification', 'wire', 'article', 'line',
  'cloth', 'brush', 'order', 'furnace', 'farm', 'repair', 'product', 'end', 'shovel', 'sheet',
  'number', 'helper', 'valve', 'hammer', 'production', 'knife', 'plate', 'solution', 'size',
  'device', 'operator', 'type', 'stack', 'fruit' ]
```

Donde el significado de los colores es el siguiente:

- Características de *technician* comunes con el resto de categorías
- Características de *mechanic* comunes solo con *engineer* y *worker*
- Características de *engineer* comunes solo con *worker*

Destaca claramente el alto grado de solapamiento de la categoría *technician*, que guarda especial parecido con la categoría *engineer*. Es por ello, que el clasificador confunde frecuentemente los ejemplos de la categoría *technician* con los de la categoría *engineer*, *mechanic* y eventualmente con los de la categoría *worker*. Dado que las otras categorías tienen menor grado de acoplamiento y además este es más cruzado, el clasificador consigue obtener mejores resultados, lo cual no quita que el rendimiento en estas clases no llegue a ser elevado.

Idealmente, la selección de características debería obtener una matriz de frecuencia que activara siempre el mismo conjunto de columnas para cada categoría y que a su vez dichas columnas fueran diferentes entre las distintas categorías. Esto ya sucede para categorías asociadas a perfiles laborales bien diferenciables.

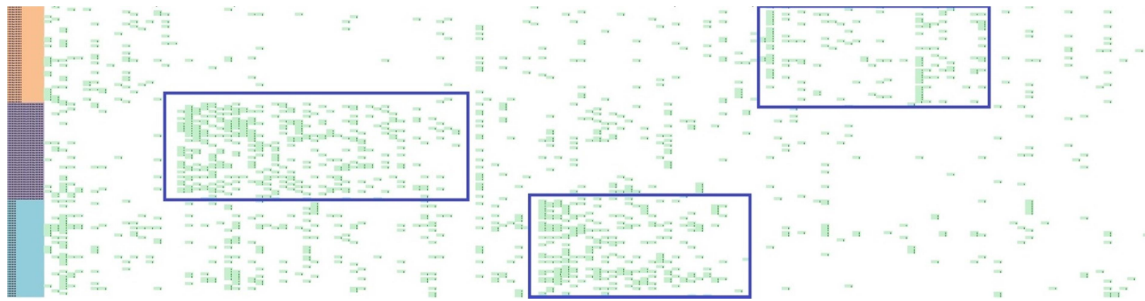


Ilustración 18. Matriz de frecuencia de términos para categorías bien diferenciables

Correspondiéndose las distintas franjas con:

	<i>COLOR_PAINTER</i>
	<i>EDUCATION_TEACHING_TRAINING_INSTRUCTOR</i>
	<i>ENGINEER</i>

En este ejemplo, se puede notar una clara diferenciación del conjunto de características que define a cada categoría. De igual forma, se puede apreciar como las características obtenidas para *education* y *engineer* son más densas y por lo tanto más representativas que para *color_painter*.

Para intentar desacoplar en la mayor medida de lo posible aquellas categorías que sean más similares, se van a eliminar las *features* más comunes entre las distintas categorías. Como esto reducirá el número de características representativas de cada clase, se tratará de compensar ampliando el límite de características seleccionables por categoría a 70 (anteriormente 60). Para la comparativa entre *technician* y *engineer*, solo una de las diez nuevas palabras coincide.

El algoritmo utilizado para cribar las palabras más comunes entre categorías es el siguiente:

1. Para cada *feature*, se evalúa el número de veces que aparece dicha palabra en cada categoría.
2. Si la frecuencia de aparición supera un umbral, se considera una característica relevante, en cuyo caso se incrementa un contador.
3. Si para la *feature* en análisis, el contador es mayor a la unidad, es decir, se ha encontrado que esta característica es relevante para más de una categoría, se procede a eliminarla de la matriz de caracterización.

En resumen, si una palabra es muy común en las descripciones de una categoría se considera relevante. Si dicha palabra es relevante para más de una categoría se considerará genérica y se procederá a eliminarla del conjunto de características seleccionadas.

Features	Frecuencia de cribado	Máximo categoría	Rendimiento validación	Rendimiento test	Modelo
N: 298	Ilimitado	60	0.7019	0.7103	NB
N: 291	60	60	0.6974	0.7051	NB
N: 325	Ilimitado	70	0.6923	0.6923	NB
N: 318	60	70	0.6891	0.6949	NB
N: 307	50	70	0.6635	0.6769	NB
N: 289	40	70	0.6923	0.6692	NB
N: 256	30	70	0.641	0.6359	NB

Tabla 16. Reducción de característica para desacoplar categorías

Se aprecia cómo está penalizando más la pérdida de información por eliminar columnas que la “ganancia” buscada al desacoplar las *features* seleccionadas.

Como enfoque complementario se va a tratar de modificar el peso que se asigna a las características en función de la frecuencia de aparición de cada término en un documento y de la frecuencia cruzada de cada término a lo largo de la colección de documentos. Para ello, vamos a construir la matriz *tf-idf* (ver [2.2.4 Clasificación de texto](#)) y a utilizarla como entrada del clasificador.

Features	Matriz de entrada	Número de categorías	Rendimiento validación	Rendimiento test	Modelo
N: 298	<i>tf</i>	13	0.7019	0.7103	NB
N: 298	<i>tf-idf</i>	13	0.7083	0.6974	NB

Tabla 17. Comparativa matriz *tf* vs. *tf-idf*

Tal y como apuntan algunos autores [34], la modificación de los pesos mediante el cálculo de la matriz *tf-idf* no siempre consigue mejorar los resultados sobre la aplicación directa de la frecuencia de términos (*tf*).

Finalmente se procede eliminar las categorías que presentan mayor grado de acoplamiento con el objetivo de preparar un *dataset* que muestre rendimientos más interesantes de cara a la última fase del trabajo, en la que se está interesado en la interfaz de usuario.

Features	Número de categorías	Rendimiento validación	Rendimiento test	Modelo
N: 255	10	0.7797	0.7993	NB
N: 242	9	0.7953	0.8513	NB

Tabla 18. Rendimiento tras acotar el número de clases

Las rutinas de código utilizadas para obtener los resultados de los experimentos planteados están contenidas en los archivos:

- “020_Features_Statistics_N” y “020_Features_Statistics_INDIVIDUAL_N”
- “020_Features_Duty_N” y “020_Features_Duty_N_INDIVIDUAL”
- “040_Classifier_by_NB”, “040_Classifier_by_NN” y “040_Classifier_by_NB_Tf_Idf”

5.3.5 Clasificación mediante *word vectors*

En este apartado se investiga la viabilidad de los métodos basados en representaciones densas para la clasificación de texto. Se ha utilizado el corpus compuesto por la colección de documentos para entrenar *word embeddings* (ver [2.1.1 Recursos lingüísticos y representación del significado](#)) y utilizarlos como entrada para un clasificador basado en redes neuronales. La implementación puede revisarse en “050_Classifier_with_w2v”.

Se debe anotar que esta técnica trabaja directamente sobre el total de las palabras, siendo el único filtro que se aplica el número de palabras distintas a considerar cuando se escoge el tamaño del vocabulario. Una vez filtrado el vocabulario, se construye un diccionario que asigna un identificador numérico a cada palabra. Dado que el algoritmo exige fijar el número de palabras que se evalúa por cada muestra, al convertir las palabras a su identificador se realizan las siguientes operaciones:

- Se asigna el identificador numérico de acuerdo al diccionario de términos.
- Si la palabra no aparece en el diccionario se asigna un dos (“*unknown*”).
- Si la muestra contiene más palabras que el número máximo evaluable se descartan las palabras que excedan dicho límite.
- Cuando la muestra contiene menos palabras, se rellena con ceros (“*padding*”).

Se presenta a continuación el procesado sobre una de las muestras:

Text-coded sample containing 160 words

```
[ 8 7 294 248 295 7 296 15 297 298 19 299 24 300 273 58 248 256
 7 17 248 295 301 260 19 302 261 303 15 304 7 305 7 262 19 302
306 248 270 7 307 308 84 166 15 248 309 294 308 248 309 7 310 7
248 295 15 297 293 311 248 312 62 297 293 7 294 313 15 248 293 38
314 174 270 19 315 248 296 98 316 7 317 248 312 98 318 258 64 319
320 19 273 38 321 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

Decoded text

```
'compiles and dispatches advertising schedule and material to composing
room of daily or weekly publication reviews advertising order and prepares
advertising schedule listing size of ad date s to appear and page and
position of ad searches advertising files and selects mat that corresponds
to advertising layout dispatches mat advertising layout and copy and
advertising schedule to composing department obtains advertising proofs
from composing department and dispatches them to advertising department for
proofreading maintains files of all advertising material may proofread and
correct advertising proofs may read advertisement in first edition of
publication for errors PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD
PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD
PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD
PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD PAD'
```

En una primera instancia, cuando se realizaron pruebas sobre el *dataset* original (boletines de trabajo) la escasez de datos propiciaba que los *word vectors* no llegaran a entrenar y el modelo era incapaz de acertar predicciones. Al trabajar sobre el *dataset* extendido se llegan a conseguir resultados similares a los obtenidos con la matriz de frecuencia, lo cual predispone esta técnica como candidata para futuras investigaciones. Cabe resaltar como el modelo obtiene mejores resultados cuanto mayor sea el tamaño del vocabulario, ya que al trabajar con vocabularios reducidos pierde detalles (información) sobre el contexto de las palabras.

Features (NB) / Vocabulario (NN)	Matriz de entrada	Número de categorías	Rendimiento validación	Rendimiento test	Modelo
N: 298	<i>Tf</i>	13	0.7019	0.7103	NB
298 (filtrado)	<i>Word vectors</i>	13	0.5833	0.5410	NN
11316 (completo)	<i>Word vectors</i>	13	0.7212	0.7128	NN

Tabla 19. Comparativa matriz *tf* vs. *word vectors*

5.4 Filtrado y recomendación

Las tareas de filtrado y recomendación van a basarse en información que se extrae del usuario mediante técnicas de reconocimiento de entidades (ver [2.2.3 Reconocimiento de entidades \(NER\)](#)). Esta información se usará para reducir el conjunto de boletines de trabajo preseleccionado (con el perfil laboral asignado) y mostrar tan solo información relevante al usuario. Por simplicidad y dado que lo que se pretende es mostrar y resolver el proceso y no hacer un desarrollo completo de la funcionalidad de filtrado, nos basaremos en el uso de un único atributo: el rango de salario buscado por el usuario. El salario mínimo y máximo para una oferta dada puede obtenerse directamente mediante la aplicación de expresiones regulares sobre el campo “*Salary*” de los boletines de trabajo.

```
#Regular expression for extracting numbers
salary_bounds=re.findall(r"[+-]?\d*\.\d+|[+-]?\d*\,\d+|\d+", salary_info)
```

Ilustración 19. Expresión regular para obtención del rango salarial

Por último se va a construir la base de datos sobre la que el módulo de filtrado y recomendación trabajará. Se aclara que para la obtención de la columna “*profile*” (perfil laboral de la oferta), en vez de utilizar el etiquetado manual propuesto inicialmente se ha procedido a etiquetar los boletines de trabajo mediante las predicciones realizadas por el primer modelo presentado en la [tabla 18](#). Aunque el primer modelo tiene menor rendimiento que el segundo, al incluir una categoría adicional (*police_guard*) que aparece bastante en los boletines, es capaz de cubrir mejor la variedad de las muestras.

	Job_name	Profile	Salary_range	Bulletin
10	AIRPORT CHIEF INFORMATION SECURITY OFFICER	GUARD_OFFICER_POLICE_SECURITY	(125.551, 183.576)	AIRPORT CHIEF INFORMATION SECURITY OFFICER 140...
11	AIRPORT ENGINEER	ENGINEER	(105.005, 163.532)	AIRPORT ENGINEER 7256 070618.txt
12	AIRPORT GUIDE	GUARD_OFFICER_POLICE_SECURITY	(31.925, 46.687)	AIRPORT GUIDE 0845 042018.txt
13	AIRPORT INFORMATION SPECIALIST	DIRECTOR_MANAGER	(42.991, 78.529)	AIRPORT INFORMATION SPECIALIST 1783 121115.txt
14	AIRPORT LABOR RELATIONS ADVOCATE	DIRECTOR_MANAGER	(93.918, 137.327)	AIRPORT LABOR RELATIONS ADVOCATE 9210 020119.txt
15	AIRPORT MANAGER	DIRECTOR_MANAGER	(99.326, 188.567)	AIRPORT MANAGER 7260 120216.txt
16	AIRPORT POLICE CAPTAIN	GUARD_OFFICER_POLICE_SECURITY	(142.568, 187.001)	AIRPORT POLICE CAPTAIN 3228 021618.txt
17	AIRPORT POLICE LIEUTENANT	GUARD_OFFICER_POLICE_SECURITY	(102.186, 134.028)	AIRPORT POLICE LIEUTENANT 3227 091616.txt
18	AIRPORT POLICE OFFICER	GUARD_OFFICER_POLICE_SECURITY	(51.448, 83.019)	AIRPORT POLICE OFFICER 3225 110906 Rev 060115.txt
19	AIRPORT POLICE SPECIALIST	GUARD_OFFICER_POLICE_SECURITY	(0.0, 0.0)	AIRPORT POLICE SPECIALIST 3236 063017 (2).txt

Ilustración 20. Base de datos para filtrado y recomendación

La rutina de código que construye la base de datos para el filtrado y recomendación se encuentra en el archivo “012_Dataset_tagged_filter”.

5.5 Agente conversacional

La última fase de presente proyecto consiste en la implementación de un agente conversacional que provea al usuario con una interfaz intuitiva y fluida. Para la implementación de este módulo se han utilizado los siguientes elementos:

- *Dialogflow*: herramienta desarrollada por Google para la creación, configuración y entrenamiento de agentes conversacionales. Tiene los siguientes cometidos:
 - Controlar y guiar el flujo de la conversación.
 - Capturar la información que necesitan los módulos de inferencia y de filtrado.
- Módulo de inferencia: utiliza las rutinas desarrolladas en los apartados anteriores.
 - Se encarga de realizar una predicción sobre el perfil laboral del usuario.
- Módulo de filtrado: utiliza las rutinas desarrolladas en los apartados anteriores.
 - Se encarga de filtrar boletines de trabajo en función de las preferencias del usuario y el perfil laboral asignado.

5.5.1 Estructura de la aplicación en Python

En esta ocasión dejaremos de lado los *notebook* de *Jupyter* (orientado a investigación) para hacer uso de *PyCharm*⁶, uno de los entornos de desarrollo (*IDE*) más potentes y populares para el desarrollo de aplicaciones en *Python*. Los módulos de inferencia y filtrado van a formar parte de una aplicación (*app*) que se ejecutará en un servidor (*local*) a modo de servicio. Dicho servicio se comunicará con *Dialogflow* a través de peticiones *HTTP* gestionadas por el *framework Flask*⁷, el cual permite crear y desplegar aplicaciones web de forma ágil y sencilla. La estructura de la aplicación (*job_consultant*) es la siguiente:

⁶ <https://www.jetbrains.com/pycharm/>

⁷ <https://palletsprojects.com/p/flask/>

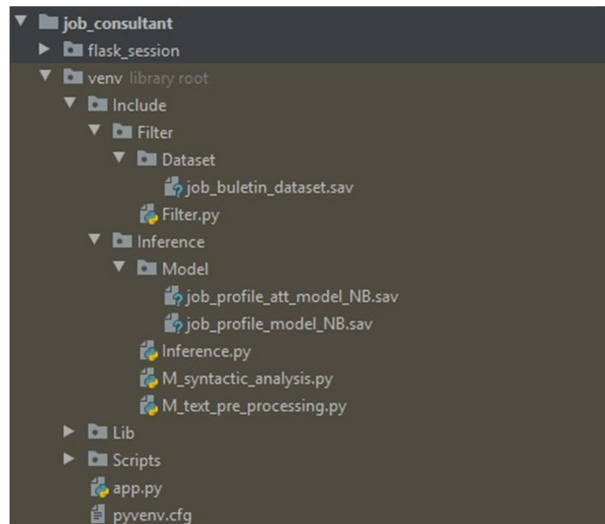


Ilustración 21. Estructura del proyecto *job_consultant*

Toda aplicación desarrollada en *PyCharm* se asigna a un entorno virtual (carpeta *venv* en la [figura 21](#)). Cada entorno virtual posibilita distinta configuración y valores para las variables de entorno y además permite tener instalado un conjunto de paquetes y módulos específico. Además de una ejecución más ligera, este aspecto abre la posibilidad de utilizar versiones diferentes de un mismo módulo en distintas aplicaciones (distintos *venv*); lo cual resulta muy interesante para solventar aquellas situaciones en las que algunas dependencias de nuestras aplicaciones no sean compatibles con la misma versión de algún otro módulo. En nuestro caso particular, dentro del entorno virtual encontramos los siguientes módulos:

- *App.py*: rutina principal donde se gestionan las peticiones HTTP y se cargan y hacen las llamadas al resto de módulos de la aplicación. La función *webhook()* representa el punto de entrada que enruta y gestiona las peticiones recibidas desde de *Dialogflow*.

```
@app.route('/webhook', methods=['POST'])
def webhook():

    debug='on'
    #Request reception
    req = request.get_json(silent=True, force=True) #Decoding the request received on json format
    if debug=='on':
        print("REQUEST:")
        print(json.dumps(req, indent=4))
    #Processing the request for generating a result
    result = request_processing(req)
    result = json.dumps(result, indent=4) #Convert result into json format for responding
    if debug=='on':
        print(result)
    response = make_response(result)
    response.headers['Content-Type'] = 'application/json'

    return response
```

Ilustración 22. Gestión de peticiones HTTP recibidas desde el *webhook*

Los datos intercambiados entre la aplicación y *Dialogflow* se codifican en formato *json*. Es importante conocer la estructura del intercambio para poder modificar el comportamiento de la conversación cuando sea necesario⁸. Por otro lado, destaca la definición del puerto configurado para escuchar las peticiones entrantes, que será el mismo por el que se envía la respuesta hacia *Dialogflow*.

```
if __name__ == '__main__':
    port = int(os.getenv('PORT', 5000))
    print("Starting app on port %d" % port)
    app.run(debug=True, port=port, host='0.0.0.0')
```

Ilustración 23. Configuración del puerto para el intercambio HTTP en la aplicación

No obstante no es suficiente con configurar el puerto, este debe ser habilitado. Para ello se ha utilizado *ngrok*⁹, una aplicación que permite crear túneles seguros hacia cualquier servidor. Ejecutando el comando `>> ngrok http 5000` estaremos creando un túnel hacia nuestro servidor local a través del puerto 5000.

```
ngrok by @inconshreveable
Session Status      online
Session Expires    7 hours, 58 minutes
Version             2.3.34
Region              United States (us)
Web Interface       http://127.0.0.1:4040
Forwarding           http://29cc1968.ngrok.io -> http://localhost:5000
                    https://29cc1968.ngrok.io -> http://localhost:5000
Connections
  ttl  opn  rt1  rt5  p50  p90
   0    0    0.00 0.00 0.00 0.00
```

Ilustración 24. Túnel creado con *ngrok*

El campo *forwarding* que puede verse en la imagen anterior representa la *URL* sobre la que realizar las peticiones para que lleguen al servidor.

- **Include:** directorio donde se incluyen los módulos de usuario.
 - **Inference:**
 - *Inference.py*: predice el perfil laboral del usuario. Utiliza el modelo almacenado en *Model* y funciones de los otros dos módulos contenidos en este directorio.
 - *M_syntactic_analysis.py*: contiene las funciones de análisis sintáctico desarrolladas en los apartados anteriores.
 - *M_text_pre_processing.py*: contiene las funciones de pre-procesado desarrolladas en los apartados anteriores.

⁸ <https://cloud.google.com/dialogflow/docs/>

⁹ <https://ngrok.com/>

- Model:
 - *job_profile_att_model_NB.sav*: *features* del modelo NB seleccionado tras el entrenamiento.
 - *job_profile_model_NB.sav*: modelo NB generado durante la fase de entrenamiento.
- Filter:
 - *Filter.py*: implementa la funcionalidad de filtrado de ofertas laborales.
 - Dataset:
 - *Job_buletin_dataset.sav*: contiene el *dataset* que recopila las ofertas de trabajo, el perfil laboral, y el rango salarial.
- Libs: librerías y paquetes de sistema instalados en el *venv*.
- Scripts: *scripts* y *.dll* del entorno de ejecución.

5.5.2 Control del flujo de diálogo

Procedemos en este apartado a especificar la estructura que se ha dado al agente conversacional. Llegados a este punto puede ser interesante recordar qué funcionalidades ofrece *Dialogflow* y cómo se interrelacionan entre ellas (ver [2.5.4 Dialogflow](#)).

El agente conversacional controla el flujo de la conversación mediante reglas de producción (*intents*). Estas reglas reaccionan ante ciertos eventos y producen una respuesta determinada por parte del agente conversacional, que representa la intención del agente en cada turno de palabra. Aunque se pueden utilizar una serie de eventos externos, lo más habitual es que los *intents* reaccionen al detectar entidades en el mensaje (en texto o audio) transmitido por el usuario. A su vez, además de generar una salida de texto cada *intent* puede activar un contexto determinado; cuyo cometido principal es desambiguar las frases de la conversación y guiar el flujo de la misma. Del mismo modo, otros *intents* pueden estar condicionados por la existencia de algún contexto de la siguiente forma:

- Un *intent* sin contexto siempre está activo.
- Un *intent* con uno o varios contextos solo se activa cuando todos sus contextos están activos.

La salida de texto no siempre tiene porqué estar generada por *Dialogflow*, sino que se puede delegar este cometido a aplicaciones externas mediante el uso de *webhooks* (invocado por acciones) dentro de la funcionalidad que *Dialogflow* denomina *fulfillment*.

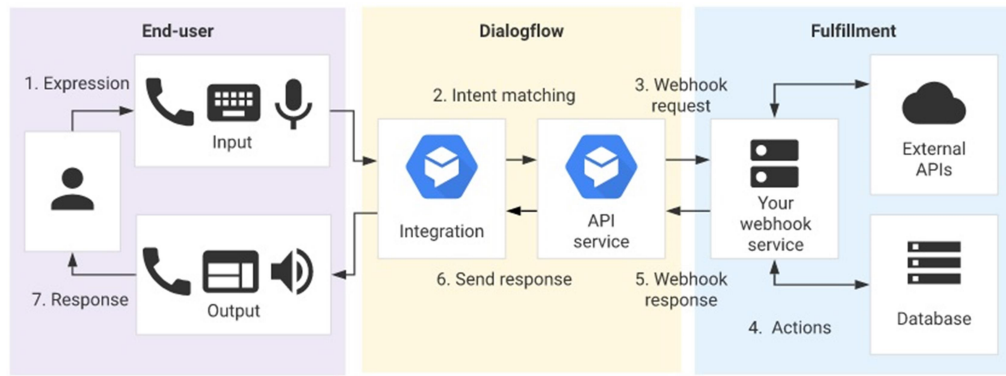


Ilustración 25. Integración de motores de inferencia externos en Dialogflow

Fuente: <https://cloud.google.com/dialogflow/docs/basics>

El mencionado *webhook* debe apuntar a la URL generada por *ngrok*.

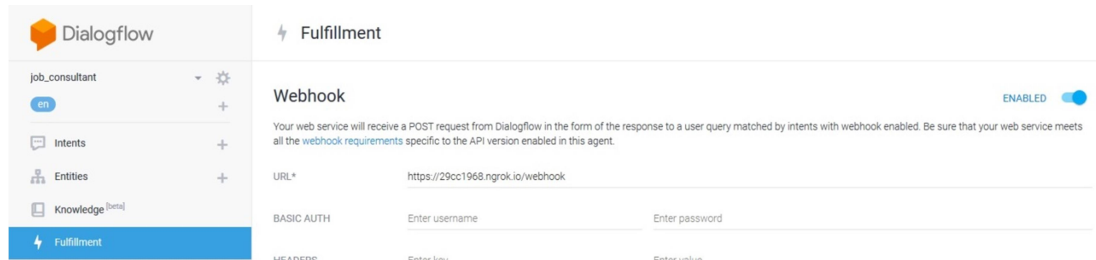


Ilustración 26. Configuración del *webhook* en Dialogflow

Describimos a continuación el cometido de cada uno de los *intents* previstos para el agente *job_consultant*:

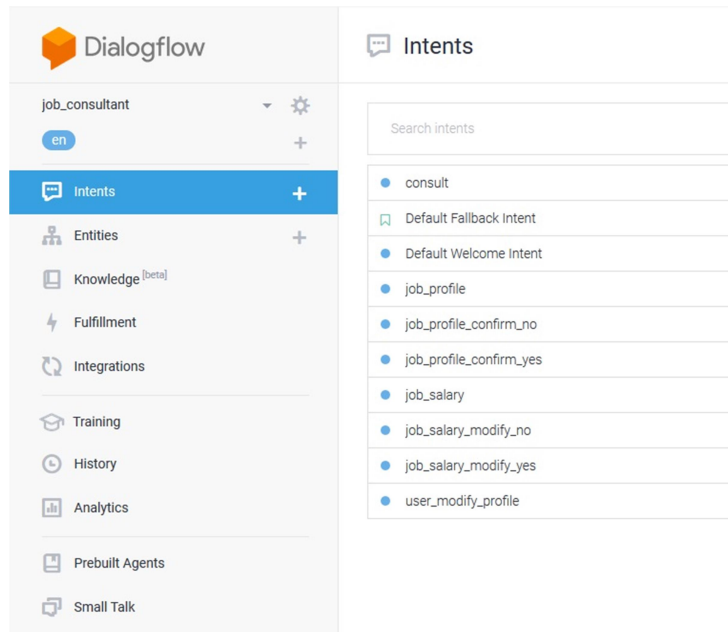


Ilustración 27. *Intents* del agente conversacional

- **Default Welcome Intent**: este *intent* se activa cuando el agente recibe una petición de iniciar un diálogo, es decir, la primera vez que el usuario escribe un texto en la interfaz. Nótese cómo es el usuario el que inicia el diálogo y cómo esto condiciona que por defecto lleve la iniciativa en la conversación. Si se desea que el agente tome la iniciativa, en algún turno de palabra tendremos que responder a la petición del usuario y además adicionar una pregunta. En nuestro caso, aprovechamos la respuesta del *intent* de bienvenida para tomar la iniciativa.

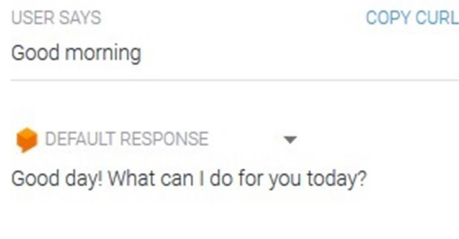


Ilustración 28. Respuesta del *intent* de bienvenida

- Evento de entrada: *wecolme* (inicio de diálogo)
- **Default Fallback Intent**: actúa cuando el texto a procesar no encaja con ninguna de las reglas activas. Pide al usuario que reformule la frase de entrada.
- **consult**: activo en todo momento, reconoce peticiones de consulta de ofertas laborales por parte del usuario.
 - Contexto de salida: *check_skills* (*verificar_habilidades*)
- **job_profile**: procesa la descripción de las competencias y habilidades facilitadas por el usuario y asigna el perfil laboral más probable.
 - Contexto de entrada: *check_skills*
 - Contexto de salida: *confirm_profile* (*confirmar_perfil*)
 - Acción: *job_profile*; utiliza el módulo de inferencia.
- **job_profile_confirm_no**: si el usuario no está de acuerdo con el perfil asignado, se le permite volver a redactar la descripción de sus competencias y habilidades.
 - Contexto de entrada: *confirm_profile*
 - Contexto de salida: *check_skills*
- **job_profile_confirm_yes**: si el usuario confirma el perfil asignado se procede a preguntar por sus preferencias salariales.
 - Contexto de entrada: *confirm_profile*
 - Contexto de salida: *profile_verified* (*perfil_verificado*)
- **job_salary**: se ofrece la lista de ofertas de trabajo que coinciden con el perfil asignado y concuerdan con las preferencias del usuario. Se pregunta al usuario si desea modificar el rango salarial para acceder a un conjunto distinto de boletines de trabajo.

- Contexto de entrada: *profile_verified*
- Contexto de salida: *salary_checked* (*salario_verificado*), *preferences_retrieval* (*obtención_preferencias*)
 - Acción: *job_salary*; utiliza el módulo de filtrado.
- *job_salary_modify_no*: si el usuario no desea modificar sus preferencias se asume que está satisfecho con las ofertas presentadas y se realiza el cierre del servicio.
 - Contexto de entrada: *salary_checked*
 - Se configura este *intent* como punto de finalización de la conversación
- *job_salary_modify_yes*: si el usuario confirma que desea modificar el rango salarial, se activa el contexto de entrada del *intent* en el que se pregunta al usuario sobre sus preferencias salariales.
 - Contexto de entrada: *salary_checked*
 - Contexto de salida: *profile_verified*
- *user_modify_profile*: desde el momento en el que se verifica el perfil laboral, se procede a la fase de obtención de preferencias del usuario, marcada por el contexto *preferences_retrieval*. Aunque la implementación aquí presentada solo incluye la evaluación de una preferencia (rango salarial), podrían anidarse un número indefinido de preferencias antes de realizar el filtrado y presentar las ofertas laborales al usuario. En todo momento el agente guiaría al usuario preguntando sobre una preferencia determinada. Para posibilitar un diálogo más natural en el que el usuario decidiera interrumpir al agente y requerir la modificación del perfil laboral o de alguna preferencia anteriormente evaluada se podría configurar un *intent* de este tipo para cada dato a modificar. En este caso concreto, se activará como salida el contexto de entrada del *intent* de asignación del perfil laboral.
 - Contexto de entrada: *preferences_retrieval*
 - Contexto de salida: *check_skills*



Ilustración 29. Toma de iniciativa del flujo de diálogo por parte del usuario

Dado que esto nos devuelve a un punto anterior del flujo de la conversación, el agente volverá a preguntar por información ya recabada. Se plantean dos opciones: forzar (por código en la respuesta del *webhook*) que se active el contexto inmediatamente posterior al que estaba activo cuando el usuario realizó la interrupción o volver a preguntar por todas las preferencias. Se ha optado por esta segunda opción porque hace al usuario reflexionar sobre la validez de los otros datos introducidos. Por supuesto, se permite que el usuario pida mantener las preferencias que introdujo con anterioridad (recordadas por la aplicación en el lado de *webhook*).

USER SAYS COPY CURL
keep the same, please

Ilustración 30. Frase para mantener las preferencias anteriores

Por defecto, los contextos permanecen activos durante 20 minutos. El siguiente esquema muestra el flujo de dialogo condicionado por los *intents* y los contextos, el cual represente la máquina de estados finitos que controla el flujo de diálogo del agente.

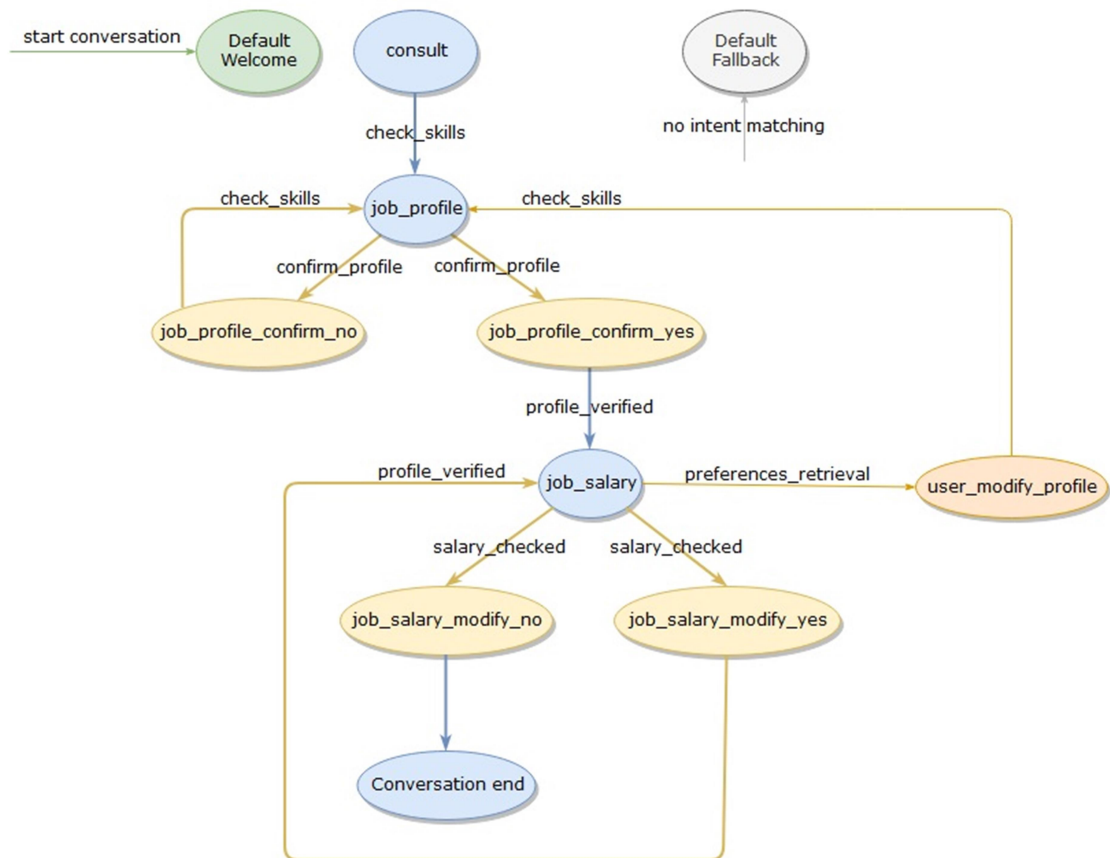


Ilustración 31. Máquina de estados del diálogo del agente conversacional

5.5.3 Entrenamiento del agente conversacional

El panel de mando (*dashboard*) de *Dialogflow* permite configurar y entrenar el agente conversacional de forma ágil e intuitiva. Cada *intent* trabajará normalmente para detectar una o más entidades. Además de las entidades de sistema¹⁰, *Dialogflow* permite crear entidades de usuario.

U_job_consult DONE

Define synonyms Regexp entity Allow automated expansion Fuzzy matching

job	career, duty, job, profession, work
job bulletin	job bulletin, job bulletins
job offer	job offer, job vacancies, job vacancy, open position, open positions, work occupation, work opportunities, work opportunity

Ilustración 32. Entidad de usuario

Para entrenar el agente, seleccionamos un *intent* y escribimos las frases con la estructura que se espera que utilice el usuario. El sistema detectará automáticamente en las frases las entidades presentes y las añadirá como parámetros que pueden o bien ser utilizados en la respuesta directa o ser utilizados como entrada en el procesado realizado por el *webhook*. Si el sistema no detecta las entidades, pueden ser asignadas manualmente. Como para el reconocimiento de entidades *Dialogflow* trabaja con porcentajes de confianza, la frase utilizada por el usuario no tiene por qué coincidir exactamente con la de entrenamiento, si bien deberá ser razonablemente parecida. Por ello, se recomienda la adición de al menos 15 variantes para entrenar cada *intent*.

¹⁰ <https://dialogflow.com/docs/reference/system-entities>

Training phrases ⓘ Search training phrases 🔍

- ” Add user expression
- ” Could you look for some job vacancies for me?
- ” Can I get information about open positions?
- ” Could you show me the job offers you have?
- ” I want to look up any work occupation at Los Angeles
- ” I'm interested in the work opportunities on the city
- ” I would like to know about the job offers you have
- ” I need assistance for getting a duty
- ” Is there any job available?
- ” my first profession
- ” A change for developing my career

Ilustración 33. Frases de entrenamiento para el intent consult

Action and parameters ⓘ

REQUIRED ⓘ	PARAMETER NAME ⓘ	ENTITY ⓘ	VALUE	IS LIST ⓘ
<input type="checkbox"/>	U_job_consult	@U_job_consult	SU_job_consult	<input type="checkbox"/>
<input type="checkbox"/>	ordinal	@sys.ordinal	Sordinal	<input type="checkbox"/>
<input type="checkbox"/>	geo-city	@sys.geo-city	Sgeo-city	<input type="checkbox"/>
<input type="checkbox"/>	Enter name	Enter entity	Enter value	<input type="checkbox"/>

Ilustración 34. Parámetros detectados para el intent consult

Para cada *intent*, tenemos la posibilidad de asociar una serie de respuestas estáticas. Es recomendable añadir varias frases como posible respuesta, para que *Dialogflow* pueda seleccionar (aleatoriamente) cada vez una diferente y se aporte mayor naturalidad a la conversación. Un enfoque interesante para hacer que la combinatoria sea mayor es escribir la respuesta separada en dos frases en vez de en una sola ([figura 35](#)), lo cual genera el resultado mostrado en la [figura 36](#).

Responses ⓘ

Text Response	Text Response
1 You are in the right place at the right time.	1 First I need to know about your skills and competencies, that is, the set of tasks you can accomplish and the related-knowledge you have.
2 I can surely find something interesting for you.	2 In order to focus the search, I will need you to enumerate and describe all the duties you can develop and the associated skills you have got.
3 Great, let's work on it.	3 For narrowing down the look up, you have to write down any labor you can do together with each ability that will allow you to perform them.
4 That is for what I've been trained.	4 Let me know about what you can tasks you can carry out and what abilities you have for helping you go through.
5 Sure I can do.	5 Enter a text response variant

Ilustración 35. Respuesta estática para el intent consult

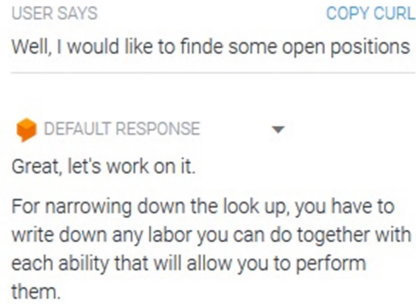


Ilustración 36. Respuesta formada por la combinación de dos frases estáticas

5.5.4 Asignación del perfil laboral

La asignación del perfil laboral es realizada por el módulo de inferencia de la aplicación *job_consultant*. El *intent consult* pregunta por las competencias y habilidades del usuario y activa el contexto de entrada del *intent job_profile*. Este último *intent* lanza la acción *job_profile*, que utiliza el *webhook* para enviar el texto introducido por el usuario a la aplicación *job_consultant*. Dado que el texto puede tener una estructura libre se utiliza la entidad de sistema *@sys.any*, que captura cualquier texto en la posición indicada de la frase de entrenamiento.

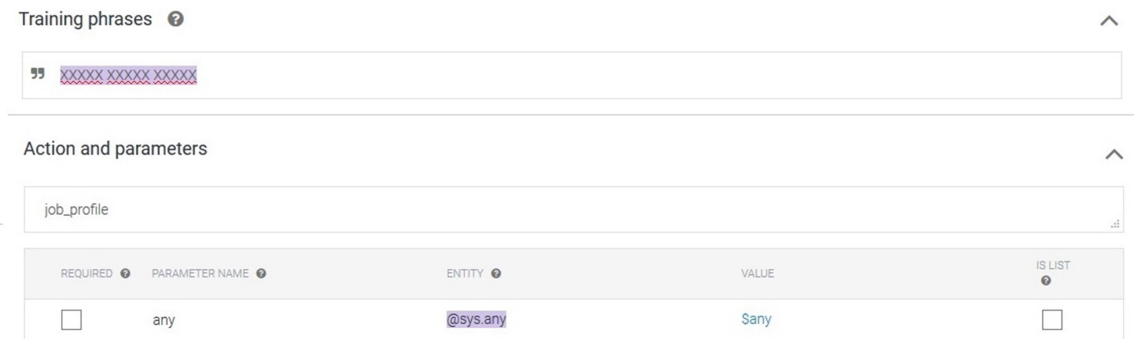


Ilustración 37. Regla para captar cualquier texto

Como esta regla captura cualquier texto, entraría en conflicto con cualquier otra regla que pudiera estar activa en el mismo instante. Por este motivo, cuando el motor de inferencia envía la respuesta generada debemos asegurarnos de cancelar el contexto *check_skills*, lo cual se consigue dando valor nulo al tiempo de permanencia del contexto (*lifespancount*).

```

#Context control
check_context_lifespan=0
for context in conversation_contexts:
    if (action=='job_profile' and "check_skills" in context["name"]) \
        or (action=='job_salary' and "confirm_profile" in context["name"]):
        check_context_name=context["name"]
        check_context_lifespan=context["lifespanCount"]
        check_context_param = context["parameters"]
        if debug == 'on':
            print("\t\tCONTEXT: check_skills")
            print("\tname:", check_context_name)
            print("\tlifespanCount:", check_context_lifespan)
            print("\tparameters:", check_context_param)
#Function result
if check_context_lifespan>0:
    assemble_result = {"fulfillmentText": speech,"source": "Inference engine",
        "outputContexts": [ {"name": check_context_name,"lifespanCount": 0} ] }
else:
    assemble_result = {"fulfillmentText": speech,"source": "Inference engine"}
return assemble_result

```

Ilustración 38. Cancelación de contextos por código

Por su parte, la aplicación *job_consultant* detecta que el *webhook* ha sido lanzado por la acción *job_profile* y gestiona los datos con el módulo de inferencia. El módulo de inferencia realiza el siguiente procesado:

- Carga el modelo de redes bayesianas ya entrenado (*job_profile_model_NB.sav*).
- Utiliza análisis sintáctico para extraer las *features* (figura 40) del texto de entrada en función de las características con las que ha sido entrenado el modelo (*job_profile_model_NB.sav*).
- Utiliza el modelo para predecir el perfil profesional del usuario.
- Compone y devuelve la respuesta a *Dialogflow*.

```

#Result composition
str_opening= "According to the description, I think you may be looking for job positions as "
str_body=job_profile.upper()
str_closing= " Are you ok with that?"
action_result = str_opening + str_body + str_closing

```

Ilustración 39. Composición del resultado del motor de inferencia

```
#Features extraction
grammar = r"""
NP: {<N.*>}
"""

index=0 #For future extension
stop_words=[]
list_chunk_bulletin=[] #Chunks in actual bulletin
sentence=textPreProcessing(sentence) #Remove quotation mark symbols
POS_tag=POS_tagging(sentence) #POS tagging
tree_phrase=grammar_chunk(grammar, POS_tag) #Chunking
list_chunk=chunk_lookup(tree_phrase, info=False, debug=False) #List of chunks
for key in list_chunk:
    key_lemma=chunk_lemma(key, 'n') #Get lemma of chunks
    match=False
    for feature in model_att:
        #Do not check stop words
        if key_lemma in stop_words:
            break
        #First check: most common features
        if key_lemma==feature:
            df_input.loc[index, feature]+=1
            match=True
            break
        #Second check: synonyms of most common features
        if synonyms==True:
            if match==False:
                for feature in model_att:
                    key_lemma=chunk_lemma(key, 'n')
                    if key_lemma in get_synonyms(feature, pos='NOUN'):
                        df_input.loc[index, feature]+=1
                        match=True
                        break
```

Ilustración 40. Implementación de la extracción de características

A continuación se muestran tres interacciones que conllevan la predicción de tres perfiles laborales distintos. Las categorías clasificables son:

```
{'NONE': 0, 'CLERK': 1, 'COLOR_PAINTER': 2, 'DIRECTOR_MANAGER': 3,
'EDUCATION_TEACHING': 4, 'ENGINEER': 5, 'GUARD_OFFICER_POLICE_SECURITY': 6,
'OPERATOR': 7, 'PRINTER': 8, 'SALES': 9}
```

Agent	Agent	Agent
<p>USER SAYS COPY CURL</p> <p>In my daily job I can perform specialized design work in connection with the preparation of plans, designs, details and estimates for water supply and distribution structures, water pump stations, pipelines and groundwater treatment facilities.</p>	<p>USER SAYS COPY CURL</p> <p>I use to perform clerical accounting work in the review, preparation, processing, reconciliation, and maintenance of accounting records and reports, or payroll and time keeping documents</p>	<p>USER SAYS COPY CURL</p> <p>A person in my charge patrols and safeguards City-owned buildings, parks, reservoirs, facilities, grounds, and equipment; answers questions and directs visitors/patrons</p>
<p>DEFAULT RESPONSE</p> <p>According to the description, I think you may be looking for job positions as ENGINEER Are you ok with that?</p>	<p>DEFAULT RESPONSE</p> <p>According to the description, I think you may be looking for job positions as CLERK Are you ok with that?</p>	<p>DEFAULT RESPONSE</p> <p>According to the description, I think you may be looking for job positions as GUARD_OFFICER_POLICE_SECURITY Are you ok with that?</p>

Ilustración 41. Resultados del motor de inferencia

5.5.5 Filtrado de ofertas laborales

Una vez asignado y confirmado el perfil laboral (por parte del usuario), se procede a preguntar al usuario por sus preferencias salariales a la vez que el contexto *profile_verified* activa el *intent job_salary*. Al detectar una respuesta que cumpla con las reglas preparadas ([figura 42](#)) se lanza la acción *job_salary* a través del *webhook*. En la aplicación *job_consultant* se realizan el filtrado según los siguientes pasos:

- Se asigna valor a variables internas que representan el salario mínimo y máximo en función de la casuística dada por la obtención de los datos:
 - El usuario solo aporta el mínimo esperado
 - El usuario aporta el mínimo y máximo esperado
 - El usuario aporta un valor medio esperado
- Se carga el *dataset* de boletines de trabajo etiquetado y extendido con información para filtrar (*job_buletin_dataset.sav*).
- Se filtran las ofertas que coinciden con el perfil laboral y cumplen el rango especificado.
- Se devuelve la lista con las ofertas encontradas. En el resultado se indica el número de ofertas que cumplen los criterios sobre el total de ofertas disponibles en la base de datos.

Training phrases ^

keep the same	about \$50000	from \$25000
the one I mentioned before	about 50000	from 25000
use the previous input	more than \$25000	Around \$50.000
don't need to change the range	more than 25000	Around 50.000
same as before	from \$25.000 to \$75.000	from 25.000 to 75.000

Ilustración 42. Frases de entrenamiento para el *intent job_salary*

Action and parameters ^


job_salary .it

REQUIRED	PARAMETER NAME	ENTITY	VALUE	IS LIST
<input type="checkbox"/>	number_range	@sys.number	\$number_range	<input type="checkbox"/>
<input type="checkbox"/>	number_from	@sys.number	\$number_from	<input type="checkbox"/>
<input type="checkbox"/>	number_to	@sys.number	\$number_to	<input type="checkbox"/>

Ilustración 43. Parámetros del *intent job_salary*

USER SAYS COPY CURL

from 25000 to 110000

 DEFAULT RESPONSE ▼

Here you can check the list of 56 JOB BULLETINS out of 101 matching your preferences for the GUARD_OFFICER_POLICE_SECURITY profile ...

```
// AIRPORT GUIDE 0845
042018.txt // AIRPORT POLICE OFFICER
3225 110906 Rev 060115.txt // ANIMAL
CONTROL OFFICER 4311 061518.txt //
CEMENT FINISHER SUPERVISOR 3354
120916.txt // CHIEF CLERK POLICE 1219
061215.txt // CHIEF CLERK POLICE 1249
083118.txt // CHIEF SECURITY OFFICER
3187 092818.txt // CUSTODIAN
SUPERVISOR 3176 042817 051117 REV.txt
// EXECUTIVE ADMINISTRATIVE
ASSISTANT 1117 083118.txt // FIRE
SPRINKLER // ...
```

Would you like to MODIFY
the SALARY RANGE?

Ilustración 44. Resultado del módulo de filtrado¹¹

La última frase del resultado se añade para preguntar al usuario si desea modificar el rango salarial para ampliar o acotar la búsqueda. En caso de recibir una respuesta negativa se da por finalizado el servicio y se cierra la conversación.

5.5.6 Integración del agente conversacional

Además de generar una URL donde directamente se puede depurar el agente conversacional creado a través del navegador web ([figura 46](#)), *Dialogflow* genera el código del *iframe* que puede ser incrustado en cualquier documento *html*:

```
<iframe
  allow="microphone;"
  width="350"
  height="430"
  src="https://console.dialogflow.com/api-client/demo/embedded/6c78f8e2-ea2c-41d4-b694-4b9e80284d8
b">
</iframe>
```

Ilustración 45. *iframe* para integración del agente conversacional

¹¹ Se muestran solo las primeras ofertas para no sobrecargar la salida por pantalla.

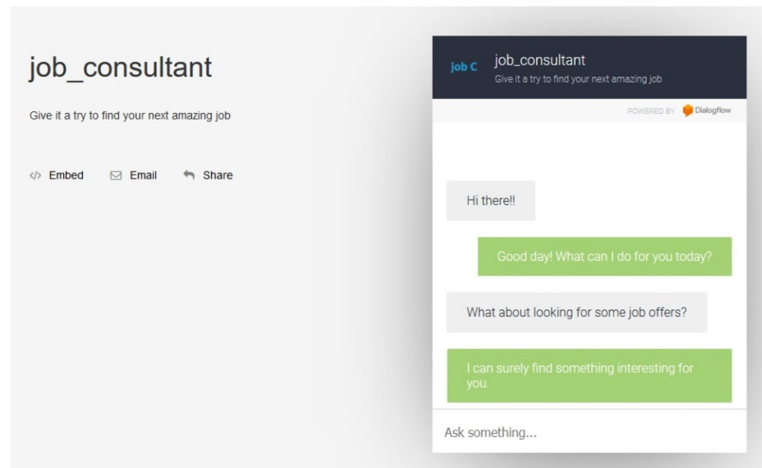


Ilustración 46. URL para depuración del agente conversacional

Para concluir esta sección, mostraremos los turnos de palabra de un diálogo completo.

Hi there!!

Greetings! How can I assist?

Can I get information about open positions?

Sure I can do. For narrowing down the look up, you have to write down any labor you can do along with each ability that will allow you to perform them.

I used to perform clerical accounting work in the review, preparation, processing, reconciliation, and maintenance of accounting records and reports, or payroll and time keeping documents.

After analyzing your expertise, I've considered that you may be interested on working as a CLERK. Is this right for you?

No, I'm not.

Sorry to hear that. Let's try again with the description. Type it again, please.

A person in my charge patrols and safeguards City-owned buildings, parks, reservoirs, facilities, grounds, and equipment; answers questions and directs visitors/patrons.

According to the description, I think you may be looking for job positions as GUARD_OFFICER_POLICE_SECURITY. Are you ok with that?

This time we got it.

Fine. In order to target the job offers that best fits on you, I will need you to type in your salary expectations.

Let's say from 25000 to 100000.

Here you can check the list of 42 JOB BULLETINS out of 101 matching your preferences for the GUARD_OFFICER_POLICE_SECURITY profile ... // AIRPORT GUIDE 0845 042018.txt // AIRPORT POLICE OFFICER 3225 110906 Rev 060115.txt // ANIMAL CONTROL OFFICER 4311 061518.txt // CEMENT FINISHER SUPERVISOR 3354 120916.txt // CHIEF CLERK POLICE 1219 061215.txt // COMMUNICATIONS INFORMATION REPRESENTATIVE 1461 101317 Revised.txt // CUSTODIAN SUPERVISOR 3176 042817 051117 REV.txt // EXECUTIVE ADMINISTRATIVE ASSISTANT 1117 083118.txt // FIRE SPRINKLER INSPECTOR 4240 062317 REVISED 070617 (2).txt // FIREFIGHTER 2112 011918.txt // HEAD CUSTODIAN SUPERVISOR 3178 050418.txt // LIBRARY ASSISTANT 1172 051118 (2).txt // PARK RANGER 1966 072718 REV 073118 (1).txt // PARKING ATTENDANT 3530 // ... **Would you like to MODIFY the SALARY RANGE?**

Yes, please.

Would you input again your salary expectations?

This time we can try from 75000 to 110000.

Please, have a look to the list of 9 OPEN POSITIONS I have found. The total number of offers for the GUARD_OFFICER_POLICE_SECURITY profile is 101 ... // CEMENT FINISHER SUPERVISOR 3354 120916.txt // FIRE SPRINKLER INSPECTOR 4240 062317 REVISED 070617 (2).txt // PLUMBING INSPECTOR 4231 042817 REV 051117.txt // POLICE SURVEILLANCE SPECIALIST 3687 052215.txt // SENIOR HEATING AND REFRIGERATION INSPECTOR 4247 121115.txt // SENIOR HOUSING INSPECTOR 4244 042718.txt // SENIOR HYDROGRAPHER 7264 030714.txt // SIGNAL SYSTEMS SUPERVISOR 3839 092818.txt // STRUCTURAL STEEL FABRICATOR SUPERVISOR 3794 060217.txt **Would you like to CHANGE your SALARY PREFERENCES?**

One moment, I need to change my work description.

That's a good way for redirecting our lookup. Maybe it's interesting to describe the job-related tasks you can perform.

In my daily job I can perform specialized design work in connection with the preparation of plans, designs, details and estimates for water supply and distribution structures, water pump stations, pipelines and groundwater treatment facilities.

Definitely you have many skills that may match the ENGINEER profile. Is this suitable?

Right this, yes.

Oh great. Let's continue then. Now it's time for knowing about your preferences.

Please, would you write down your desired range salary?

About 150000.

I've managed to collect 288 vacancies for the ENGINEER profile, which after filtering with your preferences has been bound to 25 JOB OFFERS ... // ASSOCIATE ZONING ADMINISTRATOR 7998 102618 REV 110818.txt // BUILDING MAINTENANCE DISTRICT SUPERVISOR 3190 042817 (1).txt // CHIEF AIRPORTS ENGINEER 7274 051515 (1).txt // CHIEF BUILDING OPERATING ENGINEER 5927 080516.txt // CHIEF OF DRAFTING OPERATIONS 7271 042018.txt // CHIEF OF PARKING ENFORCEMENT OPERATIONS 9180 031618.txt // DIRECTOR OF COMMUNICATIONS SERVICES 7625 042718.txt // DIRECTOR OF SYSTEMS 9375 042117.txt // ELECTRIC DISTRIBUTION MECHANIC SUPERVISOR 3873 102816.txt // ENGINEER OF SURVEYS 9486 101615.txt // GENERAL AUTOMOTIVE SUPERVISOR 3718 061915.txt // INDUSTRIAL CHEMIST 7834 020714.txt // INSTRUMENT MECHANIC SU // ... **Would you like to ADJUST your SALARY EXPECTATIONS?**

Not necessary.

Thanks for everything. Hope the information was useful and you get one of the proposed job. Bye!!

6. Conclusiones y Trabajo Futuro

Para concluir, se recogen en esta sección las conclusiones alcanzadas tras la realización del trabajo, junto con la evaluación del grado de completitud de cada uno de los objetivos y su aplicabilidad. Se evalúa en primer lugar el objetivo principal, para continuar luego con una valoración más detallada de cada uno de los subjetivos o partes constituyentes de la arquitectura del sistema (módulos). El capítulo, finaliza con una breve discusión sobre las posibles líneas de investigación futura que pueden continuar el trabajo expuesto en esta memoria.

6.1 Conclusiones

En líneas generales se puede concluir que la utilización de agentes conversacionales como interfaz con el usuario aporta una notable fluidez al proceso de búsqueda de empleo. El usuario conseguirá encontrar ofertas de empleo bien segmentadas según sus preferencias en apenas unos minutos en el marco de una conversación amena y enfocada a las necesidades de la tarea, asegurando que el usuario no olvida aportar información que resulte útil para acotar los resultados de la búsqueda. En vez de tener que rellenar formularios que pueden ser incompletos y/o ambiguos, el usuario se dedica a comunicarle al agente sus habilidades y competencias de la forma más natural posible, mediante el lenguaje. De esta forma, se consigue dar una gran visibilidad al producto y predisponer al usuario a utilizar este tipo de soluciones. Esto a su vez, abre la puerta para pensar en este tipo de interfaces como un enfoque preferente al planificar desarrollos de herramientas que impliquen intercambio directo de información con el usuario.

En cuanto a la implementación, ha quedado demostrado que es completamente viable la inserción de técnicas de procesamiento del lenguaje natural y de aprendizaje automático para resolver las tareas asociadas a cualquiera de las fases del ciclo de desarrollo y puesta en producción de la herramienta. Concretamente, se han aplicado técnicas de procesamiento del lenguaje natural para la extracción y selección de características del texto de entrada, algoritmos de aprendizaje (inferencia) para la clasificación del perfil laboral y una mezcla de ambos enfoques para la construcción del agente conversacional que actúa como interfaz. Enlazado con este aspecto, resulta interesante resaltar como la solución

desarrollada combina desarrollos propios con desarrollos de terceros (comerciales o de la comunidad de investigación). Los módulos disponibles a nivel comercial resuelven de forma cerrada problemas genéricos tales como el análisis sintáctico, reconocimiento de entidades, proporcionan arquitecturas y funcionalidades (*frameworks*) para el desarrollo de agentes conversacionales, etc. Las particularidades o necesidades específicas de cada problema deberán ser resueltas mediante adaptaciones, re-entrenamiento de modelos o desarrollos propios. Por ello, las empresas de soluciones comerciales han puesto especial atención en habilitar APIs para que los desarrolladores puedan adaptar o extender dichas herramientas de acuerdo a las necesidades concretas de sus proyectos. Esta flexibilidad permite reducir los tiempos de desarrollo y despliegue a la vez que asegura la obtención de altos rendimientos al trabajar sobre módulos que han sido desarrollados y testeados de forma muy rigurosa por equipos altamente cualificados.

Pasamos a comentar a continuación algunas reflexiones sobre los objetivos específicos del proyecto:

- Extracción de características

El uso de técnicas de procesamiento del lenguaje natural ha resultado fundamental para la extracción de información relevante sobre los datos de entrada (no estructurados). Es aquí, donde las técnicas de aprendizaje muestran su potencia al captar patrones y estructuras en los datos; habría resultado inviable capturar la complejidad y la casuística que puede llegar a darse en los datos de entrada mediante programación basadas en reglas. La caracterización de los documentos se ha basado en la frecuencia de aparición de palabras.

Un factor que ha demostrado ser altamente determinante es la cantidad y calidad de los datos. El simple hecho de pasar de un *dataset* con 676 documentos a otro con 2271 documentos propició un aumento del rendimiento de un 16% (extracción basada en nombres, [tabla 9](#)); incrementando desde un 44% (donde los modelos no tenían capacidad predictiva) hasta un 60%. El resto de metodologías y consideraciones han conseguido aportar un 12% adicional al rendimiento del sistema. Los mencionamos a continuación:

- Uso de nombres como atributos: el rendimiento de los clasificadores ha sido mayor al utilizar nombres frente a verbos, ya que estos son más diversos y por tanto más específicos a la hora de describir un perfil laboral.
- Sinonimia: el uso de la sinonimia no ha mejorado los resultados durante el entrenamiento. No obstante, se mantiene su uso durante la fase de servicio para

ayudar a representar mejor los distintos estilos o formas de expresión que pueda utilizar el usuario.

- Reducción de dimensionalidad: en vez de realizar la selección de características sobre el total de documentos, esta se acomete categoría a categoría. Esta metodología ayuda a seleccionar palabras más descriptivas de cada categoría, que habían sido desplazadas (en cuanto a frecuencia de aparición) por palabras más comunes cuando la selección se aplicaba sobre el total de documentos. Por encima del leve aumento del rendimiento (entorno al 3%), destaca que se consigue bajar notablemente el número de características necesarias para catalogar los documentos. Con esto se consigue mejorar la representatividad de las características obtenidas y aumentar la eficiencia del modelo que ahora procesará una cantidad menor de datos (en entrenamiento e inferencia).
- Eliminación de categorías con pocos ejemplos: tras analizar el rendimiento individual para la predicción de cada categoría se observaron resultados muy poco precisos para los perfiles laborales de los que se disponían de pocas muestras. Tras eliminar dichas categorías el rendimiento llega a alcanzar un 72%, resaltando así la importancia de tener un número abundante de muestras.
- Cribado de características genéricas: se detectó que el hecho de que algunas categorías a pesar de disponer de suficientes muestras no consiguieran altos rendimientos en la clasificación se debía a que compartían gran parte del conjunto de características que describen dicha clase con alguna otra clase. Para reducir el acoplamiento se trató de eliminar del conjunto de características aquellas que fueran compartidas con frecuencia entre varias categorías. Esta técnica no consiguió mejorar el rendimiento.
- Matriz *tf-idf*: como alternativa al cribado de características genéricas (reducción de la información), se probó a dar más peso a las palabras más específicas de cada perfil laboral mediante la construcción de la matriz *tf-idf* ([2.2.4 Clasificación de texto](#)) como forma de caracterizar los datos. Los rendimientos se mantuvieron en los mismos niveles.
- Eliminación de las categorías acopladas: Dejando fuera de la clasificación las categorías que se acoplan podemos fácilmente obtener un rendimiento del 85%, que puede ser mejorado si se sigue reduciendo aún más el número de categorías.
- Word vector: la utilización de representaciones densas para caracterizar los datos permite entrenar redes neuronales que alcanzan rendimientos similares a las técnicas basadas en representaciones dispersas (frecuencias). No obstante, este tipo de técnicas no son tan interpretables como las anteriores, razón por la que el grueso del proyecto se ha desarrollado en torno a la utilización de la matriz de frecuencias.

Se concluye por tanto, que los dos principales factores limitantes para alcanzar rendimientos más elevados han sido la escasez de datos (reduce capacidad de predicción) y el acoplamiento que presentan algunos perfiles laborales que son relativamente parecidos (reduce la precisión).

- Comparativa de algoritmos de aprendizaje.

Tal y como ya avanzaban algunos autores [26], se ha corroborado que los clasificadores bayesianos aportan mejores rendimientos que la redes neuronales en problemas de clasificación de texto. Es necesario anotar que para los procesos de entrenamiento supervisado que utilizan los modelos bayesianos y de redes neuronales es primordial disponer de un correcto etiquetado de las muestras. Dado que los algoritmos de agrupamiento (*clustering*) no han conseguido aportar etiquetados suficientemente consistentes, se optó por un etiquetado semi-guiado durante el propio proceso de adquisición de datos (*web scraping*). En este proceso se marca una clave de búsqueda para encontrar automáticamente un conjunto de descripciones laborales pertenecientes al mismo grupo.

Resulta interesante comentar que aunque en este trabajo se ha tenido que buscar la forma de etiquetar automáticamente las muestras, la mayoría de portales web para búsqueda de empleo ya disponen de bases de datos de ofertas laborales con un número abundante de muestras correctamente etiquetadas. Por tanto, les resultaría bastante directo adaptar la información y conseguir conjuntos de entrenamiento que aporten rendimientos bastante elevados.

- Agente conversacional como interfaz de usuario

El agente conversacional consigue guiar de forma efectiva el proceso de extracción de información a la vez que consigue mejorar la usabilidad del sistema y la experiencia del usuario al naturalizar la comunicación. La combinación de reglas de producción (*intents*) y contextos han demostrado ser técnicas bastante flexibles para guiar el flujo de la conversación. Por su parte, la utilización de técnicas de reconocimiento de entidades resulta bastante efectiva para localizar y extraer la información que se precisa del usuario.

6.2 Trabajo futuro

Con respecto a la interfaz de usuario, una ampliación bastante directa sería añadir nuevos roles al agente conversacional que permitan al usuario el uso de funcionalidades adicionales como configurar notificaciones cuando aparezcan nuevas ofertas de trabajo dado un perfil, asistencia para la elaboración de currículums bien organizados y completos, entrevistas de trabajo virtuales como preparación para entrevistas reales, etc. En estos casos el agente podría trabajar sobre distintos dominios de aplicación, por lo que en primer lugar tendría que detectar la tarea a realizar para posteriormente cargar la plantilla (*frame*) y activar las reglas de producción (*intents*) adecuadas para cada caso. En un esfuerzo algo más ambicioso se puede entrever que la mayor parte de las funcionalidades de las aplicaciones web podrían implementar su interfaz con el usuario de forma amena mediante el uso de agentes conversacionales, en especial cuando se conversa por voz. La idea subyacente es que cada funcionalidad tenga una API bien definida, completa y flexible y que el agente conversacional se dedique a rellenar los parámetros e información de entrada para la API.

Atendiendo a la clasificación de categorías laborales, se debe profundizar en la investigación de distintas alternativas que permitan desacoplar mejor aquellos perfiles laborales que sean similares. Un posible enfoque podría consistir en agrupar las muestras en categorías más genéricas y dejar que sea el usuario el que a través de sus preferencias decida si desea un trabajo más o menos específico; para por ejemplo distinguir entre los perfiles de 'ingeniero' y 'técnico'. Esta metodología sería solo posible si al etiquetar se añadieran dos atributos, la categoría principal y una categoría secundaria. Disponiendo de dos categorías, quizás podría dar mejor resultado trabajar con clasificadores en serie, de tal forma que uno de ellos se encargara de distinguir los perfiles más genéricos y a continuación un clasificador especializado (para cada perfil) tratará de distinguir las posibles subcategorías. Al ser la segunda etapa de clasificación mucho más específica, es probable que se pudiera conseguir desacoplar mejor las categorías similares. Por otra parte, también se plantea investigar los resultados de utilizar otras técnicas de clasificación de texto, como LDA ([2.2.4.1 Latent Dirichlet Allocation](#)). Aunque debido a que el algoritmo también se basa en frecuencia de palabras, es probable que el acoplamiento entre perfiles laborales propicie que la técnica no acabe siendo del todo efectiva. Queda abierta también la posibilidad de encontrar la forma de dar más o menos peso a los datos para aumentar la diferencia entre categorías similares. Algoritmos como los propuestos por Lesk [\[38\]](#) podrían ser utilizados a tal efecto. En último lugar, se propone como posible vía de investigación combinar las técnicas de NLP con algoritmos basados en *word vectors* para tratar de encontrar alguna combinación que consiga captar mejor la semántica de los textos a la hora de ser clasificados.

Bibliografía

- [1] Lombardero, L. (2015), *Trabajar en la era digital: tecnología y competencias para la transformación digital*. Lid Editorial
- [2] Neller, T. W. (2017), *AI education: Machine learning resources*. AI Matters, 3(2)
- [3] Jain, A. K. (2010), *Data clustering: 50 years beyond K-means*. *Pattern recognition letters*. 31(8), 651-666
- [4] Buchta, C., Kober, M., Feinerer, I., & Hornik, K. (2012), *Spherical k-means clustering*. *Journal of Statistical Software*, 50(10), 1-22
- [5] Blumberg, R., & Atre, S. (2003), *The problem with unstructured data*. *Dm Review*, 13(42-49), 62
- [6] Buneman, P., Davidson, S., Fernandez, M., & Suciu, D. (1997, January), *Adding structure to unstructured data*. In *International Conference on Database Theory* (pp. 336-350). Springer, Berlin, Heidelberg
- [7] Miller, G. A. (1995), *WordNet: a lexical database for English*. *Communications of the ACM*, 38(11), 39-41
- [8] Martin, J. H., & Jurafsky, D. (2009), *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River: Pearson/Prentice Hall.
- [9] Bird, S., Klein, E., & Loper, E. (2009), *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc..
- [10] Peres-Neto, P. R., Jackson, D. A., & Somers, K. M. (2005), *How many principal components? Stopping rules for determining the number of non-trivial axes revisited*. *Computational Statistics & Data Analysis*, 49(4), 974-997.

- [11] Crespo, R. G., Martínez, O. S., Lovelle, J. M. C., García-Bustelo, B. C. P., Gayo, J. E. L., & De Pablos, P. O. (2011), *Recommendation system based on user interaction data applied to intelligent electronic books*. *Computers in Human Behavior*, 27(4), 1445-1449.
- [12] Belkin, N. J., & Croft, W. B. (1992), *Information filtering and information retrieval: two sides of the same coin*. In *Communications of the ACM*.
- [13] Resnick, P., & Varian, H. R. (1997), *Recommender systems*. *Communications of the ACM*, 40(3), 56-59.
- [14] Adomavicius, G., & Tuzhilin, A. (2005), *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. *IEEE Transactions on Knowledge & Data Engineering*, (6), 734-749.
- [15] Linden, G., Smith, B., & York, J. (2003), *Amazon.com recommendations: Item-to-item collaborative filtering*. *IEEE Internet computing*, (1), 76-80.
- [16] Hofmann, T. (2003, July), *Collaborative filtering via gaussian probabilistic latent semantic analysis*. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval* (pp. 259-266). ACM.
- [17] Grosz, B. J., & Sidner, C. L. (1986), *Attention, intentions, and the structure of discourse*. *Computational linguistics*, 12(3), 175-204.
- [18] Chomsky, N. (1956), *Three models for the description of language*. *IRE Transactions on information theory*, 2(3), 113-124.
- [19] Padró, L., Collado, M., Reese, S., Lloberes, M., & Castellón, I. (2010), *Freeling 2.1: Five years of open-source language processing tools*. In *7th International Conference on Language Resources and Evaluation*.
- [20] Baeza-Yates, R. (2004, February), *Challenges in the interaction of information retrieval and natural language processing*. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 445-456). Springer, Berlin, Heidelberg.

- [21] Russell, S. J., & Norvig, P. (2016), *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.
- [22] Fortnow, L., & Homer, S. (2003), *A short history of computational complexity*. Boston University Computer Science Department.
- [23] Goodfellow, I., Bengio, Y., & Courville, A. (2016), *Deep learning*. MIT press.
- [24] Sutton, R. S., & Barto, A. G. (2011), *Reinforcement learning: An introduction*. MIT press.
- [25] Fiuza Pérez, M., & Rodríguez Pérez, J. C. (2000), *La regresión logística: una herramienta versátil*. *Nefrología*, 20(6), 495-500.
- [26] Yang, Y., & Liu, X. (1999, August), *A re-examination of text categorization methods*. In *Sigir* (Vol. 99, No. 8, p. 99).
- [27] Hossin, M., & Sulaiman, M. N. (2015), *A review on evaluation metrics for data classification evaluations*. *International Journal of Data Mining & Knowledge Management Process*, 5(2), 1.
- [28] Sagioglu, S., & Sinanc, D. (2013, May), *Big data: A review*. In 2013 International Conference on Collaboration Technologies and Systems (CTS), (pp. 42-47). IEEE.
- [29] Vargiu, E., & Urru, M. (2013), *Exploiting web scraping in a collaborative filtering-based approach to web advertising*. *Artif. Intell. Research*, 2(1), 44-54.
- [30] Myllymaki, J. (2002), *Effective web data extraction with standard XML technologies*. *Computer Networks*, 39(5), 635-644.
- [31] Moreno, I., Moreda, P., & Romá-Ferri, M. (2012), *Reconocimiento de entidades nombradas en dominios restringidos*. In *Actas del III Workshop en Tecnologías de la Informática* (pp. 41-57).
- [32] Robertson, S. (2004), *Understanding inverse document frequency: on theoretical arguments for IDF*. *Journal of documentation*, 60(5), 503-520.

- [33] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003), *Latent dirichlet allocation*. Journal of machine Learning research, 3(Jan), 993-1022.
- [34] Beel, J., Breiting, C., & Langer, S. (2017), *Evaluating the CC-IDF citation-weighting scheme: how effectively can 'Inverse Document Frequency'(IDF) be applied to references*. Proceedings of the 12th iConference.
- [35] Turing, A. M. (2004), *Computing machinery and intelligence (1950)*, *The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Ed. B. Jack Copeland. Oxford: Oxford UP, 433-64.
- [36] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. (1977), *GUS, a frame-driven dialog system*. Artificial intelligence, 8(2), 155-173.
- [37] Reiter, E., & Dale, R. (2000), *Building natural language generation systems*. Cambridge university press.
- [38] Banerjee, S., & Pedersen, T. (2002, February), *An adapted Lesk algorithm for word sense disambiguation using WordNet*. In International conference on intelligent text processing and computational linguistics (pp. 136-145). Springer, Berlin, Heidelberg.

Apéndice A

Este apéndice contiene el código desarrollado para la fase de extracción de características, análisis de las estrategias más efectivas y entrenamiento de los modelos de aprendizaje. Las rutinas programadas incluyen desarrollos propios y funcionalidades de módulos de terceros, programados en *Python*. Las funcionalidades y metodologías que implementan los programas han sido explicadas y referenciadas en la memoria del trabajo. Cualquier detalle no mencionado explícitamente en la redacción, estará explicado a través de los comentarios adjuntos en el código.

A.1 Módulo de pre-procesado: `M_text_pre_processing.py`

```
#MODULE: TEXT PRE-PROCESSING

def textPreProcessing(text):

    if not isinstance(text,str):      #Data type check out
        text=str(text)
    #LOWER CASE
    text = text.lower()
    #Break line symbol
    text=text.rstrip()              #Right strip
    text = text.replace("\\", "")
    text = text.replace(".n", "")
    #Remove INITIAL SYMBOLS
    item_e=5                          #Symbol check window
    symbols=['[',']','\','\','\']
    count=[1 for key in range(item_e) if text[key] in symbols]
    if len(count)>0:
        text=text[len(count):len(text)]
    #Remove ENDING SYMBOLS
    symbols=['[',']','\','\','\']
    count=[1 for key in range(len(text)-item_e,len(text)) if text[key] in
symbols]
    if len(count)>0:
        text=text[0:len(text)-len(count)]

    return text

def splitBySemicolon(paragraph_in):

    paragraph=str(paragraph_in)      #Cast to string
    p_s,p_n,p_e=0,0,len(paragraph)  #Start, next & end position
    sentences=[]                      #Paragraph sentences
    while(paragraph[p_s:p_e].find(';')!=-1):
        p_n=paragraph[p_s:p_e].find(';')
        sentences.append(paragraph[p_s:p_s+p_n])
        p_s+=p_n+1

    return sentences

def printPOS(POS_tag,tokens,sentence,info_mode=0):

    if info_mode==0:
        print('SENTENCE\n',sentence)
```

```

    print('TOKENS\n',tokens)
    print('POS_TAG\n',POS_tag)
if info_mode==1:
    print('POS_TAG\n',POS_tag)
if info_mode==2:
    print('TOKENS\n',tokens)
    print('POS_TAG\n',POS_tag)

```

A.2 Módulo de análisis sintáctico: M_syntactic_analysis

```

#MODULE: SYNTACTIC ANALYSIS

#NLTK functions
from nltk import pos_tag,RegexpParser,tree,word_tokenize
from nltk.stem import WordNetLemmatizer
#Wordnet corpus
from nltk.corpus import wordnet

def POS_tagging(sentence):

    if not isinstance(sentence,str):          #Data type check out
        sentence=str(sentence)
    tokens=word_tokenize(sentence)          #Get tokens for the input sentence
    POS_tag=pos_tag(tokens)                 #Get POS for the tokenized sentence

    return POS_tag

def grammar_chunk(grammar,POS_tag):         #Grammar chunking

    cp=RegexpParser(grammar)                #Built a parser using the defined
grammar
    tree_phrase=cp.parse(POS_tag)           #Get chunks from each sentence
based on POS

    return tree_phrase

def chunk_lookup(tree_phrase,info=False,debug=False):

    list_chunk=[]                           #Chunk list
    for sheet in tree_phrase:
        if isinstance(sheet, tree.Tree):
            if debug==True:
                print('Chunk item:',len(list_chunk)+1)
            str_chunk=''                     #Chunk extraction string
            for key in sheet:
                if len(str_chunk)==0:
                    str_chunk=str(key[0])
                else:
                    str_chunk+=' '+str(key[0])
                if debug==True:
                    print(key)
            list_chunk.append(str_chunk)     #Update chunk list

    if info==True:
        print('\nNumber of chunks found:',len(list_chunk))
        print('Chunk list',list_chunk)

    return list_chunk

```



```

def chunk_lemma(chunk, key):

    if not isinstance(chunk, str):
        chunk=str(chunk) #Data type check out

    chunk_split=chunk.split() #Get all words of the chunk
    #Chunk lemma composition
    chunk_composition=WordNetLemmatizer().lemmatize(chunk_split[0],key)
    if len(chunk_split)>1:
        for index in range(1,len(chunk_split)):
            chunk_composition+='
'+WordNetLemmatizer().lemmatize(chunk_split[index],key)

    return chunk_composition

def get_synonyms(word, pos=None):
    wordnet_pos = {"NOUN": wordnet.NOUN, "VERB": wordnet.VERB, "ADJ":
wordnet.ADJ, "ADV": wordnet.ADV}
    if pos:
        synsets = wordnet.synsets(word, pos=wordnet_pos[pos])
    else:
        synsets = synsets(word)
    synonyms = []
    for synset in synsets:
        synonyms += [str(lemma.name()) for lemma in synset.lemmas()]
    synonyms = [synonym.replace("_", " ") for synonym in synonyms]
    synonyms = list(set(synonyms))
    synonyms = [synonym for synonym in synonyms if synonym != word]
    return synonyms

def items_over_threshold(dict_input, threshold):

    list_output=[]
    for key,value in dict_input.items():
        if value>=threshold:
            list_output.append(key)

    return list_output

def items_over_threshold_limited(dict_input, threshold, max_items):

    list_output=[]
    sorted_dict_input = sorted(dict_input.items(), key=lambda kv:
kv[1],reverse = True)

    for key,value in sorted_dict_input:
        if len(list_output)==max_items:
            break
        if value>=threshold:
            list_output.append(key)

    return list_output

```

A.3 Datasets

A3.1 Sobre el dataset de boletines de trabajo

A.3.1.1010 Dataset_bulletins_reader.ipynb

Libraries import

```
import re                #regex
import pickle           #Data saving library
from os import walk    #System directory handling
```

Get List of Job offers

```
folder='Dataset/Corpus/'

#Job bulletin filenames
job_filenames= []
for (dirpath, dirnames, filenames) in walk(folder):
    job_filenames.extend(filenames)
    #f.extend(dirnames)
    #break #Only the target top directory

#Job names
job_names={}
for filenames in job_filenames:
    #Job dictionary
    filenames_split = re.findall(r'\w+', str(filenames))
    index=0 #Initialize
    #ID look up
    for item in filenames_split:
        index+=1 #Update index
        if item.isdigit():
            break #Stop when finding the ID code
    #Job name composition
    job_name=filenames_split[0]
    for key in range(1,index-1):
        job_name+=' '+filenames_split[key]
    #Insert an entry in the job dictionary
    #First time entry
    if job_name not in job_names:
        job_names[job_name]=filenames_split[index-1]
        #Second time entry
    else:
        job_names[job_name+'_'+str(filenames_split[index-1])]=filenames_split[index-1]

print('Number of job filenames:',len(job_filenames))
display(job_filenames[0:5])

print('\nNumber of jobs:',len(job_names),'\n')
n_print,e_print=0,5
for key,value in job_names.items():
    print(key,':',value)
    n_print+=1
    if n_print>=e_print:
        break

#Saving job bulletins info
file_path ='job_filenames.sav'
pickle.dump(job_filenames, open(file_path, 'wb'))
file_path ='job_names.sav'
pickle.dump(job_names, open(file_path, 'wb'))
```

Get duties, requirements and salary information

```
#Job bulletin attribute list
num_job_att=3 #Number of jobs attributes
```

```

list_job_att=[] #Attribute list for each job // [0]=Duties
List,[1]=Requirements List,[2]=Salary List
list_att_missing=[]
for key in range(num_job_att):
    list_job_att.append([]) #List of lists of attributes
    list_att_missing.append([]) #List of all job with some missing
    field

#Loop all job offer files for extracting attributes information
count=0
list_att=[] #Job attribute list
read_register=False #Reading flag
for file in job_filenames:
    list_line = open(folder+file,"r") #Creating a list with each line
    #Loop through lines
    read_next=0

    att01_in,att02_in,att03_in=False,False,False #Attribute detection
    flag
    for line in list_line:
        #Closure attempt
        if line.isupper():
            read_register=False
        #Attribute reading case
        if read_next!=0 and len(line)>1:
            #Add line
            if read_register==True:
                list_att.append(line)
            #Finish reading
            else:
                list_job_att[read_next-1].append(list_att)
                list_att=[] #Clear Job attribute list
                read_next=0

        #Token detection
        if line.strip() in ['DUTIES','DUTIES AND RESPONSIBILITIES']:
            read_next=1
            att01_in=True
            read_register=True
        if line.strip() in
['REQUIREMENT','REQUIREMENTS','REQUIREMENT/MINIMUM QUALIFICATION',
'REQUIREMENT/MIMINUMUM
QUALIFICATION','REQUIREMENT/ MINIMUM QUALIFICATION',
'REQUIREMENT/MINUMUM
QUALIFICATION','REQUIREMENT/MINIMUM QUALIFICATIONS',
'REQUIREMENTS/MINUMUM
QUALIFICATIONS','REQUIREMENTS/ MINIMUM QUALIFICATIONS',
'REQUIREMENTS/MINIMUM REQUIREMENTS','MINIMUM
REQUIREMENTS:',
'REQUIREMENTS/MINIMUM
QUALIFICATIONS','REQUIREMENT/MINIMUM QUALIFCAITON',
'REQUIREMENTS/MINIMUM QUALIFCATIONS']:
            read_next=2
            att02_in=True
            read_register=True
        if line.strip() in ['ANNUAL SALARY','ANNUALSALARY']:
            read_next=3
            att03_in=True
            read_register=True

    if att01_in==False:

```

```

        #print('Duty not in',file)
        list_att_missing[0].append(file)
    if att02_in==False:
        #print('Requirement not in',file)
        list_att_missing[1].append(file)
    if att03_in==False:
        #print('Salary not in',file)
        list_att_missing[2].append(file)

#Saving job attribute info // [0]=Duties List,[1]=Requirements
List,[2]=Salary List
file_path = 'list_job_att.sav'
pickle.dump(list_job_att, open(file_path, 'wb'))

```

Check missing sections in files for removing from the data set

```

print('Missing attribute:')
print('\tDuty',len(list_att_missing[0]))
if len(list_att_missing[0])>0:
    display(list_att_missing[0])
print('\tRequirement',len(list_att_missing[1]))
if len(list_att_missing[1])>0:
    display(list_att_missing[1])
print('\tSalary',len(list_att_missing[2]))
if len(list_att_missing[2])>0:
    display(list_att_missing[2])

```

Sample display

```

index=2
print(job_filenames[index])
print('\nDUTIES:',list_job_att[0][index])
print('\nREQUIREMENTS',list_job_att[1][index])
print('\nSALARY',list_job_att[2][index])

```

A3.1.2 011_Dataset_group_by_name.ipynb

Libraries import

```

import re #regex
import pickle #Data saving library
import datetime #Timestamp library
from statistics import mean,stdev #Statistics library

```

Loading job bulletins information

```

file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))

```

Finding the most common job names for grouping the jobs (learning from data)

```

dict_job_group={}
for name in job_names:
    name_split=re.findall(r'\w+', str(name))

```

```

    for word in name_split:
        if word not in dict_job_group:
            dict_job_group[word]=1                #First time entry: create
entry
        else:
            dict_job_group[word]+=1                #Second time entry: update
entry

print('Number of words present in the job names:',len(dict_job_group))
print('Ocurrences statistics')
print('\tMaximum:',round(max(list(dict_job_group.values())),2))
print('\tAverage:',round(mean(list(dict_job_group.values())),2))
print('\tStandard
deviation:',round(stdev(list(dict_job_group.values())),2))
#print(dict_job_group)
sorted_chunk_dict = sorted(dict_job_group.items(), key=lambda kv: kv[1])
print(sorted_chunk_dict)

```

Filtering functions

```

def items_over_threshold(dict_input,threshold):

    list_output=[]
    for key,value in dict_input.items():
        if value>=threshold:
            list_output.append(key)
    list_output.sort()

    return list_output

def clean_word_list(list_input,list_exception):

    list_output=[]
    for word in list_input:
        if word not in list_exception:
            list_output.append(word)

    return list_output

```

Displaying the coverage percentage according to the number of occurrences of the words

```

total_words=len(dict_job_group)
#Statistics
for threshold in range(11,21):
    list_common_job_words=items_over_threshold(dict_job_group,threshold)
    print('\nMinimum word occurrences:',threshold)
    print('\tNumber of features:',len(list_common_job_words))
    print("\tCoverage percentage
{0:.1f}".format(len(list_common_job_words)/total_words*100.0))

```

Selecting most common words given a threshold

```

#Selecting most common words
threshold=15
list_common_job_words=items_over_threshold(dict_job_group,threshold)
print('\nMost common words(th =',threshold,'):',len(list_common_job_words))
print(list_common_job_words)

threshold=20

```

```

list_common_job_words=items_over_threshold(dict_job_group,threshold)
print('\nMost common words(th =',threshold,'):',len(list_common_job_words))
print(list_common_job_words)

list_exception=['AND','OF','PRINCIPAL','SENIOR']
list_common_job_words=clean_word_list(list_common_job_words,list_exception)
print('\nSelected job groups:(th
=',threshold,'):',len(list_common_job_words))
print(list_common_job_words)
#Saving common job words info
coverage=round(len(list_common_job_words)/total_words*100.0,1)
file_path ='list_common_job_words_'+str(coverage)+'.sav'
pickle.dump(list_common_job_words, open(file_path, 'wb'))

```

Creating an ID for every selected group

```

dict_ID=0
list_common_job_words.insert(0,'OTHER')           #Default category
dict_job_group={}
for key in list_common_job_words:
    dict_job_group[key]=dict_ID
    dict_ID+=1
print('\nDictionary ID',dict_job_group)

```

Job groups tagged by hand

```

dict_job_group_tagged={'NONE':0,'AGRICULTURE, CATTLE AND
FISHING':1,'BANKING, ACCOUNTING AND INSURANCE':2
    , 'BUILDING AND CONSTRUCTION':3,'BUSINESS
ADMINISTRATION':4,'CHEMIST, BIOLOGY AND ENVIRONMENT':5
    , 'COMPUTER AND PROGRAMMING':6,'CUSTOMER SERVICE,
SECRETARIAT AND CLERKS':7,'EDUCATION AND FORMATION':8
    , 'ENGINEERING AND TECHNICIANS':9,'HEALTH, SOCIAL
SERVICES AND CARE':10,'HUMAN RESOURCES':11
    , 'LANGUAGES':12,'LAWYERS':13,'LEISURE AND
ENTERTAINMENT':14,'MARKETING, COMUNICATION AND PUBLISHING':15
    , 'MECHANICS AND MAINTENANCE':16,'MULTIMEDIA,
DESIGNERS AND ARTISTS':17,'OTHERS':18
    , 'PRODUCTION, QUALITY AND
R&D':19,'SECURITY':20,'SHOPPING, LOGISTICS AND STOCK':21
    , 'SPORTS AND BEAUTY ':22,'TOURISM, HOTEL AND
CATERING':23
    , 'TRADE AND SALES':24,'TRANSPORT':25,'WORKERS AND
OPERATORS':26}

```

After tagging the dataset we notice that only 21 categories are present

```

dict_job_group_tagged={'NONE':0,'BANKING, ACCOUNTING AND INSURANCE':1
    , 'BUILDING AND CONSTRUCTION':2,'BUSINESS
ADMINISTRATION':3,'CHEMIST, BIOLOGY AND ENVIRONMENT':4
    , 'COMPUTER AND PROGRAMMING':5,'CUSTOMER SERVICE,
SECRETARIAT AND CLERKS':6,'EDUCATION AND FORMATION':7
    , 'ENGINEERING AND TECHNICIANS':8,'HEALTH, SOCIAL
SERVICES AND CARE':9,'HUMAN RESOURCES':10
    , 'LAWYERS':11,'LEISURE AND
ENTERTAINMENT':12,'MARKETING, COMUNICATION AND PUBLISHING':13
    , 'MECHANICS AND MAINTENANCE':14,'MULTIMEDIA,
DESIGNERS AND ARTISTS':15

```

```

        , 'PRODUCTION, QUALITY AND
R&D':16, 'SECURITY':17, 'SHOPPING, LOGISTICS AND STOCK':18
        , 'TRADE AND SALES':19, 'TRANSPORT':20, 'WORKERS AND
OPERATORS':21}
print(dict_job_group_tagged)
#Saving job group info
file_path = 'list_job_group.sav'
pickle.dump(list_common_job_words, open(file_path, 'wb'))
file_path = 'dict_job_group.sav'
pickle.dump(dict_job_group, open(file_path, 'wb'))
file_path = 'dict_job_group_tagged.sav'
pickle.dump(dict_job_group_tagged, open(file_path, 'wb'))

```

A.3.1.3 012_Dataset_tagged_filter.ipynb

Test settings

```

synonyms=True
use_duty_and_req=False

```

Libraries import

```

#SYSTEM MODULES
import re                #regex
import time              #Timestamp library
import pickle            #Data saving library
import numpy as np      #Mathematical functions & data handling
import pandas as pd     #Data & statistical analysis

#USER MODULES
#Pre processing functions
from M_text_pre_processing import textPreProcessing
#Syntactic analysis
from M_syntactic_analysis import
chunk_lookup, chunk_lemma, grammar_chunk, items_over_threshold, POS_tagging
from M_syntactic_analysis import get_synonyms

```

Loading job bulletins information

```

file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))
job_name_list=list(job_names.keys())
file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_att.sav'
list_job_att = pickle.load(open(file_path, 'rb'))

```

Loading features information

```

df_category_tagged=pd.read_excel('Job_group_tagged.xlsx', index_col=None)

```

Getting salary range

```

salary_min=[]
salary_max=[]
att_salary=list_job_att[2].copy()

```

```

for salary_info_list in att_salary:
    #Compound string from ininput list
    salary_info=""
    for entry in salary_info_list:
        salary_info+=entry
    #Regular expresion for extracting numbers
    salary_bounds=re.findall(r"[-+]?[d*]\.d+|[-+]?[d*]\.d+|\d+", salary_info)
    #Removing "," & "." before casting to int
    salary_bounds=[salary.replace(",","") for salary in salary_bounds]
    salary_bounds=[salary.replace(".", "") for salary in salary_bounds]
    #Casting from string to int
    salary_bounds=[int(i) for i in salary_bounds]
    #Selectin minimum/maximum salary
    salary_bounds.sort()
    if len(salary_bounds)>0:
        salary_min.append(salary_bounds[0])
        salary_max.append(salary_bounds[len(salary_bounds)-1])
    else:
        salary_min.append(0.0)
        salary_max.append(0.0)

```

Using the model to clasify the bulletins

```

folder='Model/'
file_path = folder+'job_profile_model_NB.sav'
model = pickle.load(open(file_path, 'rb'))
file_path = folder+'job_profile_att_model_NB.sav'
model_att = pickle.load(open(file_path, 'rb'))

```

Input dataframe

```

df_input=pd.DataFrame() #Empty dataframe
for job in model_att:

df_input.insert(df_input.shape[1],job,np.ravel(np.zeros((len(job_name_list)
,1)).astype(int)),allow_duplicates=False)
def list_item_combine(a,b):

    c=[]
    if len(a)==len(b) and type(a[0])==type(b[0]):
        for item in range(len(a)):
            if isinstance(a[0],str): #Data type check out
                c.append(a[item]+' \n'+b[item])
            else:
                c.append(a[item]+b[item])
    else:
        print('List must hava the same length nad type')

    return c
if use_duty_and_req==True:
    list_job_att[0]=list_item_combine(list_job_att[0],list_job_att[1])

```

Extracting features

```

t0 = time.time()

index=0
grammar = r""

```



```

NP: {<N.*>}
"""
stop_words=[]
for sentence in list_job_att[0]:
    list_chunk_bulletin=[] #Chunks in
actual_bulletin
    sentence=textPreProcessing(sentence) #Remove
quotation mark symbols
    POS_tag=POS_tagging(sentence) #POS
tagging
    tree_phrase=grammar_chunk(grammar,POS_tag) #Chunking
    list_chunk=chunk_lookup(tree_phrase,info=False,debug=False) #List of
chunks
    for key in list_chunk:
        key_lemma=chunk_lemma(key,'n') #Get lemma
of chunks
        match=False
        for feature in model_att:
            #Do not check stop words
            if key_lemma in stop_words:
                break
            #First check: most common features
            if key_lemma==feature:
                df_input.loc[index,feature]+=1
                match=True
                break
            #Second check: synonyms of most common features
            if synonyms==True:
                if match==False:
                    for feature in model_att:
                        key_lemma=chunk_lemma(key,'n')
                        if key_lemma in get_synonyms(feature,pos='NOUN'):
                            df_input.loc[index,feature]+=1
                            match=True
                            break
        #Bulletin index update
        index+=1

tf=time.time()-t0
print('\nProcessing time:',round(tf, 2))
print('File size:',df_input.shape)

```

Making predictions

```

x_input=df_input.values[:,:]
y_out = model.predict(x_input)

```

Database with predicted job profiles

```

df_buletin_dataset=pd.DataFrame(job_name_list,columns=['Job_name'])
df_buletin_dataset.insert(df_buletin_dataset.shape[1],'Profile',y_out,allow
_duplicates=False)
df_buletin_dataset.insert(df_buletin_dataset.shape[1],'Salary_min',salary_m
in,allow_duplicates=False)
df_buletin_dataset.insert(df_buletin_dataset.shape[1],'Salary_max',salary_m
ax,allow_duplicates=False)
df_buletin_dataset.insert(df_buletin_dataset.shape[1],'Bulletin',job_filesn
ames,allow_duplicates=False)
print('File size:',df_buletin_dataset.shape)
display(df_buletin_dataset.iloc[10:20,:])

```

```
#Save into excel file
if synonyms==True:
    syn_option='_synonym'
else:
    syn_option=''
name='df_job_pred.xlsx'
writer = pd.ExcelWriter(name)
df_buletin_dataset.to_excel(writer,'df_job_pred',index=False)
writer.save()
#Save filtered buletin dataset
file_path =folder+'job_buletin_dataset.sav'
pickle.dump(df_buletin_dataset, open(file_path, 'wb'))
```

A.3.2 Sobre el *dataset* de descripciones de perfiles laborales

A.3.2.1 010_Dataset_descriptions_reader.ipynb

Libraries import

```
import re #regex
import pickle #Data saving library
import pandas as pd #Data & statistical analysis
import os #System directory handling
from os import walk
main_dir=".\\Dataset\\job_category"
dict_job_category={'NONE':0}
list_job_category=[name for name in os.listdir(main_dir) if
os.path.isdir(os.path.join(main_dir, name))]

key=1
for job in list_job_category:
    dict_job_category[job]=key
    key+=1

print('Number of categories:',len(dict_job_category))
print(dict_job_category)
#Saving job group info
file_path ='dict_job_group_tagged.sav'
pickle.dump(dict_job_category, open(file_path, 'wb'))
```

Get List of Job offers

```
folder=main_dir

job_names={}
list_job_att=[]
job_filenames=[]
for category in list_job_category:
    #Job bulletin filenames
    job_bulletins=[]
    for (dirpath, dirnames, filenames) in walk(folder+'\\'+category):
        job_bulletins.extend(filenames)
    job_filenames.extend(job_bulletins)
    #Job names
    list_att=[] #Job attribute list
    read_register=False #Reading flag
    for filename in job_bulletins:
        #Insert an entry in the job dictionary
        #First time entry
```

```

filename_clean=filename[0:len(filename)-4]
if filename_clean not in job_names:
    job_names[filename_clean]=category
#Second time entry
else:
    job_names[filename_clean+'_'+category]=category

#Job description
list_line = open(folder+'\\'+category+'\\'+filename,"r")
#Creating a list with each line
#Loop through lines
read_next=0
for line in list_line:
    #Closure attempt
    if line.isupper():
        read_register=False
    #Attribute reading case
    if read_next!=0 and len(line)>1:
        #Add line
        if read_register==True:
            list_att.append(line)

    #Reading token detection
    if line.strip() == 'DUTIES':
        read_next=1
        read_register=True

#Finish reading
list_job_att.append(list_att)
list_att=[] #Clear Job attribute list
read_next=0 #Reset reading token

print('Number of job filenames:',len(job_filenames))
display(job_filenames[0:5])

print('\nNumber of jobs:',len(job_names),'\n')
n_print,e_print=0,5
for key,value in job_names.items():
    print(key,':',value)
    n_print+=1
    if n_print>=e_print:
        break
#Saving job bulletins info
file_path ='job_filenames.sav'
pickle.dump(job_filenames, open(file_path, 'wb'))
file_path ='job_names.sav'
pickle.dump(job_names, open(file_path, 'wb'))

```

Data frame with tagged samples

```

df_category_tagged=pd.DataFrame(list(zip(job_names.keys(),job_names.values(
))), columns=['JOBS','CATEGORY'])
print(df_category_tagged.shape)
display(df_category_tagged.head(4))
#Save into excel file
name='Job_group_tagged.xlsx'
writer = pd.ExcelWriter(name)
df_category_tagged.to_excel(writer,'Tagged_categories',index=False)
writer.save()

```

Sample display

```
print('\nlist_job_duties:',len(list_job_att))
display(list_job_att[0:3])          #list_job_duties
#Saving job attribute info // 0=Duties(1),1=Requirements(2),2=Salary(3)
file_path ='list_job_att.sav'
pickle.dump(list_job_att, open(file_path, 'wb'))
```

A.3.2.2 011_Web_scraping

Libraries import

```
import re                #Regular expresions
import os                #System functions
import urllib.request    #url handling
from bs4 import BeautifulSoup #Web scraping
```

Text processing functions

```
def clean_title(title):
    return title[1:title[1:len(title)].find('"')+1]

def clean_li(li):          #Remove the expression z) from the input
    string (where z=any number)
    return li[li.find('')+2:len(li)]
```

Text extraction and file creation functions

```
def get_job_duty(url,initial_tag,info='off'):

    html=urllib.request.urlopen(url).read().decode()
    soup=BeautifulSoup(html)
    #Main section title
    title=clean_title(soup.h1.text)
    if info=='on':
        print('\n',title,'\n')
    i_tag = soup.find(initial_tag)
    #Get every <div> and <ul> element following the initial tag
    divs_tags=[]
    for next_sibling in i_tag.findNextSiblings():
        if next_sibling.name in ['div','ul']:
            divs_tags.append(next_sibling)
    li_tags=[]
    #Get <li> texts inside the filtered elements
    for div in divs_tags:
        for li in div.findAll('li'):
            li_tags.append(clean_li(li.text))
            if info=='on':
                print(li.text)
    return title,li_tags

def create_job_bulletin(title,attributes,directory, folder):

    #Check destination directory
    if not os.path.exists('./'+directory):
        os.mkdir(directory+'/')
    #Check destination folder
    if not os.path.exists('./'+directory+'\\'+folder):
```

```

    os.mkdir(directory+'\\'+folder)
#Creating text file
text_file = open(directory+'\\'+folder+'\\'+title+'.txt','w+')
text_file.write(title)
text_file.write('\n\nDUTIES\n\n')
for attribute in attributes:
    text_file.write(attribute+' ')

```

Links extraction functions (from the home page)

```

def get_links(url,info='off'):

    html=urllib.request.urlopen(url).read().decode()
    soup=BeautifulSoup(html)
    tags=soup('a')
    links = []
    for tag in tags:
        links.append(tag.get('href'))
        if info=='on':
            print(tag.get('href'))

    return links
def filter_links(links,key,info='off'):

    filtered_list=[]
    key_lower=key.lower()
    for link in links:
        link_lower=link.lower()
        if link_lower.find(key_lower)>0:
            filtered_list.append(link)
    if info=='on':
        print(len(filtered_list),'links found for',key)

    return filtered_list

```

Analyzing the number of job descriptions according to key words for assembling the job categories

```

job_links=get_links('https://dot-job-
descriptions.careerplanner.com/',info='off')
tax_jobs=filter_links(job_links,'tax',info='on')
account_jobs=filter_links(job_links,'account',info='on')
financial_jobs=filter_links(job_links,'financ',info='on')
insurance_jobs=filter_links(job_links,'insurance',info='on')
print('\n')
building_jobs=filter_links(job_links,'building',info='on')
architect_jobs=filter_links(job_links,'architect',info='on')
construction_jobs=filter_links(job_links,'construction',info='on')
print('\n')
manager_jobs=filter_links(job_links,'manager',info='on')
business_jobs=filter_links(job_links,'business',info='on')
director_jobs=filter_links(job_links,'director',info='on')
administration_jobs=filter_links(job_links,'administration',info='on')
print('\n')
chemist_jobs=filter_links(job_links,'chemi',info='on')
biology_jobs=filter_links(job_links,'biolog',info='on')
environment_jobs=filter_links(job_links,'environment',info='on')
print('\n')
service_jobs=filter_links(job_links,'service',info='on')

```

```

customer_jobs=filter_links(job_links,'customer',info='on')
print('\n')
clerk_jobs=filter_links(job_links,'clerk',info='on')
secretariat_jobs=filter_links(job_links,'secretar',info='on')
print('\n')
quality_jobs=filter_links(job_links,'quality',info='on')
engineer_jobs=filter_links(job_links,'engineer',info='on')
technician_jobs=filter_links(job_links,'technic',info='on')
print('\n')
nurse_jobs=filter_links(job_links,'nurs',info='on')
health_jobs=filter_links(job_links,'health',info='on')
social_jobs=filter_links(job_links,'social',info='on')
medical_jobs=filter_links(job_links,'medical',info='on')
psychologist_jobs=filter_links(job_links,'psycholog',info='on')
print('\n')
mechanic_jobs=filter_links(job_links,'mechanic',info='on')
maintenance_jobs=filter_links(job_links,'maintenance',info='on')
print('\n')
guard_jobs=filter_links(job_links,'guard',info='on')
police_jobs=filter_links(job_links,'police',info='on')
officer_jobs=filter_links(job_links,'officer',info='on')
security_jobs=filter_links(job_links,'security',info='on')
print('\n')
dye_jobs=filter_links(job_links,'dye',info='on')
print_jobs=filter_links(job_links,'print',info='on')
paint_jobs=filter_links(job_links,'paint',info='on')
color_jobs=filter_links(job_links,'color',info='on')
worker_jobs=filter_links(job_links,'worker',info='on')
operator_jobs=filter_links(job_links,'operator',info='on')
print('\n')
computer_jobs=filter_links(job_links,'computer',info='on')
programmer_jobs=filter_links(job_links,'programmer',info='on')
print('\n')
teaching_jobs=filter_links(job_links,'teach',info='on')
mentoring_jobs=filter_links(job_links,'mentor',info='on')
education_jobs=filter_links(job_links,'educat',info='on')
training_jobs=filter_links(job_links,'training',info='on')
instructor_jobs=filter_links(job_links,'instructor',info='on')
print('\n')
law_jobs=filter_links(job_links,'law',info='on')
print('\n')
leisure_jobs=filter_links(job_links,'leisure',info='on')
entertainment_jobs=filter_links(job_links,'entertain',info='on')
print('\n')
marketing_jobs=filter_links(job_links,'market',info='on')
publishing_jobs=filter_links(job_links,'publi',info='on')
information_jobs=filter_links(job_links,'information',info='on')
communication_jobs=filter_links(job_links,'communication',info='on')
print('\n')
designer_jobs=filter_links(job_links,'designer',info='on')
multimedia_jobs=filter_links(job_links,'multimedia',info='on')
print('\n')
art_jobs=filter_links(job_links,'art ',info='on')
film_jobs=filter_links(job_links,'film',info='on')
photo_jobs=filter_links(job_links,'photo',info='on')
artist_jobs=filter_links(job_links,'artist',info='on')
theatre_jobs=filter_links(job_links,'theatre',info='on')
print('\n')
stock_jobs=filter_links(job_links,'stock',info='on')
shopping_jobs=filter_links(job_links,'shop',info='on')
logistic_jobs=filter_links(job_links,'logistic',info='on')

```

```

print('\n')
trade_jobs=filter_links(job_links,'trade',info='on')
sales_jobs=filter_links(job_links,'sales',info='on')
print('\n')
drive_jobs=filter_links(job_links,'drive',info='on')
transport_jobs=filter_links(job_links,'transport',info='on')

```

Dataset category extraction

```

dataset_tag=False          #Enable datase tagging
#Categories to tag
category=[account_jobs,financial_jobs,tax_jobs,
          manager_jobs,director_jobs,
          clerk_jobs,
          engineer_jobs,technician_jobs,
          nurse_jobs,health_jobs,
          mechanic_jobs,maintenance_jobs,
          police_jobs,officer_jobs,security_jobs,guard_jobs,
          worker_jobs,operator_jobs,
          teaching_jobs,education_jobs,training_jobs,instructor_jobs,
          drive_jobs,
          sales_jobs,
          stock_jobs,shopping_jobs,
          print_jobs,dye_jobs,paint_jobs,
          photo_jobs,color_jobs]

folder=['ACCOUNT','FINANCIAL','TAX'
        'MANAGER','DIRECTOR',
        'CLERK',
        'ENGINEER','TECHNICIAN',
        'NURSE','HEALTH',
        'MECHANIC','MAINTENANCE',
        'POLICE','OFICER','SECURITY','GUARD'
        'WORKER','OPERATOR',
        'TEACHING','EDUCATION','TRAINING','INSTRUCTOR',
        'DRIVER',
        'SALES',
        'STOCK','SHOPPING',
        'PRINTER','DYE','PAINTER'
        'PHOTO','COLOR']

if dataset_tag==True:
    #Category extraction
    index=0
    for url_categ in category:
        for url in url_categ:
            title,attributes=get_job_duty(url,'h1',info='off')
            create_job_bulletin(title,attributes,'Data
mining',folder[index])
            index+=1

```

Individual category extraction

```

example_category_jobs=['https://dot-job-
descriptions.careerplanner.com/ACCORDION-TUNER.cfm']
category=[example_category_jobs]
folder=['EXAMPLE CATEGORY']

#Category extraction
index=0

```

```

for url_categ in category:
    for url in url_categ:
        title,attributes=get_job_duty(url,'h1',info='off')
        create_job_bulletin(title,attributes,'Data mining',folder[index])
    index+=1

```

A.4 Características (*features*)

A.4.1 Selección de características sobre el *dataset* completo

A.4.1.1020 *Features_Statistics_N.ipynb*

Test settings

```

threshold_T=20           #Noun chunk minimum occurrences
stats=True              #Calculate statistics

```

Modules import

```

#SYSTEM MODULES
import pickle           #Data saving library

#USER MODULES
#Pre processing functions
from M_text_pre_processing import textPreProcessing
#Syntactic analysis
from M_syntactic_analysis import
chunk_lookup,chunk_lemma,grammar_chunk,items_over_threshold,POS_tagging

```

Loading job bulletins information

```

file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_att.sav'
list_job_att = pickle.load(open(file_path, 'rb'))

```

Chunk detection

```

grammar = r"""
    NP: {<N.*>}
    """
chunk_dict={}
list_chunk_all=[]
stop_words=[]
for sentence in list_job_att[0]:
    sentence=textPreProcessing(sentence)
    POS_tag=POS_tagging(sentence)
    tree_phrase=grammar_chunk(grammar,POS_tag)
    list_chunk=chunk_lookup(tree_phrase,info=False,debug=False)
    for key in list_chunk:
        key_lemma=chunk_lemma(key,'n')           #Get lemma of chunk
        if key_lemma not in stop_words:
            chunk_dict[key_lemma]=0
            list_chunk_all.append(key_lemma)

```



```

for key in list_chunk_all:
    chunk_dict[key]+=1
print(len(chunk_dict),'chunks found
along',len(list_job_att[0]),'documents')
sorted_chunk_dict = sorted(chunk_dict.items(), key=lambda kv: kv[1])
print(sorted_chunk_dict)

```

Statistics & selecting most common chunks

```

#Statistics
if stats==True:
    total_chunk=len(chunk_dict)
    for threshold in range(1,31):
        list_duty_features=items_over_threshold(chunk_dict,threshold)
        print('\nMinimum occurrences:',threshold)
        print('\tNumber of features:',len(list_duty_features))
        print("\tCoverage percentage
{0:.1f}".format(len(list_duty_features)/total_chunk*100.0))
        #print(list_duty_features)

#Selecting most common chunks
list_common_duty_features=items_over_threshold(chunk_dict,threshold_T)
print('\nMost common features:',len(list_common_duty_features))
print(list_common_duty_features)
coverage=round(len(list_common_duty_features)/total_chunk*100.0,1)
#Saving list of selected features
file_path
='list_common_duty_features_N_th'+str(threshold_T)+'_p'+str(coverage)+'.sav
,
pickle.dump(list_common_duty_features, open(file_path, 'wb'))

```

A.4.1.2020_Features_Statistics_V.ipynb

Las rutinas nombradas como *_V* son variantes de las *_N* que utilizan verbos como constituyentes sintácticos en vez de nombres. Solo difieren en la gramática utilizada y las llamadas a las funciones *chunk_lemma(key,'v')* y *get_synonyms(feature,pos='VERB')*. La gramática utilizada es:

```

VP: {<TO> <V.*>}      #TO=to
    {<V.*> <RB> <V.*>} #RB=adverb
    {<V.*> <V.*> <TO>}
    {<V.*> <IN>}       #IN=preposition/subordination conjunction
    {<V.*> <RP>}       #RP=participle
    {<V.*> <VBN>}     #VBN=past participle
    {<V.*> <VBG>}     #VBG=gerund
    {<V.*>+}
    {<V.*>}
    <TO>{              #Pick up only the verb from previous <TO> rules
    <RB> <V.*>{       #Pick up only the verb from previous <RB> <V.*> rules

```

A.4.2 Selección de características individual por categorías

A.4.2.1020_Features_Statistics_INDIVIDUAL_N.ipynb

Test settings

```
threshold_T=20          #Noun chunk minimum occurrences
```

Modules import

```
#SYSTEM MODULES
import pickle          #Data saving library

#USER MODULES
#Pre processing functions
from M_text_pre_processing import textPreProcessing
#Syntactic analysis
from M_syntactic_analysis import
chunk_lookup,chunk_lemma,grammar_chunk,items_over_threshold_limited,POS_tagging
```

Loading job bulletins information

```
file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_att.sav'
list_job_att = pickle.load(open(file_path, 'rb'))
```

Chunk detection

```
grammar = r"""
    NP: {<N.*>}
    """
stop_words=[]

list_features=[]
categories=set(job_names.values())

for category in categories:
    item=0          #Loop all bulletins
    chunk_dict={}
    list_chunk_all=[]
    for bulletin in job_names.keys():
        if job_names[bulletin]==category:
            #Duty attribute
            sentence=list_job_att[item]
            sentence=textPreProcessing(sentence)
            POS_tag=POS_tagging(sentence)
            tree_phrase=grammar_chunk(grammar,POS_tag)
            list_chunk=chunk_lookup(tree_phrase,info=False,debug=False)
            for key in list_chunk:
                key_lemma=chunk_lemma(key,'n')          #Get lemma of chunk
                if key_lemma not in stop_words:
                    chunk_dict[key_lemma]=0
                    list_chunk_all.append(key_lemma)
```

```

        item+=1

    for key in list_chunk_all:
        chunk_dict[key]+=1

    total_chunk=len(chunk_dict)
    #Selecting most common chunks

list_common_duty_features=items_over_threshold_limited(chunk_dict,threshold
_T,60)
    print(category)
    print('Most common features:',len(list_common_duty_features))
    print(list_common_duty_features)
    coverage=round(len(list_common_duty_features)/total_chunk*100.0,1)

    for common_feature in list_common_duty_features:
        if common_feature not in list_features:
            list_features.append(common_feature)
print('Number of features selected:',len(list_features))
print(list_features)
#Saving list of selected features
file_path ='list_features_N_th'+str(threshold_T)+'.sav'
pickle.dump(list_features, open(file_path, 'wb'))

```

A4.3 Extracción de características

A4.3.1 020_Features_Duty_N.ipynb

Test settings

```

threshold_T=20          #Noum chunk minimum ocurrences
features_list='_N_th20' #Feature selection list
synonyms=True          #Use of synonyms when counting the input
words
downsize_features=False

```

Libraries import

```

#SYSTEM MODULES
import time          #Timestamp library
import pickle        #Data saving library
import numpy as np   #Mathematical functions & data handling
import pandas as pd  #Data & statistical analysis

#USER MODULES
#Pre processing functions
from M_text_pre_processing import textPreProcessing
#Syntactic analysis
from M_syntactic_analysis import
chunk_lookup,chunk_lemma,grammar_chunk,items_over_threshold,POS_tagging
from M_syntactic_analysis import get_synonyms

```

Loading job bulletins information

```

file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))

```

```
file_path = 'list_job_att.sav'
list_job_att = pickle.load(open(file_path, 'rb'))
```

Loading most common features

```
file_path = 'list_features'+features_list+'.sav'
list_features = pickle.load(open(file_path, 'rb'))
```

Features data set: duty

```
def init_df_job():
    #Job Name & ID
    df_job=pd.DataFrame(list(zip(job_names.keys(),job_names.values())),
    columns=['Jobs','ID'])
    #Features
    for feature in list_features:

df_job.insert(df_job.shape[1],feature,np.ravel(np.zeros((df_job.shape[0],1)
).astype(int)),allow_duplicates=False)
    #Statistics

df_job.insert(df_job.shape[1],'unseen',np.ravel(np.zeros((df_job.shape[0],1)
).astype(int)),allow_duplicates=False)

df_job.insert(df_job.shape[1],'features',np.ravel(np.zeros((df_job.shape[0]
,1)).astype(int)),allow_duplicates=False)

df_job.insert(df_job.shape[1],'synonyms',np.ravel(np.zeros((df_job.shape[0]
,1)).astype(int)),allow_duplicates=False)
    return df_job
df_job=init_df_job()
print(df_job.shape,len(list_job_att))
display(df_job.head(2))
```

Extracting features for each sample:

1. Check most common features matching
2. If not, check matching with synonyms of most common features
3. If not, increase the unseen column

```
t0 = time.time()

index=0
grammar = r"""
    NP: {<N.*>}
    """
stop_words=[]
for sentence in list_job_att:
    list_chunk_bulletin=[] #Chunks in
actual_bulletin
    sentence=textPreProcessing(sentence) #Remove
quotation mark symbols
    POS_tag=POS_tagging(sentence) #POS
tagging
    tree_phrase=grammar_chunk(grammar,POS_tag) #Chunking
    list_chunk=chunk_lookup(tree_phrase,info=False,debug=False) #List of
chunks
    for key in list_chunk:
```

```

        key_lemma=chunk_lemma(key, 'n') #Get lemma
of chunks
    match=False
    for feature in list_features:
        #Do not check stop words
        if key_lemma in stop_words:
            break
        #First check: most common features
        if key_lemma==feature:
            df_job.loc[index, feature]+=1
            df_job.loc[index, 'features']+=1
            match=True
            break
        #Second check: synonyms of most common features
        if synonyms==True:
            if match==False:
                for feature in list_features:
                    key_lemma=chunk_lemma(key, 'n')
                    if key_lemma in get_synonyms(feature, pos='NOUN'):
                        df_job.loc[index, feature]+=1
                        df_job.loc[index, 'synonyms']+=1
                        match=True
                        break
            #Increase unseen category
            if match==False:
                df_job.loc[index, 'unseen']+=1
        #Bulletin index update
        index+=1

tf=time.time()-t0
print('\nProcessing time:', round(tf, 2))
if downsize_features==True:
    categories=set(job_names.values())

    df_job_reduced=df_job.copy()
    th=60

    print('initial cols', df_job_reduced.shape[1])
    print('\nOver threshold:')
    for col in list(df_job_reduced.columns.values):

        num_categ=0
        if col not in ['Jobs', 'ID', 'unseen', 'features', 'synonyms']:
            num_categ=0
            for category in categories:
                df_pd_categ=df_job_reduced[df_job_reduced['ID'] ==
category]

                df_pd=df_pd_categ[df_pd_categ[col] > 0]
                #print(df_pd.shape[0])
                if df_pd.shape[0] > th:
                    num_categ+=1

    #print('\tFrequency:', df_pd_categ.shape[0], 'Info:', category, col)
    if num_categ > 1:
        del df_job_reduced[col]
        print('\t\tDELTED', col)

    print('\nfinal cols', df_job_reduced.shape[1])

```

Statistics

```

print('Number of dataset features:',len(list_features))
print('Extraction average')
if downsize_features==True:
    av_features=df_job_reduced['features'].mean()
    av_synonyms=df_job_reduced['synonyms'].mean()
    av_unseen=df_job_reduced['unseen'].mean()
else:
    av_features=df_job['features'].mean()
    av_synonyms=df_job['synonyms'].mean()
    av_unseen=df_job['unseen'].mean()
print('\tFeatures:',round(av_features,1))
print('\tSynonyms:',round(av_synonyms,1))
print('\tUnseen:',round(av_unseen,1))
print('\t\tExtraction:',round(av_features+av_synonyms,1))
print('\t\t(features+synonyms)')
print('\t\tUsage
percentaje:',round((av_features+av_synonyms)/len(list_features)*100,1),'%')
print('\t\t(extraction/dataset features)')
#Save into excel file
if synonyms==True:
    syn_option='_synonym'
else:
    syn_option=''
if downsize_features==True:
    num_features=df_job_reduced.shape[1]-5

name='Duty_INDIVIDAL_features_N_th'+str(threshold_T)+syn_option+'_DOWNSIZE_
'+str(num_features)+'_th'+str(th)+'.xlsx'
    writer = pd.ExcelWriter(name)
    df_job_reduced.to_excel(writer,'Duty_features',index=False)
else:
    name='Duty_INDIVIDAL_features_N_th'+str(threshold_T)+syn_option+'.xlsx'
    writer = pd.ExcelWriter(name)
    df_job.to_excel(writer,'Duty_features',index=False)
writer.save()

```

A.5 Agrupamiento (*clustering*)

A.5.1 030_Clustering_by_N.ipynb

Test settings

```

data_ones=False #Bernoulli data distribution
(true/false)
data_scaled=True #Scale dataset columns between
max/min
save_tagged_features=False #Save tagged data frame into
exel file
category_tagged=False
dataframe_src='Duty_features_N_th20_synonym' #Data frame features file

```

Libraries import

```

#General libraries
import re #regex
import pickle #Data saving library
import numpy as np #Mathematical functions & data handling

```

```

from numpy import array
import pandas as pd          #Data & statistical analysis
import matplotlib.pyplot as plt  #Visualization library

#Machine Learning libraries
from sklearn import metrics
from sklearn.cluster import KMeans
from sklearn.metrics import pairwise_distances
from sklearn.preprocessing import MaxAbsScaler
from sklearn.metrics import davies_bouldin_score

```

Loading job bulletins information

```

file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_att.sav'
list_job_att = pickle.load(open(file_path, 'rb'))

```

Loading jobs groups

```

file_path = 'dict_job_group.sav'
dict_job_group = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_group.sav'
list_job_group = pickle.load(open(file_path, 'rb'))
file_path = 'dict_job_group_tagged.sav'
dict_job_group_tagged = pickle.load(open(file_path, 'rb'))

```

Loading data set

```

df_data=pd.read_excel(dataframe_src+'.xlsx',index_col=None)
df_data_src=pd.read_excel(dataframe_src+'.xlsx',index_col=None)
df_category_tagged=pd.read_excel('Job_group_tagged.xlsx',index_col=None)
print('File size:',df_data.shape)
display(df_data.head(2))
#Dataset characterization
print(' ')
print('Statistical properties')
display(df_data.describe())
#Correlación
print(' ')
print('Attributes correlations')
display(df_data.corr())

```

Expanding data with learnend job category

```

if category_tagged==False:

df_data.insert(df_data.shape[1],'category','OTHER',allow_duplicates=False)

df_data_src.insert(df_data_src.shape[1],'category','OTHER',allow_duplicates
=False)
else:

df_data.insert(df_data.shape[1],'category',df_category_tagged.loc[:,'CATEGO
RY'],allow_duplicates=False)

```

```

df_data_src.insert(df_data_src.shape[1], 'category', df_category_tagged.loc[:,
'CATEGORY'], allow_duplicates=False)
df_data.insert(df_data.shape[1], 'category_ID', np.ravel(np.zeros((df_data.sh
ape[0], 1)).astype(int)), allow_duplicates=False)
df_data_src.insert(df_data_src.shape[1], 'category_ID', np.ravel(np.zeros((df
_data_src.shape[0], 1)).astype(int)), allow_duplicates=False)
if category_tagged==False:
    for row in range(df_data.shape[0]):
        value_count=0 #If the
word is repeated chose the most common one
        job_name_split=re.findall(r'\w+', df_data.loc[row, 'Jobs'])
        for word in job_name_split:
            for key in dict_job_group.keys():
                if word==key and dict_job_group[key]>value_count:
                    value_count=dict_job_group[key]
#Update iteration max value
                    df_data.loc[row, 'category']=key
#Update category
                    df_data.loc[row, 'category_ID']=dict_job_group[key]
#Update category
else:
    for row in range(df_data.shape[0]):

df_data.loc[row, 'category_ID']=dict_job_group_tagged[df_data.loc[row, 'categ
ory']]
print('File size:', df_data.shape)
display(df_data.head(5))
#Save into excel file
if save_tagged_features==True:
    name='Duty_tagged_features_N.xlsx'
    writer = pd.ExcelWriter(name)
    df_data.to_excel(writer, 'Duty_tagged_features_N', index=False)
    writer.save()

```

Data distribution

```

#Attributes histogram visualization
fig, (ax1, ax2, ax3) = plt.subplots(nrows=1, ncols=3, figsize=(16, 10))

X_hist=df_data['unseen'].values
ax1.hist(X_hist, bins=30, alpha=1, edgecolor = 'black', linewidth=1)
ax1.set_title('unseen')
ax1.grid(True)
X_hist=df_data['features'].values
ax2.hist(X_hist, bins=30, alpha=1, edgecolor = 'black', linewidth=1)
ax2.set_title('features')
ax2.grid(True)
X_hist=df_data['synonyms'].values
ax3.hist(X_hist, bins=30, alpha=1, edgecolor = 'black', linewidth=1)
ax3.set_title('synonyms')
ax3.grid(True)

plt.show()
plt.clf()

fig, bx1 = plt.subplots(nrows=1, ncols=1, figsize=(6, 6))

X_hist=df_data['category_ID'].values
bx1.hist(X_hist, bins=len(list_job_group), alpha=1, edgecolor = 'black',
linewidth=1)

```



```
bx1.set_title('category_ID')
bx1.grid(True)
```

```
plt.show()
plt.clf()
```

Extracting attributes for training the cluster algorithm

```
y_ID=df_data.values[:,1]
y_job=df_data.values[:,0]
y_category_ID=df_data.values[:,df_data.shape[1]-1] #Subtracts 1 because is
starting at zero
x=df_data.values[:,2:df_data.shape[1]-5]
#Unseen//features//synonyms//category//category_ID (start: 2)
```

Pre-processing option

```
#Bernoulli data distribution
if data_ones==True:
    for row in range(x.shape[0]):
        for col in range(x.shape[1]):
            if x[row,col]>0:
                x[row,col]=1
#Scaling variables
if data_scaled==True:
    scaler = MaxAbsScaler()
    x=scaler.fit_transform(x)
```

Data clustering

```
def get_index_groupby(df, keyword, n_groups, debug='off'):

    df_i=df.groupby([keyword, 'clusters']).count()
    lst_i=df_i.index.tolist()
    c_gx_list=[]
    if(debug=='on'):
        display(df_i)
        print(lst_i)
    #Loop over component
    for key in range(n_groups+1):
        c_gx_set=set()
        for index in range(len(lst_i)):
            if lst_i[index][0]==key:
                c_gx_set.add(lst_i[index][1])
        c_gx_list.append(list(c_gx_set)) #Cast to list
    if(debug=='on'):
        print('c_gx_list', len(c_gx_list))
        print(c_gx_list)

    return c_gx_list

def equalize_size(i_in, v_in, size, debug='off'):
    #i_in: index of input cluster found for this group, v_in: value for
    each cluster found
    #size: total number of cluster for equalizing
    if len(i_in)>0:
        iter_in=0
        key_in=i_in[0]
        key_out=0
```

```

i_out=[]
v_out=[]
for key in range(size):
    if key_in==key_out:
        if iter_in<len(i_in):
            i_out.append(i_in[iter_in])
            v_out.append(v_in[iter_in])
            if (iter_in+1)<len(i_in):
                iter_in+=1
                key_in=i_in[iter_in]
            key_out+=1
        else:
            i_out.append(0)
            v_out.append(0)
            key_out+=1
    else:
        i_out=[0 for i in range(size)]
        v_out=[0 for i in range(size)]
if debug=='on':
    print('Input')
    print(i_in,v_in)
    print('Output')
    print(np.array(i_out),np.array(v_out))
return np.array(i_out),np.array(v_out)
def
kmeans_clustering(df,df_src,n_clusters,X,ref_group,debug='off',info='off'):

    #Training the model
    kmeans=KMeans(n_clusters=n_clusters,init='k-
means+',random_state=0).fit(X)
    labels=kmeans.labels_
    #Evaluating the model
    m_silhouette=metrics.silhouette_score(X,labels,metric='sqeuclidean')
    m_calinski=metrics.calinski_harabaz_score(X,labels)

    #Adding lables to dataframe
    df['clusters']=labels
    df_src['clusters']=labels
    if debug=='on':
        print('clusters')
        print(labels)
    #Number of categories
    n=max(labels)+1

    #Regions for customer of each cluster
    df_r=df.groupby(['category_ID','clusters']).count()
    n_groups=[i for i in range(1,len(ref_group)+1)]
    r_c_gx=get_index_groupby(df,'category_ID',max(n_groups),debug=debug)
#Cluster index for each group
#Display info
if debug=='on':
    print('n_groups',len(n_groups))

    #Values of each cluster within every region(group)
    g_s,g_e=0,len(r_c_gx[0])
#Group start index
    r_v_groups=[]
#List of group fragments
    r_v_groups.append(df_r.values[g_s:g_e,0])
    for index in range(0,len(r_c_gx)-1):
        g_s+=len(r_c_gx[index])

```

```

        g_e=g_s+len(r_c_gx[index+1]) #Group end index
        r_v_groups.append(df_r.values[g_s:g_e,0])
    r_v=np.array([item for item in r_v_groups])

#Barplot aggregate visualization
fig,ax1=plt.subplots()
#Regions for customer of each cluster
list_r_vx=[]
for item in range(len(r_v)):
    cx,rx=equalize_size(r_c_gx[item],r_v[item],n,debug=debug)
    list_r_vx.append(rx)
r_vx=array(list_r_vx)
#Display info
if debug=='on':
    print('size r_vx',len(r_vx))
    print('Value clusters (columns) by groups (rows)\n',r_vx)
    #Region Bar plot
#Cluster index
ci = np.array([i for i in range(n)])
ax1.bar(ci,r_vx[0], edgecolor='white',label=ref_group[0])
#Group 0
    bottom_gp=r_vx[0]
    ax1.bar(ci,r_vx[1],
bottom=bottom_gp,edgecolor='white',label=ref_group[1]) #Group
1
    for index in range(2,len(ref_group)):
        bottom_gp+=r_vx[index-1]
        ax1.bar(ci,r_vx[index],
bottom=bottom_gp,edgecolor='white',label=ref_group[index]) #Group x>1
    #Layout
    ax1.set_xlabel('Category')
    ax1.legend(loc='upper left',bbox_to_anchor=(1,1),ncol=1)
    ax1.set_title('K-means with '+str(n_clusters)+' clusters'+ ' &
'+str(len(ref_group))+' categories')
    plt.show()

#Display metrics
print('\nMetrics')
print('Silhouette Coefficient:',m_silhouette)
print('Calinski-Harabaz Index:',m_calinski)

#Statistics
for key in range(n_clusters):
    count=0
    cluster_elements=[]
    dict_cluster_categ={}
    for row in range(df.shape[0]) :
        if df.loc[row,'clusters']==key:
            count+=1
            if df.loc[row,'category'] in dict_cluster_categ:
                dict_cluster_categ[df.loc[row,'category']] +=1
            else:
                dict_cluster_categ[df.loc[row,'category']] =1
            cluster_elements.append(df.loc[row,'Jobs'])
    print('\nItems on cluster',key,':',count)
    sorted_dict_cluster_categ = sorted(dict_cluster_categ.items(),
key=lambda kv: kv[0])
    print(sorted_dict_cluster_categ)
    if info=='on':
        print('\n',cluster_elements)
if category_tagged==False:

```

```

kmeans_clustering(df_data,df_data_src,10,x,list_job_group,debug='off',info=
'off')

kmeans_clustering(df_data,df_data_src,15,x,list_job_group,debug='off',info=
'off')

kmeans_clustering(df_data,df_data_src,20,x,list_job_group,debug='off',info=
'off')
else:

kmeans_clustering(df_data,df_data_src,15,x,list(dict_job_group_tagged.keys(
)),debug='off',info='off')

kmeans_clustering(df_data,df_data_src,20,x,list(dict_job_group_tagged.keys(
)),debug='off',info='off')

kmeans_clustering(df_data,df_data_src,25,x,list(dict_job_group_tagged.keys(
)),debug='off',info='on')

```

A.6 Clasificadores

A6.1 040_Classifier_by_NB.ipynb

Test settings

```

dataframe_src='Duty_INDIVIDAL_features_N_th20_synonym' #Data frame
features file
data_ones=False #Bernoulli data distribution
(true/false)
data_scaled=False #Scale dataset columns
between max/min
data_scaled_tf=False #Scale dataset rows between
max/min
data_scaled_tfidf=True #Modify features weights by
tfidf
#Default dataset
if data_ones==False and data_scaled==False and data_scaled_tf==False and
data_scaled_tfidf==False:
    data_default=True
else:
    data_default=False

```

Libraries import

```

#General Libraries
import time #Timestamp library
import math #Math library
import pickle #Data saving library
import numpy as np #Mathematical functions & data handling
from numpy import array
import pandas as pd #Data & statistical analysis
from sklearn.preprocessing import MaxAbsScaler
#Machine Learning Libraries
from sklearn.model_selection import cross_val_score
from sklearn.naive_bayes import
GaussianNB,MultinomialNB,ComplementNB,BernoulliNB

```

Loading dataset group information

```
file_path = 'dict_job_group_tagged.sav'
dict_job_group_tagged = pickle.load(open(file_path, 'rb'))
```

Loading features information

```
df_data=pd.read_excel(dataframe_src+'.xlsx',index_col=None)
df_category_tagged=pd.read_excel('Job_group_tagged.xlsx',index_col=None)
```

Building tagged dataset

```
df_data.insert(df_data.shape[1], 'category', df_category_tagged.loc[:, 'CATEGORY'], allow_duplicates=False)
df_data.insert(df_data.shape[1], 'category_ID', np.ravel(np.zeros((df_data.shape[0], 1)).astype(int)), allow_duplicates=False)
for row in range(df_data.shape[0]):

df_data.loc[row, 'category_ID']=dict_job_group_tagged[df_data.loc[row, 'category']]
print('File size:', df_data.shape)
display(df_data.head(3))
```

Extracting attributes for training the cluster algorithm

```
num_categ=df_data.values[:, df_data.shape[1]-1].max()+1
Y=df_data.values[:, df_data.shape[1]-2]
X=df_data.values[:, 2:df_data.shape[1]-5]
#Unseen//features//synonyms//category//category_ID
print('Number of categories:', num_categ)
```

Splitting sample data set into training(80%) & test (20%)

```
#Output vector
y=Y
#Bernoulli data distribution
if data_ones==True:
    x=X.copy()
    for row in range(x.shape[0]):
        for col in range(x.shape[1]):
            if x[row,col]>0:
                x[row,col]=1
#Scaling variables (columns)
if data_scaled==True:
    scaler = MaxAbsScaler()
    x=scaler.fit_transform(X)
#Scaling variables (rows) -> tf
if data_scaled_tf==True:
    row_max = X.max(axis=1)
    x = X / row_max[:, np.newaxis]
#tfidf factor
if data_scaled_tfidf==True:
    #Tf by raw term count
    tf=X.copy()
    #Idf calculation
    row_count = np.count_nonzero(tf, axis=0) #Count of documents where
the term occurs (rows.value>0)
    idf=np.zeros(tf.shape[1]) #Vector with features size
```

```

    for index in range (len(idf)):
        if row_count[index]!=0:
            #Logarithm of (number of documents/number of documents in which
k term occurs)
            idf[index]=math.log(X.shape[0]/row_count[index],10) #Base 10
logarithm
        else:
            idf[index]=0 #Avoid division by zero
    #Tf-idf calculation
    x= tf * idf
#Default dataset
if data_default==True:
    x=X.copy()

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.20,shuffle=True,random_state=80)

```

Splitting training data set into training(80%) & validation (20%)

```

from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val =
train_test_split(x_train,y_train,test_size=0.2,shuffle=True,random_state=80
)

```

Model training

```

def
NB_classifier(x_train,y_train,x_test,y_test,x_val,y_val,NB_type='BernoulliNB
',info=False):
    #MODEL
    if NB_type=='GaussianNB':
        model = GaussianNB()
    if NB_type=='MultinomialNB':
        model = MultinomialNB()
    if NB_type=='ComplementNB':
        model = ComplementNB()
    if NB_type=='BernoulliNB':
        model = BernoulliNB()
    #TRAINING
    t0 = time.time()
    model.fit(x_train, np.ravel(y_train))
    tf=time.time()-t0
    print('\nTRAINING TIME:',round(tf, 4))
    #EVALUATING
    score_val=model.score(x_val,np.ravel(y_val))
    score_test=model.score(x_test,np.ravel(y_test))
    score_cv=cross_val_score(model,x_test,np.ravel(y_test),cv=5)
    if info==True:
        print('\nNB type:',NB_type.upper())
        print('NB accuracy on VALIDATION',round(score_val,4))
        print('NB accuracy on TEST',round(score_test,4))
    for key in range(len(score_cv)):
        score_cv[key]=round(score_cv[key],4)
    print('NB accuracy on cross-validation',score_cv)
    #PREDICTION
    y_out = model.predict(x_test)

    return model,y_out,score_test,score_cv

```



```

from keras import optimizers
from keras.models import Sequential
from keras import utils as np_utils
from keras.callbacks import EarlyStopping
from keras.layers import BatchNormalization
from keras.regularizers import l1, l2, l1_l2
from keras.layers import Dense, Activation, Flatten, Dropout

```

Visualization funtions

```

def plot_acc(history, title="Model Accuracy"):
    """Imprime una gráfica mostrando la accuracy por epoch obtenida en un
    entrenamiento"""
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

def plot_loss(history, title="Model Loss"):
    """Imprime una gráfica mostrando la pérdida por epoch obtenida en un
    entrenamiento"""
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()

```

Loading dataset group information

```

file_path = 'dict_job_group_tagged.sav'
dict_job_group_tagged = pickle.load(open(file_path, 'rb'))

```

Loading features information

```

df_data=pd.read_excel(dataframe_src+'.xlsx',index_col=None)
df_category_tagged=pd.read_excel('Job_group_tagged.xlsx',index_col=None)

```

Building tagged dataset

```

df_data.insert(df_data.shape[1],'category',df_category_tagged.loc[:, 'CATEGO
RY'],allow_duplicates=False)
df_data.insert(df_data.shape[1],'category_ID',np.ravel(np.zeros((df_data.sh
ape[0],1)).astype(int)),allow_duplicates=False)
for row in range(df_data.shape[0]):

df_data.loc[row,'category_ID']=dict_job_group_tagged[df_data.loc[row,'categ
ory']]
print('File size:',df_data.shape)
display(df_data.head(3))

```

Extracting attributes for training the cluster algortihm

```

Y=df_data.values[:,df_data.shape[1]-1]

```



```
X=df_data.values[:,2:df_data.shape[1]-5]
#Unseen//features//synonyms//category//category_ID
num_categ=Y.max()+1
print('Number of categories:',num_categ)
```

Splitting sample data set into training(80%) & test (20%)

```
#One-hot-encoding for output vector
y=keras.utils.to_categorical(Y)
#Scaling variables (columns)
if data_scaled==True:
    scaler = MaxAbsScaler()
    x=scaler.fit_transform(X)
#Scaling variables (rows) -> tf
if data_scaled_tf==True:
    row_max = X.max(axis=1)
    x = X / row_max[:, np.newaxis]
#tfidf factor
if data_scaled_tfidf==True:
    #Tf by raw term count
    tf=X.copy()
    #Idf calculation
    row_count = np.count_nonzero(tf, axis=0) #Count of documents where
the term occurs (rows.value>0)
    idf=np.zeros(tf.shape[1]) #Vector with features size
    for index in range (len(idf)):
        if row_count[index]!=0:
            #Logarithm of (number of documents/number of documents in which
k term occurs)
            idf[index]=math.log(X.shape[0]/row_count[index],10) #Base 10
logarithm
        else:
            idf[index]=0 #Avoid division by zero
    #Tf-idf calculation
    tfidf= tf * idf
    #Scaling in range [0,1]
    scaler = MaxAbsScaler()
    x=scaler.fit_transform(tfidf)

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x,y,test_size=0.20,shuffle=True,random_state=80)
```

Splitting training data set into training(80%) & validation (20%)

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val =
train_test_split(x_train,y_train,test_size=0.2,shuffle=True,random_state=80
)
```

Model built-up

```
#MODEL DEFINITION
def
nn_model(input_sh,layer_u,activation_u,activation_out,kernel_i,bias_i,categ
_num,p_l1,p_l2,dropout=False,BN=False):
    #input_sh: tuple with input shape for input layer
    #layer_u: number of units for each hidden layer
    #categ_num: number of categories
```

```

#Linear stack of layers model
model = Sequential()
#HIDDEN LAYERS
for key in range(len(layer_u)):
    name_u='Hidden_layer_'+str(key)
    if BN==False:

model.add(Dense(layer_u[key], activation=activation_u, kernel_initializer=kernel_i, bias_initializer=bias_i, activity_regularizer=l1_l2(l1=p_l1, l2=p_l2), name=name_u))
    if dropout==True:
        model.add(Dropout(0.125))
    else:

model.add(Dense(layer_u[key], kernel_initializer=kernel_i, use_bias=False))
    model.add(BatchNormalization())
    model.add(Activation(activation_u))
#OUTPUT LAYER

model.add(Dense(categ_num, activation=activation_out, kernel_initializer=kernel_i, bias_initializer=bias_i, name="Output_layer"))

return model

```

Optimizer settings

```

#OPTIMIZER SETTINGS
def nn_model_compile(model, optimizer):
    if optimizer=='SGD':
        nn_opt=optimizers.SGD()
    if optimizer=='SGD_momentum':

nn_opt=optimizers.SGD(lr=0.02, momentum=0.88, decay=0.001, nesterov=False)
    if optimizer=='Adagrad':
        nn_opt=optimizers.Adagrad()
    if optimizer=='RMSprop':
        nn_opt=optimizers.RMSprop()
    if optimizer=='Adam':
        nn_opt=optimizers.Adam()
#MODEL COMPILATION

model.compile(loss=keras.losses.categorical_crossentropy, optimizer=nn_opt, metrics=['categorical_accuracy', 'acc'])

```

Model training

```

#TRAINING SETTINGS
def
nn_model_fit(model, x_train, y_train, x_val, y_val, x_test, y_test, b_size, epo, info):
    #Early stopping
    callbacks
=[EarlyStopping(monitor='val_acc', min_delta=0.001, patience=5, verbose=0, mode='auto', baseline=None, restore_best_weights=False)]

    t0 = time.time()

model_fit=model.fit(x=x_train, y=y_train, batch_size=b_size, epochs=epo, verbose=info, validation_data=(x_val, y_val), shuffle=True, callbacks=callbacks)

```

```

tf=time.time()-t0
print('\nTRAINING TIME:',round(tf, 4))
#Loss & accuracy info
model_metrics=model.evaluate(x=x_val,y=y_val,batch_size=b_size)
print('\nVALIDATION dataset')

print('Accuracy:',round(model_metrics[1],4),'Loss:',round(model_metrics[0],
4))
model_metrics=model.evaluate(x=x_test,y=y_test,batch_size=b_size)
print('\nTEST dataset')

print('Accuracy:',round(model_metrics[1],4),'Loss:',round(model_metrics[0],
4))

plot_acc(model_fit, title="Model Accuracy")
plot_loss(model_fit, title="Model Loss")

return model_fit

```

TESTING

```

#Model built-up
#Model definition
rx,cx=x_train.shape
l_in,h_01,h_02,h_03,h_04=cx,int(cx*1.2),int(cx*0.6),int(cx*0.4),max(int(cx*
0.2),int(num_categ*1.4))
print('l_in',l_in,'h_01',h_01,'h_02',h_02,'h_03',h_03,'h_04',h_04,'l_out',n
um_categ)

model=nn_model((0,0),[l_in,h_01,h_02,h_03,h_04],'relu','softmax','he_normal
','Zeros',num_categ,0.0,0.0,dropout=True,BN=False)
#Compiling de model
nn_model_compile(model,'Adam')
#Training the model
b_size,epochs=96,100
model_fit=nn_model_fit(model,x_train,y_train,x_val,y_val,x_test,y_test,b_si
ze,epochs,0)
model.summary()

```

Generate predictions

```

y_out = model.predict(x_test)
#Other suitable metrics: Confussion Matrix
y_out_pred=np.argmax(y_out, axis=1)
nn_report=classification_report(np.argmax(y_test, axis=1),y_out_pred)
print(nn_report)

```

A6.3 050_Classifier_by_w2v.ipynb

Test settings

```

len_text=160 #128,160,256
features_list='_N_th20' #Feature selection list
features_by_freq=False #Use most common features as vocabulary

```

Libraries import

```

#General Libraries

```

```

import re                                     #regex
import math
import time
import pickle                                 #Data saving library
import numpy as np
#Workaround Keras
old = np.load
np.load = lambda *a,**k: old(*a,allow_pickle=True,**k)
#End Workaround
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
#NN Libraries
import keras
import tensorflow as tf
from keras import optimizers
from keras import regularizers
from keras import utils as np_utils
from keras.callbacks import EarlyStopping
from keras.regularizers import l1, l2, l1_l2
from keras.models import Sequential
from keras.layers import BatchNormalization
from keras.layers import Dense, Activation, Flatten, Dropout
from keras.layers import Conv2D, MaxPooling2D
from keras.preprocessing.image import ImageDataGenerator
#Pre processing functions
from M_text_pre_processing import textPreProcessing

```

Visualizatin functions

```

def plot_acc(history, title="Model Accuracy"):
    """Imprime una gráfica mostrando la accuracy por epoch obtenida en un
    entrenamiento"""
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title(title)
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper left')
    plt.show()

def plot_loss(history, title="Model Loss"):
    """Imprime una gráfica mostrando la pérdida por epoch obtenida en un
    entrenamiento"""
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title(title)
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Val'], loc='upper right')
    plt.show()

```

Loading job bulletins information

```

file_path = 'job_filenames.sav'
job_filenames = pickle.load(open(file_path, 'rb'))
file_path = 'job_names.sav'
job_names = pickle.load(open(file_path, 'rb'))
file_path = 'list_job_att.sav'

```

```
list_job_att = pickle.load(open(file_path, 'rb'))
```

Loading most common features

```
file_path = 'list_features'+features_list+'.sav'
list_features = pickle.load(open(file_path, 'rb'))
```

Loading dataset group information

```
file_path = 'dict_job_group_tagged.sav'
dict_job_group_tagged = pickle.load(open(file_path, 'rb'))
list_job_group = list(dict_job_group_tagged.keys())
```

Building tagged dataset

```
df_data=pd.read_excel('Job_group_tagged.xlsx',index_col=None)
df_data.insert(df_data.shape[1],'CATEGORY_ID',np.ravel(np.zeros((df_data.shape[0],1)).astype(int)),allow_duplicates=False)
for row in range(df_data.shape[0]):

df_data.loc[row,'CATEGORY_ID']=dict_job_group_tagged[df_data.loc[row,'CATEGORY_ID']]
print('File size:',df_data.shape)
display(df_data.head(3))
```

Total number of words in the samples

```
vocabulary={}
if features_by_freq==False:
    for sentence in list_job_att:
        sentence=textPreProcessing(sentence)
        sentence_split=re.findall(r'\w+',sentence)
        for word in sentence_split:
            if word not in vocabulary:
                vocabulary[word]=1
            else:
                vocabulary[word]+=1
else:
    for word in list_features:
        if word not in vocabulary:
            vocabulary[word]=1
        else:
            vocabulary[word]+=1
print('Vocabulary:',len(vocabulary))
#print(vocabulary)
```

Word dictionary

```
word_index={}
word_index['PAD']=0
word_index['START']=1
word_index['UNK']=2
word_index['UNUSED']=3
index=4
for sentence in list_job_att:
    if features_by_freq==False:
        sentence=textPreProcessing(sentence)
        sentence_split=re.findall(r'\w+',sentence)
```

```

        for word in sentence_split:
            if word not in word_index:
                word_index[word]=index
                index+=1
    else:
        for word in list_features:
            if word not in word_index:
                word_index[word]=index
                index+=1

reverse_word_index=dict([(value,key) for (key,value) in
word_index.items()])

def decode_review(text):
    return ' '.join([reverse_word_index.get(i,'?') for i in text])

vocab_size=len(word_index)
print('Vocabulary including general tokens:',vocab_size)
#print(word_index)
#print(reverse_word_index)

```

Coding text information

```

list_join_att_coded=[]
for sentence in list_job_att:
    list_sentence=[]
    sentence=textPreProcessing(sentence)
    sentence_split=re.findall(r'\w+',sentence)
    for word in sentence_split:
        if word in word_index:
            list_sentence.append(word_index[word])
        else:
            list_sentence.append(2)
    list_join_att_coded.append(list_sentence)
print(len(list_join_att_coded))
print(list_join_att_coded[0])
#Max lenght lookup
def max_len(data01):
    max_len=0
    index=0
    for key in range(len(data01)):
        if len(data01[key])>max_len:
            max_len=len(data01[key])
            index=key

    return max_len,index

#Lenght over threshold lookup
def over_thr(data01,threshold,info=False):
    num_over=0
    per_over=0
    for key in range(len(data01)):
        if len(data01[key])>threshold:
            num_over+=1

    per_over=round(num_over/len(data01)*100,2)

    if info==True:
        print('Instances
over',threshold,'words:',num_over,[' ',per_over,'%'])

```

```

    return num_over, round(per_over, 2)

#Text length analysis
larger_len, larger_i = max_len(list_join_att_coded)
print('Larger text length found:', larger_len)
over, per = over_thr(list_join_att_coded, 64, True)
over, per = over_thr(list_join_att_coded, 128, True)
over, per = over_thr(list_join_att_coded, 160, True)
over, per = over_thr(list_join_att_coded, 192, True)
over, per = over_thr(list_join_att_coded, 256, True)
over, per = over_thr(list_join_att_coded, 320, True)
over, per = over_thr(list_join_att_coded, 384, True)

```

Number of jobs categories

```

num_categ = len(dict_job_group_tagged)
print('Number of categories:', num_categ)

```

Padding addition or sentence shorten to have all inputs with the same length

```

att_coded_PAD = keras.preprocessing.sequence.pad_sequences(list_join_att_coded, value=word_index['PAD'], padding='post', maxlen=len_text)
print(len(att_coded_PAD))

```

After the conversion, we can notice that short strings are filled in with zeros, which is the key number for the 'PAD' symbol.

```

print('Text-coded sample containing', len(att_coded_PAD[5]), 'words')
print(att_coded_PAD[5])
print('Decoded text')
decode_review(att_coded_PAD[5])

```

Splitting sample data set into training(80%) & test (20%)

```

Y = df_data.values[:, df_data.shape[1]-1]
#One-hot-encoding for output vector
y = keras.utils.to_categorical(Y)
x = att_coded_PAD

print(y[300])
print(x[300])
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(x, y, test_size=0.2, shuffle=True, random_state=80)

```

Splitting training data set into training(80%) & validation (20%)

```

from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val =
train_test_split(x_train, y_train, test_size=0.2, shuffle=True, random_state=80)

```

For each word of the vocabulary, the model learns an embedding (word vector) of the defined length. Then, an **embedding (words) average** is done **for each sample**. By this, the algorithm gets a **weighing of the meaning of the text** through the meaning of the words within it.

```
#NN MODEL
def nn_model_built(vocab_size,embedding_size,num_categ,len_text):
    #NN model
    model = Sequential()
    #NN Layers

    model.add(keras.layers.Embedding(vocab_size,embedding_size,embeddings_initializer='glorot_normal',input_length=len_text)) #Learning embeddings while training
    model.add(keras.layers.GlobalAveragePooling1D())
    model.add(Dropout(0.15))

    model.add(keras.layers.Dense(embedding_size,activation='relu',kernel_initializer='he_uniform',bias_initializer='Zeros'))
    model.add(Dense(num_categ,activation='softmax'))

    return model
```

Parameters and training settings

```
#OPTIMIZER SETTINGS
def nn_model_compile(model,optimizer):
    if optimizer=='SGD':
        nn_opt=optimizers.SGD()
    if optimizer=='SGD_momentum':

nn_opt=optimizers.SGD(lr=0.02,momentum=0.88,decay=0.001,nesterov=False)
    if optimizer=='Adagrad':
        nn_opt=optimizers.Adagrad()
    if optimizer=='RMSprop':
        nn_opt=optimizers.RMSprop()
    if optimizer=='Adam':
        nn_opt=optimizers.Adam()
    #MODEL COMPILATION

model.compile(loss=keras.losses.categorical_crossentropy,optimizer=nn_opt,metrics=['categorical_accuracy','acc'])
```

TRAINING

```
#TRAINING SETTINGS
def
nn_model_fit(model,x_train,y_train,x_val,y_val,x_test,y_test,b_size,epo,info):

    t0 = time.time()
    model_fit=model.fit(x=x_train,y=y_train,batch_size=b_size,epochs=epo,
verbose=info,validation_data=(x_val,y_val),shuffle=True)

    tf=time.time()-t0
    print('\nTRAINING TIME:',round(tf, 4))
    #Loss & accuracy info
    model_metrics=model.evaluate(x=x_val,y=y_val,batch_size=b_size)
    print('\nVALIDATION dataset')

print('Accuracy:',round(model_metrics[1],4),'Loss:',round(model_metrics[0],4))
    model_metrics=model.evaluate(x=x_test,y=y_test,batch_size=b_size)
```



```

    print('\nTEST dataset')

print('Accuracy:', round(model_metrics[1], 4), 'Loss:', round(model_metrics[0],
4))

    plot_acc(model_fit, title="Model Accuracy")
    plot_loss(model_fit, title="Model Loss")

    return model_fit

```

Testing modifying the embedding size.

```

#Model built-up
embedding_size=48
model=nn_model_built(vocab_size, embedding_size, num_categ, len_text)
#Compiling de model
nn_model_compile(model, 'Adam')
#Training the model
b_size, epochs=128, 60
model_fit=nn_model_fit(model, x_train, y_train, x_val, y_val, x_test, y_test, b_si
ze, epochs, 0)

#Model built-up
embedding_size=64
model=nn_model_built(vocab_size, embedding_size, num_categ, len_text)
#Compiling de model
nn_model_compile(model, 'Adam')
#Training the model
b_size, epochs=128, 60
model_fit=nn_model_fit(model, x_train, y_train, x_val, y_val, x_test, y_test, b_si
ze, epochs, 0)

```

Finally we analyze some usual metrics for classification problems

```

#Confussion Matrix
nn_pred=model.predict(x_test, batch_size=b_size)
nn_predicted=np.argmax(nn_pred, axis=1)
nn_report=classification_report(np.argmax(y_test, axis=1), nn_predicted)
print(nn_report)

```

Apéndice B

Este apéndice contiene el código utilizado por la herramienta analítica que trabaja conjuntamente con el agente conversacional durante la fase de servicio. La estructura del proyecto puede consultarse en la [figura 21](#).

B.1 Configuración Pycharm

Es importante que las variables de entorno apunten al módulo donde se gestionan las peticiones del *webhook* (*app.py*). Anotar también que el entorno *Pycharm* debe invocar el ejecutable de la librería *flask* para que el servicio se active correctamente.

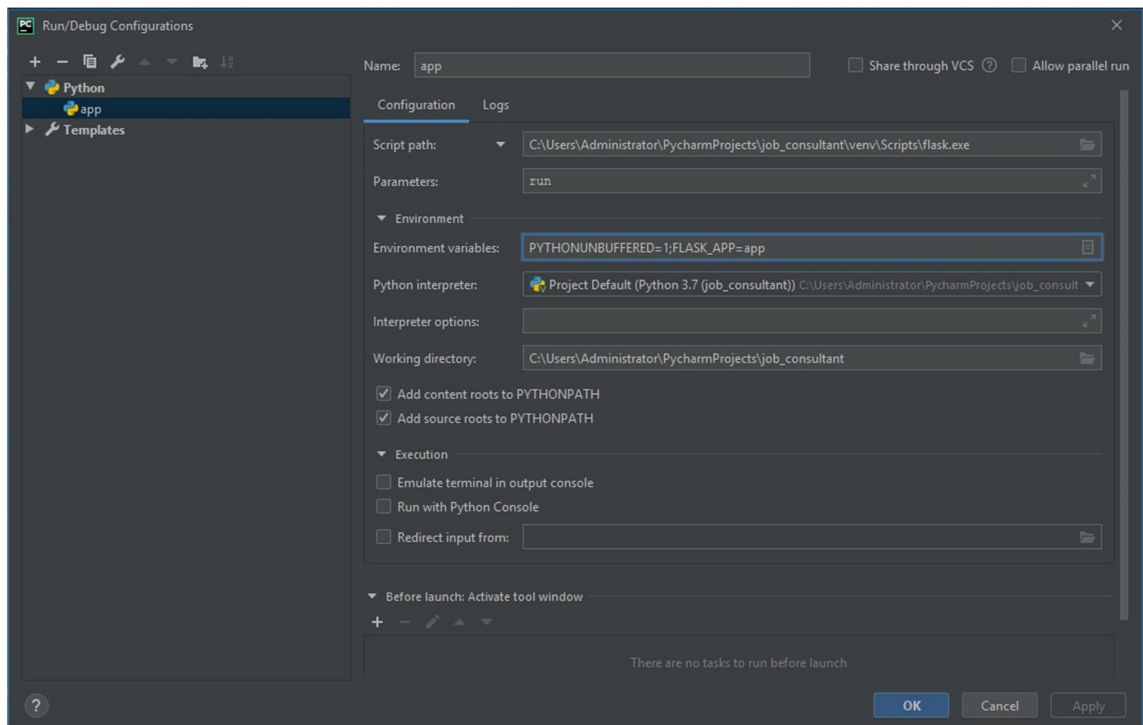


Ilustración 47. Configuración del entorno virtual de *PyCharm*

La configuración del túnel (*ngrok*) y del *webhook* de *Dialogflow* se ha descrito detalladamente en la sección [5.5.1 Estructura de la aplicación en Python](#).

B.2 Módulos del proyecto

Los módulos `M_text_pre_processing.py` y `M_syntactic_analysis.py` ya han sido adjuntados en el [apéndice A](#).

B.2.2 app.py

```
# LIBRARIES IMPORT
#url
from urllib.parse import urlparse, urlencode
from urllib.request import urlopen, Request
from urllib.error import HTTPError
#system & file types
import os
import json
#flask: create a service to run a webapp
from flask import Flask
from flask import request
from flask import make_response
from flask import session
#User modules
from Include.Inference.Inference import model_load, features_extraction,
model_predict
from Include.Filter.Filter import dataset_load, filter_salary

# Flask app should start in global layout
app = Flask(__name__)
#Global variables
g_job_profile = ""
g_min_salary,g_max_salary,g_range_salary=0,0,0

#Session settings
#app.config.from_object(__name__)
#app.secret_key = os.urandom(15)
#Session(app)

@app.route('/webhook', methods=['POST'])
def webhook():

    debug='on'
    #Request reception
    req = request.get_json(silent=True, force=True)           #Decoding the
request received on json format
    if debug=='on':
        print("REQUEST:")
        print(json.dumps(req, indent=4))
    #Processing the request for generating a result
    result = request_processing(req)
    result = json.dumps(result, indent=4)                     #Convert result
into json format for responding
    if debug=='on':
        print(result)
    response = make_response(result)
    response.headers['Content-Type'] = 'application/json'

    return response

def request_processing(request,debug='off'):
```

```

    if debug=='on':
        print ("REQUEST PROCESSING:
",request.get("queryResult").get("action"))
        #Get active contexts
        conversation_contexts=request.get("queryResult").get("outputContexts")
        #Action-dependent task
        webhook_actions_set=["job_profile","job_salary","job_city"]
#Actions allowed for dispatching
        webhook_action=request.get("queryResult").get("action")           #Action
of the processed request

    if webhook_action not in webhook_actions_set:
        print ("Please check your action name in DialogFlow...")
        return {}
    if webhook_action=="job_profile":
        string_result = action_job_profile(request)
    if webhook_action=="job_salary":
        string_result = action_job_salary(request,'on')

    if debug=='on':
        print ("RESULT after processing the request",string_result)
    response =
WebhookResult_assemble(string_result,conversation_contexts,webhook_action)

    return response

def action_job_profile(request,debug="off"):

    sentence = request.get("queryResult").get("queryText")

    model,model_att=model_load()
    df_input=features_extraction(sentence,model_att)
    y_out=model_predict(model,df_input)
    job_profile=y_out[0]
    if debug=='on':
        print("\tPredicted JOB PROFILE:",job_profile)

    #Result composition
    str_opening= "According to the description, I think you may be looking
for job positions as "
    str_body=job_profile.upper()
    str_closing= " Are you ok with that?"
    action_result = str_opening + str_body + str_closing

    if debug=='on':
        print("\tRESULT OF action_job_profile:",action_result)

    # Session action_job_profile result
    #session['job_profile'] = job_profile
    #Storing job_profile
    global g_job_profile
    g_job_profile = job_profile

    return action_result

def action_job_salary(request,debug="off"):

    #Get requested salary preferences
    range_salary =
request.get("queryResult").get("parameters").get("number_range")

```

```

min_salary =
request.get("queryResult").get("parameters").get("number_from")
max_salary =
request.get("queryResult").get("parameters").get("number_to")
#Check previous salary preferences
if range_salary == "" and min_salary == "":
    global g_range_salary, g_min_salary, g_max_salary
    range_salary = g_range_salary
    min_salary = g_min_salary
    max_salary = g_max_salary
else:
#Use input preferences
    salary_bound = 35000

    if range_salary != "":
        #By range
        range_salary = int(range_salary)
        min_salary = max(0, range_salary - salary_bound)
        max_salary = range_salary + salary_bound
    else:
        #By min/max
        range_salary = 0
        min_salary = int(min_salary)
        if max_salary == "":
            max_salary = 9000000
        else:
            max_salary = int(max_salary)
#Store salary preferences

g_min_salary, g_max_salary, g_range_salary = min_salary, max_salary, range_salary
if debug == 'on':
    print('Previous
preferences:', g_range_salary, g_min_salary, g_max_salary)
    print('Actual preferences', range_salary, min_salary, max_salary)

#Session action_job_profile result
#target_job=session.get('job_profile')

df_dataset=dataset_load()
lst_job_name, lst_job_bulletins, profile_matching =
filter_salary(g_job_profile, min_salary, max_salary, df_dataset)

# Result composition
if len(lst_job_bulletins)>0:
    str_opening="Here you can check the list of " +
str(len(lst_job_bulletins)) + " JOB BULLETINS out of "\
    + str(profile_matching) + " matching your preferences
for the " +g_job_profile+ " profile ... "
    str_body=""
    for item in lst_job_bulletins:
        str_body+=" // "+item
    #Avoid overwhelming the user
    char_limit=640
    if len(str_body)>char_limit:
        str_body=str_body[0:char_limit]+" // ... "
    str_closing=" Would you like to MODIFY the SALARY RANGE?"
    action_result = str_opening + str_body + str_closing
else:
    str_opening="Sorry, "
    str_body="we didn't find any matching job offer for that range and
" + g_job_profile + " profile."
    str_closing=" Would you like to MODIFY the SALARY RANGE?"
    action_result = str_opening + str_body + str_closing

```

```

    return action_result

def WebhookResult_assemble(data, conversation_contexts, action, debug="off"):

    speech = data #Speech output
    if debug=='on':
        print("\tSPEECH RESPONSE:", speech)
    #Context control
    check_context_lifespan=0
    for context in conversation_contexts:
        if (action=='job_profile' and "check_skills" in context["name"]) or
(action=='job_salary' and "confirm_profile" in context["name"]):
            check_context_name=context["name"]
            check_context_lifespan=context["lifespanCount"]
            check_context_param = context["parameters"]
            if debug == 'on':
                print("\t\tCONTEXT: check_skills")
                print("\t\tname:", check_context_name)
                print("\t\tlifespanCount:", check_context_lifespan)
                print("\t\tparameters:", check_context_param)
    #Function result
    if check_context_lifespan>0:
        assemble_result = {"fulfillmentText": speech,"source": "Inference
engine",
            "outputContexts": [ {"name":
check_context_name,"lifespanCount": 0} ] }
    else:
        assemble_result = {"fulfillmentText": speech,"source": "Inference
engine"}

    return assemble_result

if __name__ == '__main__':

    port = int(os.getenv('PORT', 5000))
    print("Starting app on port %d" % port)
    app.run(debug=True, port=port, host='0.0.0.0')

```

B.2.3 Inference.py

```

#SYSTEM MODULES
import os #Operating System functions
import pickle #Data saving library
import numpy as np #Mathematical functions & data handling
import pandas as pd #Data & statistical analysis

#USER MODULES
#Pre processing functions
from .M_text_pre_processing import textPreProcessing
#Syntactic analysis
from .M_syntactic_analysis import
chunk_lookup, chunk_lemma, grammar_chunk, items_over_threshold, POS_tagging
from .M_syntactic_analysis import get_synonyms

#MODEL LOAD
def model_load():

```

```

dirname = os.path.dirname(__file__) #Absolute path

file_path = os.path.join(dirname, 'Model/job_profile_model_NB.sav')
model = pickle.load(open(file_path, 'rb'))
file_path = os.path.join(dirname, 'Model/job_profile_att_model_NB.sav')
model_att = pickle.load(open(file_path, 'rb'))

return model,model_att

#EXTRACTING FEATURES
def features_extraction(sentence,model_att):

    #Settings
    synonyms=True
    #Input dataframe
    df_input=pd.DataFrame() #Empty dataframe
    for job in model_att:

df_input.insert(df_input.shape[1],job,np.ravel(np.zeros((1,1)).astype(int))
,allow_duplicates=False)
    #Features extaction
    grammar = r"""
    NP: {<N.*>}
    """
    index=0 #For future extension
    stop_words=[]
    list_chunk_bulletin=[] #Chunks in
actual bulletin
    sentence=textPreProcessing(sentence) #Remove
quotation mark symbols
    POS_tag=POS_tagging(sentence) #POS
tagging
    tree_phrase=grammar_chunk(grammar,POS_tag) #Chunking
    list_chunk=chunk_lookup(tree_phrase,info=False,debug=False) #List of
chunks
    for key in list_chunk:
        key_lemma=chunk_lemma(key,'n') #Get lemma
of chunks
        match=False
        for feature in model_att:
            #Do not check stop words
            if key_lemma in stop_words:
                break
            #First check: most common features
            if key_lemma==feature:
                df_input.loc[index,feature]+=1
                match=True
                break
            #Second check: synonyms of most common features
            if synonyms==True:
                if match==False:
                    for feature in model_att:
                        key_lemma=chunk_lemma(key,'n')
                        if key_lemma in get_synonyms(feature,pos='NOUN'):
                            df_input.loc[index,feature]+=1
                            match=True
                            break

    return df_input

#MAKING PREDICTIONS

```

```
def model_predict(model,df_input):

    x_input=df_input.values[:,:]
    y_out = model.predict(x_input)

    return y_out
```

B.2.4 Filter.py

```
#SYSTEM MODULES
import os                #Operating System functions
import pickle           #Data saving library
import pandas as pd     #Data & statistical analysis

#MODEL LOAD
def dataset_load():

    dirname = os.path.dirname(__file__)           #Absolute path

    file_path = os.path.join(dirname, 'Dataset/job_buletin_dataset.sav')
    df_dataset = pickle.load(open(file_path, 'rb'))

    return df_dataset

#PREFERENCES FILTER
def filter_salary(target_job,min_salary,max_salary,df_dataset):

    #Total number offers matching the profile
    df_temp=df_dataset[(df_dataset.Profile==target_job)]
    profile_matching=df_temp.shape[0]
    #Intermediate dataframe to apply the filter
    df_temp=df_dataset[(df_dataset.Profile==target_job)&
    (df_dataset.Salary_min>=min_salary)&
    (df_dataset.Salary_max<=max_salary)]
    #Output list
    list_job_name=df_temp['Job_name'].values
    list_bulletin=df_temp['Bulletin'].values

    return list_job_name,list_bulletin, profile_matching
```


Apéndice: Artículo de investigación en español

Apéndice: Artículo de investigación en inglés

Agente conversacional para búsqueda y recomendación de ofertas laborales

Héctor Gómez Loizaga

Universidad Internacional de la Rioja, Logroño (España)

Fecha: 19/09/2019



RESUMEN

El objetivo del presente trabajo es el desarrollo de la arquitectura y metodología utilizada por un agente conversacional para interactuar con un usuario y extraer la información necesaria que le permita actuar como orientador laboral. Se aplicarán técnicas de procesamiento del lenguaje natural orientadas a la extracción de información que será utilizada para la generación de características que puedan ser procesadas por algoritmos de aprendizaje automático. Es especialmente relevante que dichas características sean suficientemente discriminatorias entre los distintos perfiles laborales. El motor de inferencia se implementará como un clasificador que asignará el grupo o perfil laboral al que el usuario potencialmente pertenece y en función de información adicional sobre preferencias y formación del interlocutor, encontrará y mostrará un conjunto de ofertas laborales a las que el usuario esté capacitado para aplicar y que le puedan resultar de interés.

PALABRAS CLAVE

Agente conversacional,
Aprendizaje automático,
Extracción de características,
Procesamiento del lenguaje natural

I. INTRODUCCIÓN

ESTE trabajo trata de investigar la viabilidad de aplicar técnicas de aprendizaje automático y de procesamiento del lenguaje natural para incrementar el nivel de automatización y la usabilidad de los portales de búsqueda de empleo.

Se pretende abarcar el ciclo completo del desarrollo y puesta en producción de una herramienta analítica, incluyendo el procesado de datos para la extracción de indicadores representativos del proceso a resolver, la inferencia sobre los datos para generar conocimiento y la asistencia a la toma de decisiones (recomendación).

Las técnicas de procesamiento del lenguaje natural se han utilizado para acometer dos objetivos: la extracción de información relevante sobre datos no estructurados (texto) y la agilización y naturalización de la comunicación con el usuario. Por otro lado, las técnicas de aprendizaje automático han sido la base para la construcción de un clasificador que asigna el perfil laboral al usuario.

La herramienta desarrollada consiste en un agente conversacional que actúa como interfaz para el usuario. Este agente recopila la información que necesita el módulo de inferencia (clasificador) y utiliza los resultados generados por este para sugerir al usuario el conjunto de ofertas de trabajo que mejor se adaptan a su formación y sus preferencias. Con esta herramienta el usuario es capaz de encontrar de forma efectiva la información que buscaba en el marco de una conversación amena y concisa, mejorando notablemente la experiencia de usuario de este tipo de sistemas.

II. ESTADO DEL ARTE

En esta sección, se documentan los ámbitos tecnológicos que sirven de base para el desarrollo del presente trabajo.

A. Procesamiento del lenguaje natural (NLP)

El procesamiento del lenguaje natural (*Natural Language*

Processing, NLP) consiste en una serie de técnicas enfocadas al tratamiento y análisis del lenguaje natural. Su principal objetivo es que las computadoras puedan llevar a cabo tareas donde participa el lenguaje natural o humano [8]. Es sin duda un campo altamente interdisciplinar que abarca un amplio rango de tareas y los conocimientos asociados que se requieren para acometerlas: procesamiento de la señal (ingeniería electrónica y de telecomunicaciones), análisis sintáctico e interpretación semántica de palabras y frases (informática), morfología, fonología y pragmática (lingüística), estudio de los mecanismos cognitivos para la adquisición del lenguaje y cómo influye la sociedad en el uso de la lengua (sociólogos y psicólogos).

Las aplicaciones de procesamiento del lenguaje natural se distinguen de otros sistemas de procesamiento de datos por requerir el uso de conocimiento del lenguaje. Dicho conocimiento será necesario para el desarrollo de sistemas que sean capaces de realizar diverso tipo de tareas como las de análisis morfológico, análisis sintáctico, uso de gramáticas, análisis semántico y análisis pragmático del lenguaje.

Independientemente de la diversidad de idiomas y sistemas lingüísticos propios de cada lenguaje, un aspecto común que dificulta la computación del mismo es la ambigüedad, entendida como aquellas situaciones en las que la información se puede entender o interpretar de más de una forma.

1) Representación del significado

Los recursos lingüísticos (diccionarios, lexicón, tesauros, bases de datos de relaciones léxicas, corpus lingüísticos) sirven de base para la creación de las distintas estrategias de representación del significado de las palabras. Entre las más utilizadas destacan las siguientes:

- **Bases de datos relacionales:** las palabras aparecen agrupadas por significado y las relaciones semánticas que se dan entre ellas. *Wordnet* [7], es un representativo ejemplo de este tipo de estrategias de representación del significado. A pesar de que este tipo de recursos funciona bastante bien, en muchas ocasiones se pierden matices sobre el contexto.

- **Representación discreta:** partiendo de un vocabulario determinado, se representan las oraciones o textos mediante un vector donde cada elemento indica el número de veces que aparece cada palabra del vocabulario en dicho fragmento del documento (*term frequency*). Referenciadas como *bag-of-words*, este tipo de representación no evalúa el orden de aparición de las palabras (contexto).
- **Representación distribuida:** consiste en la representación de una palabra mediante un vector denso. Conocidas como *word vectors*, *word embeddings* o *word representations*, constituyen una representación manejable (computacionalmente) para expresar información semántica sobre similitud entre palabras. Los vectores se extraen mediante procedimientos estadísticos soportados por modelos de aprendizaje automático (redes neuronales) que trabajan sobre corpus lingüísticos.

2) Modelos del lenguaje

Principalmente, el problema de la modelización matemática para el análisis y tratamiento del lenguaje se aborda desde dos enfoques:

- **Lógico:** los expertos en lingüística establecen reglas para reconocer patrones estructurales y reflejan la estructura lógica del lenguaje. Hacen uso de lenguajes formales que permiten una representación inequívoca de los objetos, sus propiedades y las relaciones que guardan entre sí. Además, al estar basados en reglas permiten producir nuevo conocimiento mediante mecanismos de inferencia. Se basan en los estudios publicados por Chomsky en los años cincuenta [18].
- **Probabilístico:** basados en el análisis de colecciones de datos que conforman un corpus lingüístico. Se calculan las frecuencias de las unidades lingüísticas y se aplican técnicas de aprendizaje automático. La construcción de estos modelos se realiza principalmente mediante técnicas de entrenamiento supervisado, aunque también se está investigando la utilización de entrenamiento no supervisado con el objetivo de reducir el esfuerzo de tener que etiquetar los inmensos corpus sobre los que se entrena.

3) Aplicaciones y herramientas para PLN

La interdisciplinariedad del campo conlleva implícitamente que exista una gran variedad de aplicaciones basadas en técnicas de procesamiento del lenguaje natural. Entre ellas podemos destacar el desarrollo de agentes conversacionales (*chatbots*), la traducción automática y la búsqueda de respuestas (*Question Answering, QA*).

De entre las numerosas herramientas y librerías enfocadas al desarrollo de aplicaciones de procesamiento del lenguaje [19] destacamos el *Natural Language Toolkit* y la *Stanford CoreNLP*.

Por otro lado, cabe destacar el interés demostrado por las grandes empresas de desarrollo tecnológico (*Google Cloud Natural Language*, *Amazon Comprehend*, *IBM Watson Natural Language Understanding*, *Microsoft Language Understanding*) en copar el mercado con sus propias soluciones comerciales, las cuales ofrecen servicios completos, escalables y fáciles de integrar a través de las APIs disponibles y que normalmente son distribuidos desde sus plataformas *cloud*.

B. Extracción automática de características

La extracción de características es un proceso de codificación y reducción que tiene el objetivo de reducir un conjunto de datos a características o atributos que captan con precisión la información contenida en los datos y que sean tratables por programas informáticos. Se enumeran a continuación algunos de los métodos de uso más habitual para la extracción de características sobre documentos de texto.

1) Tratamiento de datos no estructurados

Una de las dificultades que tiene trabajar sobre documentos de texto es el hecho de que se trata de datos que no incorporan una

estructura implícita y que por tanto requieren de un procesamiento adicional para añadir información o estructura a los datos de partida, de tal forma que puedan ser gestionados de forma adecuada por los motores de inferencia [5][6]. Mediante el uso de herramientas y técnicas de *Big Data* [28], tales como *Hadoop* o bases de datos no relacionales como *NoSQL*, se puede convertir dicha información en información relevante. En el caso de documentos de texto, se hará prácticamente imprescindible el uso de técnicas de procesamiento del lenguaje natural.

2) Rastreo de datos (web scraping)

El rastreo de datos es una técnica que permite navegar de forma automática por la red y extraer información de los portales web que se visitan. La extracción de información se basa en el conocimiento de las etiquetas HTML con las que se estructuran las distintas secciones y el contenido en las páginas web. Existen varias herramientas que permiten crear sin mucho esfuerzo un software que realice esta función y que recibe entre otras las denominaciones *spider*, *bot* o *crawler*. *Python* es sin duda el lenguaje mayormente empleado para rastrear mediante las implementaciones de varias de sus librerías: *Scrapy*, *BeautifulSoup* y *Selenium*.

3) Reconocimiento de entidades (NER)

El reconocimiento de entidades (*Named Entity Recognition*), es una técnica que permite localizar e identificar todas las menciones textuales a una serie de entidades predefinidas. Estas entidades podrían ser organizaciones, personas, ubicaciones, fechas, instalaciones, etc. Cualquier instancia de estas entidades será localizada y etiquetada junto con su correspondiente entidad [31]. Las implementaciones disponibles están basadas en el uso de algoritmos de aprendizaje automático, gramáticas y corpus lingüísticos. Este tipo de herramientas juega un papel muy interesante en las tareas de recuperación de información (*information retrieval*), ya que permite obtener directamente los pares clave-valor evitando así tener que recopilar grandes cantidades de texto a las que se les tendría que aplicar posteriormente expresiones regulares.

4) Clasificación de texto

Como paso previo a las tareas de clasificación de texto necesitaremos obtener descriptores de la información contenida en el texto que sea manejable por los algoritmos.

- **Tf-idf** (*Term frequency – inverse document frequency*): cuantificador utilizado para ponderar la relevancia de una palabra en un documento que forma a su vez parte de una colección de documentos. El valor asignado aumenta cuanto mayor sea la frecuencia de la palabra en un documento dado y disminuye cuando mayor sea la frecuencia de dicha palabra en el global de documentos. De esta forma se trata de penalizar las palabras genéricas que se repiten numerosamente en la colección de documentos. Su fórmula matemática es:

$$tf\ idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

siendo $tf(t, d)$ la frecuencia de aparición del término t (palabra) en el documento d y $idf(t, D) = \log(|D| / |\{d \in D : t \in d\}|)$, con D el número total de documentos y $|\{d \in D : t \in d\}|$ el número de documentos donde aparece el término t , al que se suma la unidad para evitar la división entre cero si el término no aparece en la colección [32].

- **Latent Dirichlet Allocation (LDA):** técnica ideada para la detección de tópicos (categorías) en documentos de textos. Este método trata de modelar los documentos como un conjunto finito de tópicos asumiendo que estos siguen una distribución de Dirichlet. A su vez, cada tópico se asume que puede ser descrito por una distribución de palabras [33].

C. Aprendizaje automático (ML)

La disciplina del aprendizaje automático engloba una serie de métodos y algoritmos orientados a la extracción de conocimiento de

forma autónoma a partir de datos. Los datos, representan los ejemplos que los algoritmos procesan para ajustar sus parámetros internos. De esta forma, durante proceso de entrenamiento (ajuste) a medida que se procesa un mayor número de muestras el algoritmo adquiere mayor conocimiento a través de la experiencia [21]. A su vez, el entrenamiento se corresponde matemáticamente con la resolución de problemas de optimización, donde el procesado de cada muestra sirve al algoritmo para ajustar mejor sus parámetros buscando minimizar alguna métrica o medida del error cometido al realizar inferencias.

Podemos identificar principalmente dos requisitos sobre los datos de entrenamiento:

- **Conjunto de datos abundantes:** para que los algoritmos tengan la oportunidad de cubrir la diversidad del total de la población que se pretende clasificar o mostrar la dinámica del sistema a modelar en todo su rango de actuación.
- **VARIABLES REPRESENTATIVAS:** las variables seleccionadas (subconjunto de los datos de entrada) deben ser suficientemente predictores del sistema a modelar, es decir, deben ser aquellas variables que mejor describan y/o diferencien las respuestas del sistema.

1) Tipos de algoritmos de aprendizaje automático

Podemos realizar una clasificación de los distintos tipos de algoritmos de aprendizaje automático en función de las técnicas para su implementación y de la metodología de entrenamiento utilizada:

- **Aprendizaje supervisado:** se dispone de un conjunto de datos etiquetados con la respuesta asociada. Durante el entrenamiento, en sucesivas iteraciones se procesa cada muestra y se compara con la salida esperada. La cuantía del error cometido en la predicción se evalúa a través de alguna métrica y se utiliza para reajustar los parámetros internos del sistema.
 - o Aplicable a problemas de clasificación (asignar una entre n clases a cada muestra) y regresión (predicción de un valor numérico).
- **Aprendizaje no supervisado:** en este caso no se dispone de salida o respuesta esperada, sino que se procede a determinar la estructura interna de los datos. Estas técnicas trabajan con métricas de similitud entre los vectores de características de las muestras.
 - o Aplicable a problemas de *clustering* (agrupar instancias muy similares en categorías) y detección de anomalías (detectar valores muy poco habituales).
- **Aprendizaje semisupervisado:** este tipo de algoritmos combina aprendizaje en base a etiquetas y clasificación en función de la estructura interna de los datos.
- **Aprendizaje por refuerzo:** los algoritmos de aprendizaje por refuerzo están orientados a la obtención de conocimiento del entorno con el que se interactúa y permiten tener en consideración las consecuencias futuras de realizar una acción en el momento de tomar la decisión [24]. Matemáticamente se modelan mediante los procesos de decisión de Markov.
 - o Aplicable a la implementación de agentes inteligentes.

2) Modelos de aprendizaje automático

Existen múltiples implementaciones en función de la tipología del problema y el objetivo a cubrir. Cada modelo se diferencia en la forma en la que aproximan el proceso de aprendizaje pudiendo encontrar enfoques analíticos (regresión logística), geométricos (*Support Vector Machines*), estructurales (árboles de decisión y *Random Forest*), probabilísticos (redes bayesianas) y conexionistas (redes neuronales), entre otros.

D. Agentes conversacionales

Los agentes conversacionales son sistemas diseñados para comunicarse con los usuarios mediante el uso del lenguaje natural. Si bien la interacción suele establecerse principalmente a través de mensajes de voz (audio) o mediante la escritura (texto), también se

puede hacer uso de otras modalidades de la comunicación integrando gráficos, imágenes o animaciones, que pueden contribuir a la construcción de entornos altamente interactivos. Los agentes conversacionales son empleados en ámbitos tan diversos como el entretenimiento, la educación, psicología, idiomas, ventas, marketing, etc. Entre sus principales aplicaciones podemos destacar su uso en sistemas de atención al cliente así como en la asistencia al proceso de compra, donde se presentan como interfaz para el motor de recomendación.

1) Tipos de agentes conversacionales

En base al tipo de conversación que sean capaces de mantener y el objetivo con el que se hayan implementado, distinguimos entre:

- **Agentes conversacionales dedicados:** diseñados para el desarrollo de una tarea concreta y por tanto restringidos a un dominio de aplicación. Generalmente, esto implica que están enfocados a la resolución de tareas sencillas, para las cuales es suficiente mantener una conversación corta que requiera pocas interacciones con el usuario. En esta categoría encontramos asistentes personales como Alexa (Amazon), Siri (Apple), Google Assistant, etc.
- **Chatbots:** Permiten entablar conversaciones extensas con múltiples interacciones, basadas en un diálogo no necesariamente estructurado y que pueden versar sobre cualquier materia.

2) Arquitectura de los agentes conversacionales

Un agente conversacional presentará los siguientes módulos [8]:

- **Reconocimiento del habla:** generación de texto (palabras) a partir de una señal acústica. Usa técnicas de modelado acústico-fonético, gramáticas de estados finitos y modelos probabilísticos.
- **Comprensión del lenguaje natural:** implementa tareas de procesamiento del lenguaje natural enfocadas a extraer el significado (semántica) de las frases. Basado en plantillas (*frames* [36]) con pares atributo-valor que el agente rellena tras examinar el texto de entrada.
- **Gestión del diálogo:** determina la acción a realizar y/o el concepto a transmitir en el turno de palabra actual como respuesta a la petición realizada por el usuario. Necesita mantener el estado y flujo de la conversación y requerir información al usuario cuando esta sea necesaria para generar la respuesta. El flujo del diálogo suele gestionarse mediante máquinas de estados finitos y reglas de producción.
- **Generación del lenguaje natural:** escoge los conceptos a transmitir y planificar la forma de expresarlos [37]. Se procede primero a la generación de una frase des lexicalizada (con huecos) que se rellena a continuación con valores que serán inferidos en función de la información extraída del usuario.
- **Conversión del texto al habla:** sintetiza en formato acústico el mensaje a transmitir. Puede utilizar técnicas de síntesis de formantes, síntesis articularia y síntesis basada en concatenación (resultado más natural).

3) Dialogflow

Dialogflow es la plataforma de Google de desarrollo de productos orientados a facilitar la interacción entre humanos y computadoras basada en aprendizaje automático y técnicas de procesamiento del lenguaje natural. Consta de una serie de elementos y funcionalidades que le permiten la creación, configuración y entrenamiento de agentes conversacionales de forma cómoda e intuitiva:

- **Intención (*intents*):** un *intent* sirve para relacionar una serie de mensajes (frases) que el usuario puede emitir con alguna tarea que el agente está capacitado para resolver. Los *intents* reaccionan ante un conjunto de frases a las que se le asignan una serie de respuestas (estáticas, dinámicas o generadas por aplicaciones externas) durante el entrenamiento del agente. A su

vez, *Dialogflow* permite la activación condicionada de los *intents* a que ciertos contextos estén activos. Por tanto, los contextos sirven para guiar el flujo de la conversación y para desambiguar el significado de las frases.

- **Entidades** (*entities*): *Dialogflow* se basa en *NER* para resolver los pares atributo-valor de las plantillas (*intents*) que aplica sobre los textos de entrada.
- **Fullfillment**: API que habilita la comunicación entre *Dialogflow* y cualquier aplicación externa mediante el uso de *webhooks*. Permite generar las respuestas del agente mediante el uso de motores de inferencia o reglas de producción externas.
- **Integrations**: permite integrar de forma ágil el agente conversacional con los servicios de mensajería, asistentes y aplicaciones de chat más populares: Facebook Messenger, Slack, Twitter, Kik, Line, Skype, Telegram, Twilio and Viber, Google Assistant, Amazon Alexa, Microsoft Cortana, etc.

III. OBJETIVOS Y METODOLOGÍA

A. Objetivo general

El objetivo principal del presente trabajo es investigar la viabilidad del uso de técnicas de procesamiento del lenguaje natural y de aprendizaje automático para naturalizar y asistir el proceso de búsqueda de trabajo a través de medios digitales. Implícitamente, se trata de construir una herramienta de búsqueda de empleo con interfaz implementada a través de un agente conversacional que identifique el perfil laboral del usuario y filtre la base de datos de ofertas laborales en función del perfil asignado y las preferencias del usuario.

B. Objetivos específicos

Los objetivos específicos descomponen el objetivo principal en tareas contenidas que pueden ser planificadas y abordadas con mayor eficiencia. Se enumeran a continuación las metas que se pretenden abarcar:

- Extracción de características sobre datos no estructurados. Este tipo de datos suele suponer un alto porcentaje de la cantidad de datos generados por las empresas (hasta un 80%)[5] [6].
- Comparar el rendimiento de distintas técnicas de clasificación automática y evaluar los requisitos para ser aplicadas.
- Usar técnicas de filtrado y recomendación para refinar la clasificación de categorías y para ayudar a obtener resultados más precisos.
- Guiar el proceso de extracción de información y naturalizar la comunicación con el usuario mediante el uso de agentes conversacionales como interfaz para el sistema.

C. Metodología

Se comenzará acudiendo a repositorios de datos y seleccionando algún *dataset* que se adapte a las necesidades del trabajo. Tras evaluar su representatividad, se ponderará la posibilidad de ampliar el conjunto de datos mediante técnicas de *web scraping*.

El siguiente paso consistirá en extraer características de nuestra fuente de datos no estructurada. Para ello, se utilizarán técnicas de procesamiento del lenguaje natural mediante analizadores morfosintácticos y un posterior filtrado aplicando gramáticas para extraer los constituyentes gramaticales que puedan ser más predictivos de nuestro conjunto de datos.



Figura 1. Pipeline de procesamiento del lenguaje natural

Se comparará el grado de representatividad que se obtiene del uso de sintagmas nominales, sintagmas verbales o la combinación de ambos. Se contabilizará la frecuencia de aparición de los

constituyentes gramaticales obtenidos y se seleccionarán los más frecuentes (matriz de frecuencia) como características de entrada para el motor de inferencia. También se probarán distintos enfoques orientados a refinar el proceso de selección de características para conseguir que sean lo más representativas posible y tratar de reducir el solapamiento entre categorías que sean similares. Se utilizará la sinonimia para cubrir las distintas variantes que el usuario puede utilizar para transmitir el contenido.

La información extraída de la descripción de cada profesión será utilizada para alimentar los algoritmos de aprendizaje automático (motores de inferencia) encargados de catalogar el perfil laboral del usuario. Para este fin, se van a utilizar y comprar los resultados de dos tipos de modelos de aprendizaje automático: las redes bayesianas y las redes neuronales. Los primeros se han seleccionado como técnica preferente debido a su demostrada eficacia y extendido uso para la clasificación de texto [26]. Los segundos se utilizarán por ser uno de los modelos más potentes y flexibles. Reservaremos un 80% de los datos para el entrenamiento de los modelos y un 20% para la evaluación de los mismos.

La evaluación de los modelos bayesianos se realizará mediante la métrica *f1*, basada en la matriz de confusión [27]:

$$precision = \frac{VP}{VP+} \quad recall = \frac{VP}{VP+FN}$$

$$f1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

Por su parte, el rendimiento de la clasificación mediante redes neuronales se evaluará mediante la función de pérdida optimizada durante el entrenamiento. La función escogida a tal efecto es *Categorical Cross-Entropy*:

$$CE = -\log \left(\frac{e^{s_p}}{\sum_j e^{s_j}} \right)$$

Para evitar presentar información abundante e innecesaria al usuario, se realiza un filtrado atendiendo al perfil laboral asignado y a información adicional extraída del usuario, tal como la horquilla salarial, la ciudad en la que busca incorporarse, etc. Para la obtención de estos datos se utilizará reconocimiento de entidades (*NER*) en el lado del agente conversacional y expresiones regulares en el lado del motor de filtrado en *Python*.

En todo momento, el agente conversacional se presentará como el punto de entrada y salida para el usuario. Nos centraremos en la definición de la estrategia de gestión del diálogo y en verificar que el usuario ha aportado la información requerida mediante el uso de máquinas de estado finitos y el empleo de técnicas basadas en *frames*. También se prestará atención a las estrategias para conseguir mayor variabilidad en la formulación de las preguntas, lo cual aportará más naturalidad a la conversación.

IV. CONTRIBUCIÓN

El objetivo del presente trabajo es acometer el ciclo completo de desarrollo de una aplicación basada en técnicas de inteligencia artificial. El sistema desarrollado ha sido dividido en cuatro módulos atendiendo a la tarea a completar.

A. Módulo de extracción de características

- **Cometido**: tratamiento de datos no estructurados para obtención de indicadores predictores para el problema a resolver.
- **Técnicas**: procesamiento del lenguaje natural.
- **Herramientas**: NLTK en *Python*.
- **Entradas**: texto con descripción de tareas a realizar en cada trabajo.
- **Salidas**: frecuencia de palabras relevantes en cada texto.

B. Módulo de inferencia

- **Cometido:** asignación del perfil laboral más probable.
- **Técnicas:** clasificación mediante aprendizaje automático.
- **Herramientas:** *Keras* y *Scikit-Learn* en *Python*.
- **Entradas:** frecuencia de palabras relevantes en cada texto.
- **Salidas:** categoría laboral.

C. Módulo de filtrado y recomendación

- **Cometido:** selección en base de datos de ofertas relevantes para el usuario.
- **Técnicas:** filtrado con claves de búsqueda.
- **Herramientas:** *Python*.
- **Entradas:** categoría laboral asignada y preferencias del usuario.
- **Salidas:** conjunto de ofertas laborales.

D. Módulo de interfaz conversacional

- **Cometido(1):** guiar al usuario para asegurar que aporta información precisa y completa para realizar la búsqueda.
- **Técnicas:** modelo de conversación basado en *frames* y gestión del diálogo mediante máquinas de estados finitos.
- **Herramientas:** *Dialogflow*.
- **Cometido(2):** tratamiento de datos no estructurados para obtención de preferencias del usuario.
- **Técnicas:** expresiones regulares y reconocimiento de entidades.
- **Herramientas:** *Python* y *Dialogflow*.
- **Entradas:** texto de usuario.
- **Salidas:**
 - **Módulo de inferencia:** texto introducido por el usuario.
 - **Módulo de filtrado y recomendación:** preferencias del usuario.
 - **Usuario:** pregunta para requerir información, texto informativo y conjunto de ofertas laborales encontradas.

V. RESULTADOS O EVALUACIÓN

A. Pre-procesado de datos

Originalmente se utilizó el *dataset* “Data Science for Good: City of Los Angeles” (en adelante *DS_O*) publicado en la plataforma web *Kaggle* [2]. Al comprobarse que el *dataset* no era suficientemente representativo se obtuvo otro *dataset* mediante técnicas de *web scrapping* (*DS_E*). Los documentos contienen distintas secciones en formato texto. La etapa de pre-procesado recolecta la información relevante para los algoritmos de clasificación: nombre de la oferta laboral y el texto de las secciones ‘*Duties*’, ‘*Salary*’, las cuales contienen la descripción de las tareas que realizan en el puesto ofertado y el rango salarial al que se puede acceder respectivamente.

B. Etiquetado del dataset

La aplicación de técnicas de agrupamiento (*clustering*) para obtener un etiquetado automático de los boletines de trabajo aporta resultados poco consistentes (figura 2).

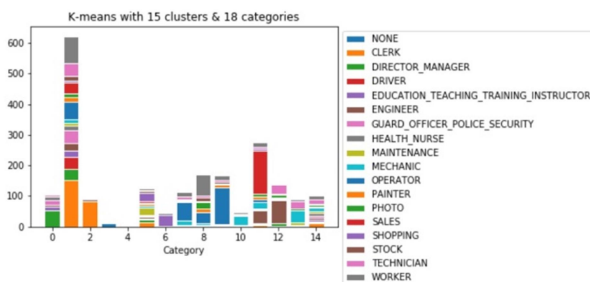


Figura 2. *Clustering* para etiquetado automático

Como alternativa, se utilizó un etiquetado semi-guiado que conllevó la definición de 18 categorías:

```
{'NONE': 0, 'CLERK': 1, 'COLOR_PAINTER': 2,
'DIRECTOR_MANAGER': 3, 'DRIVER': 4,
'EDUCATION_TEACHING_TRAINING_INSTRUCTOR': 5, 'ENGINEER':
6, 'GUARD_OFFICER_POLICE_SECURITY': 7, 'HEALTH_NURSE': 8,
'MAINTENANCE': 9, 'MECHANIC': 10, 'OPERATOR': 11, 'PHOTO': 12,
'PRINTER': 13, 'SALES': 14, 'SHOPPING': 15, 'STOCK': 16,
'TECHNICIAN': 17, 'WORKER': 18}
```

C. Selección de características y rendimiento de clasificadores

Se muestran los rendimientos obtenidos al aplicar distintas metodologías para la extracción y selección de características.

1) Comparativa de datasets y tipo de atributos

Se compara el rendimiento entre el *dataset* original (*DS_O*) y extendido (*DS_E*) para la extracción características (*features*) basadas en verbos (V) y en nombres (N).

Dataset	Features	Categorías	Rendimiento en test	Modelo
<i>DS_O</i>	V: 44	Manual:22	0.1863	NB
<i>DS_E</i>	V: 283	Dirigido:19	0.4879	NB
<i>DS_O</i>	N: 205	Manual:22	0.4412	NB
<i>DS_E</i>	N: 1027	Dirigido:19	0.6066	NB

Tabla 1. Comparativa de *datasets* y atributos verbos vs. nombres

A partir de este momento, todas las pruebas se realizan sobre el *dataset* extendido (*DS_E*) utilizando los modelos bayesianos (NB).

2) Reducción de dimensionalidad

Se comparan resultados de realizar la extracción y selección de características sobre el total de los documentos frente a proceder evaluando los documentos por grupos de categorías.

Dataset	Features	Frecuencia mínima	Features categoría	Rendimiento en test
Completo	N: 1027	20	Sin límite	0.6066
Grupos	N: 316	20	60	0.6352

Tabla 2. Comparativa selección sobre *dataset* completo vs. grupos

Todas las pruebas que se muestran a continuación aplican la metodología de selección de características por grupos.

3) Mejora de la representatividad de las características

Se analiza el rendimiento individual de cada categoría mediante las métricas obtenidas por los algoritmos de redes neuronales (NN).

Categoría	precision	recall	f1-score	Muestras
1	0.74	0.74	0.74	61
2	0.64	0.64	0.64	22
3	0.5	0.62	0.55	34
4	0.56	0.6	0.58	15
5	0.58	0.55	0.56	20
6	0.64	0.55	0.59	51
7	0.72	0.5	0.59	26
8	0.25	0.25	0.25	8
9	0.2	0.08	0.11	13
10	0.44	0.55	0.49	22
11	0.65	0.67	0.66	91
12	0.41	0.41	0.41	17
13	0.72	0.5	0.59	36
14	0.73	0.72	0.73	46

15	0.4	0.4	0.4	10
16	0.12	0.12	0.12	8
17	0.22	0.19	0.2	32
18	0.46	0.66	0.54	56

Tabla 3. Rendimiento por categorías.

Las filas marcadas se corresponden con categorías que tienen pocas muestras o que comparten características comunes (acopladas): *Health_Nurse* (48), *Maintenance* (55), *Mechanic* (142), *Photo* (75), *Shopping* (44), *Stock* (46), *Technician* (154).

Primero se eliminan las clases que disponen de pocas muestras. A continuación, para intentar desacoplar las categorías similares se ha procedido a eliminar las características que son frecuentes y comunes entre varias categorías. Otro enfoque ha sido utilizar la matriz *tf-idf*. Se añaden también los resultados obtenidos al eliminar las categorías que se acoplan.

Metodología	Features	Categorías	Rendimiento en test
Mínimo muestras	N: 298	13	0.7103
Eliminación características	N: 291	13	0.7051
Matriz <i>tf-idf</i>	N: 298	13	0.6974
Eliminación acopladas	N: 242	9	0.8513

Tabla 4. Enfoques para mejorar la representatividad.

4) Word vectors

Resultados de trabajar sobre representaciones densas.

Features / Vocabulario	Entrada	Categorías	Rendimiento en test	Modelo
N: 298	<i>Tf</i>	13	0.7103	NB
298 (filtrado)	<i>Word vectors</i>	13	0.5410	NN
11316 (completo)	<i>Word vectors</i>	13	0.7128	NN

Tabla 5. Resultado para modelos sobre representaciones densas.

D. Filtrado y recomendación

Se muestra la base de datos sobre la que trabajan los algoritmos de filtrado.

	Job_name	Profile	Salary_range	Bu
10	AIRPORT CHIEF INFORMATION SECURITY OFFICER	GUARD_OFFICER_POLICE_SECURITY	(125.551, 183.576)	AIRPORT CHIEF INFORMATION SECURITY OFF
11	AIRPORT ENGINEER	ENGINEER	(105.305, 163.532)	AIRPORT ENGINEER 7256 0706
12	AIRPORT GUIDE	GUARD_OFFICER_POLICE_SECURITY	(31.925, 46.687)	AIRPORT GUIDE 0845 0420
13	AIRPORT INFORMATION SPECIALIST	DIRECTOR_MANAGER	(42.991, 78.529)	AIRPORT INFORMATION SPECIALIST 1783 1211
14	AIRPORT LABOR RELATIONS ADVOCATE	DIRECTOR_MANAGER	(93.918, 137.327)	AIRPORT LABOR RELATIONS ADVOCATE 0201
15	AIRPORT MANAGER	DIRECTOR_MANAGER	(99.326, 188.567)	AIRPORT MANAGER 7260 1202
16	AIRPORT POLICE CAPTAIN	GUARD_OFFICER_POLICE_SECURITY	(142.568, 197.001)	AIRPORT POLICE CAPTAIN 3228 0216
17	AIRPORT POLICE LIEUTENANT	GUARD_OFFICER_POLICE_SECURITY	(102.186, 134.028)	AIRPORT POLICE LIEUTENANT 3227 0916
18	AIRPORT POLICE OFFICER	GUARD_OFFICER_POLICE_SECURITY	(51.448, 83.019)	AIRPORT POLICE OFFICER 3225 11090 0601
19	AIRPORT POLICE SPECIALIST	GUARD_OFFICER_POLICE_SECURITY	(0.0, 0.0)	AIRPORT POLICE SPECIALIST 3236 063017

Figura 3. Base de datos de ofertas laborales y preferencias.

E. Agente conversacional

El siguiente esquema representa la máquina de estados que determina las reglas de producción activas (estados) y los contextos (arcos) que condicionan el flujo del diálogo del agente conversacional.

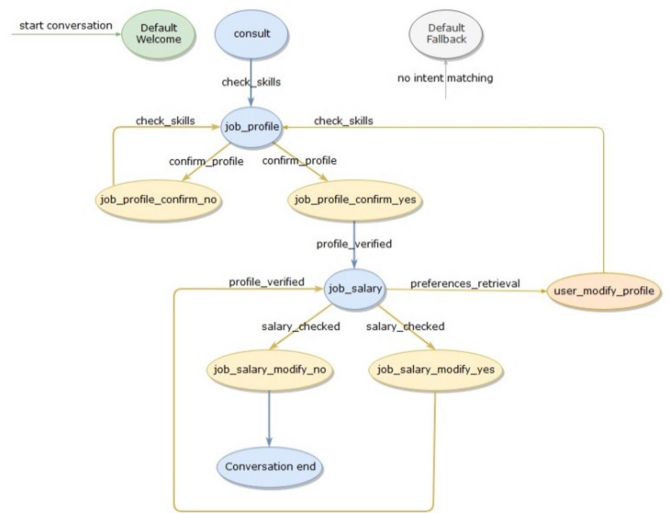


Figura 4. Máquina de estados para el control del flujo del diálogo.

VI. DISCUSIÓN O ANÁLISIS DE RESULTADOS

Pasamos a comentar a continuación algunas reflexiones sobre los objetivos específicos del proyecto.

A. Extracción de características

El uso de técnicas de procesamiento del lenguaje natural ha resultado fundamental para la extracción de información relevante sobre los datos de entrada (no estructurados). Es aquí, donde las técnicas de aprendizaje muestran su potencia al captar patrones y estructuras en los datos; habría resultado inviable capturar la complejidad y la casuística que puede llegar a darse en los datos de entrada mediante programación basada en reglas. La caracterización de los documentos se ha basado en la frecuencia de aparición de palabras.

Un factor que ha demostrado ser altamente determinante es la cantidad y calidad de los datos. El simple hecho de pasar de un *dataset* con 676 documentos a otro con 2271 documentos propició un aumento del rendimiento de un 16% (extracción basada en nombres, tabla 1); incrementando desde un 44% (donde los modelos no tenían capacidad predictiva) hasta un 60%. El resto de metodologías y consideraciones han conseguido aportar un 12% adicional al rendimiento del sistema. Los mencionamos a continuación:

- **Uso de nombres como atributos:** el rendimiento de los clasificadores ha sido mayor al utilizar nombres frente a verbos, ya que estos son más diversos y por tanto más específicos a la hora de describir un perfil laboral.
- **Sinonimia:** el uso de la sinonimia no ha mejorado los resultados. No obstante, se mantiene su uso en la fase de servicio para ayudar a captar mejor los distintos estilos o formas de expresión que pueda utilizar el usuario.
- **Reducción de dimensionalidad:** en vez de realizar la selección de características sobre el total de documentos, esta se acomete categoría a categoría. Esta metodología ayuda a seleccionar palabras más descriptivas de cada categoría, que habían sido desplazadas (en cuanto a frecuencia de aparición) por palabras más comunes cuando la selección se aplicaba sobre el total de documentos. Por encima del leve aumento del rendimiento (entorno al 3%), destaca que se consigue bajar notablemente el número de características necesarias para catalogar los documentos. Con esto se consigue mejorar la representatividad de las características obtenidas y aumentar la eficiencia del modelo que ahora procesará una cantidad menor de datos (en entrenamiento e inferencia).

- **Eliminación de categorías con pocos ejemplos:** tras analizar el rendimiento individual para la predicción de cada categoría se observaron resultados muy poco precisos para los perfiles laborales de los que se disponían de pocas muestras. Tras eliminar dichas categorías el rendimiento llega a alcanzar un 72%, resaltando así la importancia de tener un número abundante de muestras.
- **Cribado de características genéricas:** se detectó que el hecho de que algunas categorías a pesar de disponer de suficientes muestras no consiguieran altos rendimientos en la clasificación se debía a que compartían gran parte del conjunto de características que describen dicha clase con alguna otra clase. Para reducir el acoplamiento se trató de eliminar del conjunto de características aquellas que fueran compartidas con frecuencia entre varias categorías. Esta técnica no mejoró el rendimiento.
- **Matriz tf-idf:** como alternativa al cribado de características genéricas (reducción de la información), se probó a dar más peso a las palabras más específicas de cada perfil laboral mediante la construcción de la matriz tf-idf como forma de caracterizar los datos. Los rendimientos se mantuvieron en los mismos niveles.
- **Eliminación de las categorías acopladas:** dejando fuera de la clasificación las categorías que se acoplan podemos fácilmente obtener un rendimiento del 85%, que puede ser mejorado si se sigue reduciendo aún más el número de categorías.
- **Word vector:** la utilización de representaciones densas para caracterizar los datos permite entrenar redes neuronales que alcanzan rendimientos similares a las técnicas basadas en representaciones dispersas (frecuencias). No obstante, este tipo de técnicas no son tan interpretables como las anteriores, razón por la que se optó por las primeras para el desarrollo del trabajo.

Se concluye por tanto, que los dos principales factores limitantes para alcanzar rendimientos más elevados han sido la escasez de datos (reduce capacidad de predicción) y el acoplamiento que presentan algunos perfiles laborales que son relativamente parecidos (reduce la precisión).

B. Comparativa de algoritmos de aprendizaje

Tal y como ya avanzaban algunos autores [26], se ha corroborado que los clasificadores bayesianos aportan mejores rendimientos que las redes neuronales en problemas de clasificación de texto. Es necesario anotar que para los procesos de entrenamiento supervisado es primordial disponer de un correcto etiquetado de las muestras. Dado que los algoritmos de agrupamiento (*clustering*) no han conseguido aportar etiquetados suficientemente consistentes, se optó por un etiquetado semi-guiado durante el propio proceso de adquisición de datos (*web scraping*). En este proceso se marca una clave de búsqueda para encontrar automáticamente un conjunto de descripciones laborales pertenecientes al mismo grupo.

Resulta interesante comentar que aunque en este trabajo se ha tenido que buscar la forma de etiquetar automáticamente las muestras, la mayoría de portales web para búsqueda de empleo ya disponen de bases de datos de ofertas laborales con un número abundante de muestras correctamente etiquetadas. Por tanto, les resultaría bastante directo adaptar la información y conseguir conjuntos de entrenamiento que aporten rendimientos elevados.

C. Agente conversacional como interfaz de usuario

El agente conversacional consigue guiar de forma efectiva el proceso de extracción de información a la vez que consigue mejorar la usabilidad del sistema y la experiencia del usuario al naturalizar la comunicación. La combinación de reglas de producción (*intents*) y contextos han demostrado ser técnicas bastante flexibles para guiar el flujo de la conversación. Por su parte, la utilización de técnicas de reconocimiento de entidades resulta bastante efectiva para localizar y extraer la información que se precisa del usuario.

A. Conclusiones

En líneas generales se puede concluir que la utilización de agentes conversacionales como interfaz con el usuario aporta una notable fluidez al proceso de búsqueda de empleo. El usuario conseguirá encontrar ofertas de empleo bien segmentadas según sus preferencias en apenas unos minutos en el marco de una conversación amena y enfocada a las necesidades de la tarea, asegurando que el usuario no olvida aportar información que resulte útil para acotar los resultados de la búsqueda. En vez de tener que rellenar formularios que pueden ser incompletos y/o ambiguos, el usuario se dedica a comunicarle al agente sus habilidades y competencias de la forma más natural posible, mediante el lenguaje. De esta forma, se consigue dar una gran visibilidad al producto y predisponer al usuario a utilizar este tipo de soluciones. Esto a su vez, abre la puerta para pensar en este tipo de interfaces como un enfoque preferente al planificar desarrollos de herramientas que impliquen intercambio directo de información con el usuario.

En cuanto a la implementación, ha quedado demostrado que es completamente viable la inserción de técnicas de procesamiento del lenguaje natural y de aprendizaje automático para resolver las tareas asociadas a cualquiera de las fases del ciclo de desarrollo y puesta en producción de la herramienta. Concretamente, se han aplicado técnicas de procesamiento del lenguaje natural para la extracción y selección de características del texto de entrada, algoritmos de aprendizaje (inferencia) para la clasificación del perfil laboral y una mezcla de ambos enfoques para la construcción del agente conversacional que actúa como interfaz. Enlazado con este aspecto, resulta interesante resaltar como la solución desarrollada combina desarrollos propios con desarrollos de terceros (comerciales o de la comunidad de investigación). Los módulos disponibles a nivel comercial resuelven de forma cerrada problemas genéricos tales como el análisis sintáctico, reconocimiento de entidades, proporcionan arquitecturas y funcionalidades (*frameworks*) para el desarrollo de agentes conversacionales, etc. Las particularidades o necesidades específicas de cada problema deberán ser resueltas mediante adaptaciones, re-entrenamiento de modelos o desarrollos propios. Por ello, las empresas de soluciones comerciales han puesto especial atención en habilitar APIs para que los desarrolladores puedan adaptar o extender dichas herramientas de acuerdo a las necesidades concretas de sus proyectos. Esta flexibilidad permite reducir los tiempos de desarrollo y despliegue a la vez que asegura la obtención de altos rendimientos al trabajar sobre módulos que han sido desarrollados y testeados de forma muy rigurosa por equipos altamente cualificados.

B. Trabajo futuro

Con respecto a la interfaz de usuario, una ampliación bastante directa sería añadir nuevos roles al agente conversacional que permitan al usuario el uso de funcionalidades adicionales como configurar notificaciones cuando aparezcan nuevas ofertas de trabajo dado un perfil, asistencia para la elaboración de currículums bien organizados y completos, entrevistas de trabajo virtuales como preparación para entrevistas reales, etc. En estos casos el agente podría trabajar sobre distintos dominios de aplicación, por lo que en primer lugar tendría que detectar la tarea a realizar para posteriormente cargar la plantilla (*frame*) y activar las reglas de producción (*intents*) adecuadas para cada caso. En un esfuerzo algo más ambicioso se puede entrever que la mayor parte de las funcionalidades de las aplicaciones web podrían implementar su interfaz con el usuario de forma amena mediante el uso de agentes conversacionales, en especial cuando se conversa por voz. La idea subyacente es que cada funcionalidad tenga una API bien definida, completa y flexible y que el agente conversacional se dedique a rellenar los parámetros e información de entrada para la API.

Atendiendo a la clasificación de categorías laborales, se debe profundizar en la investigación de distintas alternativas que permitan desacoplar mejor aquellos perfiles laborales que sean similares. Un posible enfoque podría consistir en agrupar las muestras en categorías más genéricas y dejar que sea el usuario el que a través de sus preferencias decida si desea un trabajo más o menos específico; para por ejemplo distinguir entre los perfiles de ‘ingeniero’ y ‘técnico’. Esta metodología sería solo posible si al etiquetar se añadieran dos atributos, la categoría principal y una categoría secundaria. Disponiendo de dos categorías, quizás podría dar mejor resultado trabajar con clasificadores en serie, de tal forma que uno de ellos se encargara de distinguir los perfiles más genéricos y a continuación un clasificador especializado (para cada perfil) tratara de distinguir las posibles subcategorías. Al ser la segunda etapa de clasificación mucho más específica, es probable que se pudiera conseguir desacoplar mejor las categorías similares. Por otra parte, también se plantea investigar los resultados de utilizar otras técnicas de clasificación de texto, como LDA [33]. Aunque debido a que el algoritmo también se basa en frecuencia de palabras, es probable que el acoplamiento entre perfiles laborales propicie que la técnica no acabe siendo del todo efectiva. Queda abierta también la posibilidad de encontrar la forma de dar más o menos peso a los datos para aumentar la diferencia entre categorías similares. Algoritmos como los propuestos por Lesk [38] podrían ser utilizados a tal efecto. En último lugar, se propone como posible vía de investigación combinar las técnicas de NLP con algoritmos basados en *word vectors* para tratar de encontrar alguna combinación que consiga captar mejor la semántica de los textos a la hora de ser clasificados.

REFERENCIAS

- [1] Lombardero, L. (2015), Trabajar en la era digital: tecnología y competencias para la transformación digital. Lid Editorial
- [2] Neller, T. W. (2017), *AI education: Machine learning resources*. AI Matters, 3(2)
- [3] Jain, A. K. (2010), Data clustering: 50 years beyond K-means. *Pattern recognition letters*. 31(8), 651-666
- [4] Buchta, C., Kober, M., Feinerer, I., & Hornik, K. (2012), *Spherical k-means clustering*. *Journal of Statistical Software*, 50(10), 1-22
- [5] Blumberg, R., & Atre, S. (2003), *The problem with unstructured data*. *Dm Review*, 13(42-49), 62
- [6] Buneman, P., Davidson, S., Fernandez, M., & Suciu, D. (1997, January), *Adding structure to unstructured data*. In *International Conference on Database Theory* (pp. 336-350). Springer, Berlin, Heidelberg
- [7] Miller, G. A. (1995), *WordNet: a lexical database for English*. *Communications of the ACM*, 38(11), 39-41
- [8] Martin, J. H., & Jurafsky, D. (2009), *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River: Pearson/Prentice Hall.
- [9] Bird, S., Klein, E., & Loper, E. (2009), *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc..
- [10] Peres-Neto, P. R., Jackson, D. A., & Somers, K. M. (2005), *How many principal components? Stopping rules for determining the number of non-trivial axes revisited*. *Computational Statistics & Data Analysis*, 49(4), 974-997.
- [11] Crespo, R. G., Martínez, O. S., Lovelle, J. M. C., García-Bustelo, B. C. P., Gayo, J. E. L., & De Pablos, P. O. (2011), *Recommendation system based on user interaction data applied to intelligent electronic books*. *Computers in Human Behavior*, 27(4), 1445-1449.
- [12] Belkin, N. J., & Croft, W. B. (1992), Information filtering and information retrieval: two sides of the same coin. In *Communications of the ACM*.
- [13] Resnick, P., & Varian, H. R. (1997), *Recommender systems*. *Communications of the ACM*, 40(3), 56-59.
- [14] Adomavicius, G., & Tuzhilin, A. (2005), *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. *IEEE Transactions on Knowledge & Data Engineering*, (6), 734-749.
- [15] Linden, G., Smith, B., & York, J. (2003), *Amazon.com recommendations: Item-to-item collaborative filtering*. *IEEE Internet computing*, (1), 76-80.
- [16] Hofmann, T. (2003, July), *Collaborative filtering via gaussian probabilistic latent semantic analysis*. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval* (pp. 259-266). ACM.
- [17] Grosz, B. J., & Sidner, C. L. (1986), *Attention, intentions, and the structure of discourse*. *Computational linguistics*, 12(3), 175-204.
- [18] Chomsky, N. (1956), *Three models for the description of language*. *IRE Transactions on information theory*, 2(3), 113-124.
- [19] Padró, L., Collado, M., Reese, S., Lloberes, M., & Castellón, I. (2010), *Freeling 2.1: Five years of open-source language processing tools*. In *7th International Conference on Language Resources and Evaluation*.
- [20] Baeza-Yates, R. (2004, February), *Challenges in the interaction of information retrieval and natural language processing*. In *International Conference on Intelligent Text Processing and Computational Linguistics* (pp. 445-456). Springer, Berlin, Heidelberg.
- [21] Russell, S. J., & Norvig, P. (2016), *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.
- [22] Fortnow, L., & Homer, S. (2003), *A short history of computational complexity*. Boston University Computer Science Department.
- [23] Goodfellow, I., Bengio, Y., & Courville, A. (2016), *Deep learning*. MIT press.
- [24] Sutton, R. S., & Barto, A. G. (2011), *Reinforcement learning: An introduction*. MIT press.
- [25] Fiuza Pérez, M., & Rodríguez Pérez, J. C. (2000), *La regresión logística: una herramienta versátil*. *Nefrología*, 20(6), 495-500.
- [26] Yang, Y., & Liu, X. (1999, August), *A re-examination of text categorization methods*. In *Sigir* (Vol. 99, No. 8, p. 99).
- [27] Hossin, M., & Sulaiman, M. N. (2015), *A review on evaluation metrics for data classification evaluations*. *International Journal of Data Mining & Knowledge Management Process*, 5(2), 1.
- [28] Sagirolgu, S., & Sinanc, D. (2013, May), *Big data: A review*. In *2013 International Conference on Collaboration Technologies and Systems (CTS)*, (pp. 42-47). IEEE.
- [29] Vargiu, E., & Urru, M. (2013), Exploiting web scraping in a collaborative filtering-based approach to web advertising. *Artif. Intell. Research*, 2(1), 44-54.
- [30] Myllymaki, J. (2002), Effective web data extraction with standard XML technologies. *Computer Networks*, 39(5), 635-644.
- [31] Moreno, I., Moreda, P., & Romá-Ferri, M. (2012), *Reconocimiento de entidades nombradas en dominios restringidos*. In *Actas del III Workshop en Tecnologías de la Informática* (pp. 41-57).
- [32] Robertson, S. (2004), Understanding inverse document frequency: on theoretical arguments for IDF. *Journal of documentation*, 60(5), 503-520.
- [33] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003), *Latent dirichlet allocation*. *Journal of machine Learning research*, 3(Jan), 993-1022.
- [34] Beel, J., Breitinger, C., & Langer, S. (2017), Evaluating the CC-IDF citation-weighting scheme: how effectively can ‘Inverse Document Frequency’(IDF) be applied to references. *Proceedings of the 12th iConference*.
- [35] Turing, A. M. (2004), *Computing machinery and intelligence (1950)*, *The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Ed. B. Jack Copeland. Oxford: Oxford UP, 433-64.
- [36] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. (1977), *GUS, a frame-driven dialog system*. *Artificial intelligence*, 8(2), 155-173.
- [37] Reiter, E., & Dale, R. (2000), *Building natural language generation systems*. Cambridge university press.
- [38] Banerjee, S., & Pedersen, T. (2002, February), *An adapted Lesk algorithm for word sense disambiguation using WordNet*. In *International conference on intelligent text processing and computational linguistics* (pp. 136-145). Springer, Berlin, Heidelberg.

Conversational agent for searching and recommending job offers

Héctor Gómez Loizaga

International University of La Rioja, Logroño (Spain)

Date: 19/09/2019



ABSTRACT

The purpose of this project is developing the base architecture and methodology for a conversational agent conceived for extracting the key information from an end user to whom the system is due to give orientation for open job positions to apply. Natural language processing is used for getting the features that are later feed to a machine learning algorithm. It is crucial to find well-balanced features able to accurately split the different job profiles. The inference motor is implemented as an automatic classifier. This classifier is first predicting the user job profile and then is presenting the user with suitable job bulletins, which are filtered by using information about skills and preferences of the user.

KEYWORDS

Conversational agent,
Feature extraction,
Machine learning,
Natural language processing

I. INTRODUCTION

THIS paper aims to investigate the feasibility of applying machine learning and natural language processing techniques to increase the level of automation and usability of job search portals.

It is intended to cover the complete development and commissioning cycle of an analytical tool, including the processing of data for the extraction of representative indicators of the process to be solved, the inference about the data to generate knowledge and assistance in decision-making (recommendation).

Natural language processing techniques have been used to accomplish two objectives: the extraction of relevant information on unstructured data (text) and the speeding up and naturalization of communication with the user. On the other hand, machine learning techniques have been the basis for the construction of a classifier that assigns the work profile to the user.

The tool developed consists of a conversational agent that acts as an interface for the user. This agent collects the information needed by the inference module (classifier) and uses the results generated by it to suggest to the user the set of job offers that best suit their background and preferences. With this tool the user is able to effectively find the information he was looking for in a pleasant and concise conversation, significantly improving the user experience of this type of systems.

II. STATE OF THE ART

In this section, the technological fields that serve as basis for the development of this work are documented.

A. Natural language processing (NLP)

Natural Language Processing (NLP) consists of a series of techniques focused on the treatment and analysis of natural language. Its main objective is that computers can carry out tasks where natural or human language participates [8]. It is undoubtedly a highly

interdisciplinary field that encompasses a wide range of tasks and the associated knowledge required to undertake them: signal processing (electronic and telecommunications engineering), syntactic analysis and semantic interpretation of words and phrases (computer science), morphology, phonology and pragmatics (linguistics), study of cognitive mechanisms for language acquisition and how society influences the use of language (sociologists and psychologists).

Natural language processing applications are distinguished from other data processing systems by requiring the use of language knowledge. Such knowledge will be necessary for the development of systems that are capable of performing different types of tasks such as morphological analysis, syntactic analysis, use of grammars, semantic analysis and pragmatic language analysis..

Regardless of the diversity of languages and linguistic systems specific to each language, a common aspect that makes computing difficult is ambiguity, understood as situations in which information can be understood or interpreted in more ways than one..

1) Representation of meaning

The linguistic resources (dictionaries, lexicon, thesauri, databases of lexical relations, linguistic corpus) serve as the basis for the creation of the different strategies of representation of the meaning of the words. Among the most used are the following:

- **Relational databases:** the words appear grouped by meaning and the semantic relationships that occur between them. Wordnet [7], is a representative example of this type of meaning representation strategies. Although this type of resources works quite well, nuances about the context are often lost.
- **Discrete representation:** based on a specific vocabulary, the sentences or texts are represented by a vector where each element indicates the number of times each word of the vocabulary appears in said fragment of the document (term frequency). Referenced as bag-of-words, this type of representation does not evaluate the order of appearance of words (context).
- **Distributed representation:** it consists of the representation of

a word by means of a dense vector. Known as word vectors, word embeddings or word representations, they constitute a manageable representation (computationally) to express semantic information about similarity between words. The vectors are extracted through statistical procedures supported by machine learning models (neural networks) that work on linguistic corpus.

2) Language models

Mainly, the problem of mathematical modeling for language analysis and treatment is approached from two approaches.

- **Logic:** linguistic experts establish rules to recognize structural patterns and reflect the logical structure of language. They make use of formal languages that allow an unequivocal representation of objects, their properties and the relationships they have with each other. In addition, being based on rules allows new knowledge to be produced through inference mechanisms. They are based on studies published by Chomsky in the fifties.[18].
- **Probabilistic:** based on the analysis of data collections that make up a linguistic corpus. The frequencies of the linguistic units are calculated and machine learning techniques are applied. The construction of these models is mainly carried out through supervised training techniques, although the use of unsupervised training is also being investigated in order to reduce the effort of having to label the huge corpus on which you train.

3) Applications and tools for PLN

The interdisciplinary of the field implies implicitly that there is a great variety of applications based on natural language processing techniques. Among them we can highlight the development of conversational agents (chatbots), automatic translation and search for answers (Question Answering, QA).

Among the numerous tools and libraries focused on the development of language processing applications [19] we highlight the Natural Language Toolkit and the Stanford CoreNLP.

On the other hand, it is worth highlighting the interest shown by the large technology development companies (Google Cloud Natural Language, Amazon Comprehend, IBM Watson Natural Language Understanding, Microsoft Language Understanding) in capturing the market with their own commercial solutions, which offer complete services, scalable and easy to integrate through the available APIs and which are normally distributed from their cloud platforms.

B. Automatic feature extraction

Feature extraction is a coding and reduction process that aims to reduce a set of data to features or attributes that accurately capture the information contained in the data and that are treatable by computer programs. Listed below are some of the most commonly used methods for extracting features on text documents.

1) Unstructured data

One of the difficulties of working on text documents is the fact that it is data that does not incorporate an implicit structure and therefore requires additional processing to add information or structure to the starting data, so that they can be properly managed by inference engines [5] [6]. Through the use of Big Data tools and techniques [28], such as Hadoop or non-relational databases such as NoSQL, such information can be converted into relevant information. In the case of text documents, the use of natural language processing techniques will become almost essential.

2) Web scraping

Data tracking is a technique that allows you to automatically browse the network and extract information from the web portals you visit. The extraction of information is based on the knowledge of the HTML tags with which the different sections and the content on the

web pages are structured. There are several tools that allow you to create, without much effort, software that performs this function and that receives among others the denominations spider, bot or crawler. Python is undoubtedly the language mostly used to track through the implementations of several of its libraries: Scrapy, BeautifulSoup and Selenium.

3) Entity recognition (NER)

Entity recognition (Named Entity Recognition), is a technique that allows you to locate and identify all textual references to a series of predefined entities. These entities could be organizations, people, locations, dates, facilities, etc. Any instance of these entities will be located and labeled together with their corresponding entity [31]. The available implementations are based on the use of machine learning algorithms, grammar and linguistic corpus. This type of tools plays a very interesting role in information retrieval tasks (information retrieval), since it allows to obtain key-value pairs directly, thus avoiding having to collect large amounts of text to which expressions would have to be applied later regular.

4) Text classification

As a previous step to the text classification tasks we will need to obtain descriptors of the information contained in the text that is manageable by the algorithms.

- **Tf-idf (Term frequency - inverse document frequency):** quantifier used to weigh the relevance of a word in a document that is also part of a collection of documents. The assigned value increases the higher the frequency of the word in a given document and decreases the higher the frequency of that word in the global documents. In this way it is about penalizing generic words that are repeated numerously in the collection of documents. Its mathematical formula is:

$$tf\ idf(t, d, D) = tf(t, d) \cdot idf(t, D)$$

where $tf(t, d)$ is the frequency of occurrence of the term t (word) in the document d by $idf(t, D) = \log(|D| / |\{d \in D: t \in d\}|)$, with D the total number of documents and $1 + |\{d \in D: t \in d\}|$ the number of documents where the term t appears, to which the unit is added to avoid division by zero if the term does not appear in the collection [32].

- **Latent Dirichlet Allocation (LDA):** technique designed for the detection of topics (categories) in text documents. This method tries to model the documents as a finite set of topics assuming that they follow a Dirichlet distribution. In turn, each topic is assumed to be described by a word distribution [33].

C. Machine learning (ML)

The discipline of machine learning encompasses a series of methods and algorithms oriented towards the extraction of knowledge autonomously from data. The data represent the examples that the algorithms process to adjust their internal parameters. In this way, during the training process (adjustment) as a larger number of samples are processed, the algorithm acquires greater knowledge through experience [21]. In turn, the training corresponds mathematically with the resolution of optimization problems, where the processing of each sample serves the algorithm to better adjust its parameters seeking to minimize some metric or measure of the error made when making inferences.

We can mainly identify two requirements on training data:

- **Abundant data set:** so that the algorithms have the opportunity to cover the diversity of the total population that is intended to classify or show the dynamics of the system to be modeled throughout its range of action.
- **Representative variables:** the selected variables (subset of the input data) must be sufficiently predictive of the system to be modeled, that is, they must be those variables that best describe and / or differentiate the system responses.

1) Types of machine learning algorithms

We can classify the different types of machine learning algorithms based on the techniques for their implementation and the training methodology used:

- **Supervised learning:** a set of data labeled with the associated response is available. During the training, in successive iterations each sample is processed and compared with the expected output. The amount of error made in the prediction is evaluated through some metric and is used to readjust the internal parameters of the system.
 - o Applicable to classification problems (assign one among n classes to each sample) and regression (prediction of a numerical value).
- **Unsupervised learning:** in this case there is no expected output or response, but the internal structure of the data is determined. These techniques work with similarity metrics between sample feature vectors.
 - o Applicable to clustering problems (grouping very similar instances into categories) and anomaly detection (detecting very unusual values).
- **Semisupervised learning:** this type of algorithm combines learning based on labels and classification based on the internal structure of the data.
- **Reinforcement learning:** reinforcement learning algorithms are aimed at obtaining knowledge of the environment with which it interacts and allow consideration of the future consequences of taking an action at the time of making the decision [24]. Mathematically they are modeled through Markov's decision processes.
 - o Applicable to the implementation of intelligent agents.

2) Machine learning models

There are multiple implementations depending on the type of problem and the objective to be covered. Each model differs in the way they approach the learning process by finding analytical (logistic regression), geometric (Support Vector Machines), structural (decision trees and Random Forest), probabilistic (Bayesian networks) and connectionist (networks) approaches neural, among others.

D. Conversational agents

Conversational agents are systems designed to communicate with users through the use of natural language. Although the interaction is usually established mainly through voice messages (audio) or through writing (text), other communication modalities can also be used integrating graphics, images or animations, which can contribute to the construction of environments highly interactive. Conversational agents are employed in fields as diverse as entertainment, education, psychology, languages, sales, marketing, etc. Among its main applications we can highlight its use in customer service systems as well as in the assistance to the purchase process, where they are presented as an interface for the recommendation engine.

1) Types of conversational agents

Based on the type of conversation they are able to maintain and the objective with which they have been implemented, we distinguish between:

- **Dedicated conversational agents:** designed for the development of a specific task and therefore restricted to an application domain. Generally, this implies that they are focused on solving simple tasks, for which it is sufficient to keep a short conversation that requires little interaction with the user. In this category we find personal assistants such as Alexa (Amazon), Siri (Apple), Google Assistant, etc.
- **Chatbots:** They allow to initiate extensive conversations with multiple interactions, based on a dialogue that is not necessarily structured and that can deal with any subject.

2) Arquitectura de los agentes conversacionales

A conversational agent will present the following modules [8]:

- **Speech recognition:** generation of text (words) from an acoustic signal. It uses acoustic-phonetic modeling techniques, finite state grammars and probabilistic models.
- **Understanding natural language:** implements natural language processing tasks focused on extracting the (semantic) meaning of sentences. Based on templates (frames [36]) with attribute-value pairs that the agent fills after examining the input text.
- **Dialogue management:** determines the action to be performed and / or the concept to be transmitted in the current word shift in response to the request made by the user. You need to maintain the state and flow of the conversation and request information from the user when it is necessary to generate the response. The flow of dialogue is usually managed using finite state machines and production rules.
- **Generation of natural language:** choose the concepts to be transmitted and plan how to express them [37]. We first proceed to the generation of a lexicalized phrase (with gaps) that is then filled in with values that will be inferred based on the information extracted from the user.
- **Text to speech conversion:** synthesizes the message to be transmitted in acoustic format. You can use formant synthesis techniques, articulation synthesis and concatenation-based synthesis (more natural result).

3) Dialogflow

Dialogflow is Google's product development platform aimed at facilitating human-computer interaction based on machine learning and natural language processing techniques. It consists of a series of elements and functionalities that allow the creation, configuration and training of conversational agents in a comfortable and intuitive way:

- **Intention (intents):** an intent is used to relate a series of messages (phrases) that the user can issue with a task that the agent is able to solve. The intents react to a set of phrases that are assigned a series of responses (static, dynamic or generated by external applications) during agent training. In turn, *Dialogflow* allows conditional activation of the intents to certain contexts being active. Therefore, the contexts serve to guide the flow of the conversation and to disambiguate the meaning of the sentences.
- **Entities:** *Dialogflow* relies on NER to solve the attribute-value pairs of the templates (intents) that it applies to the input texts.
- **Fullfillment:** API that enables communication between *Dialogflow* and any external application through the use of *webhooks*. It allows generating the agent's responses through the use of inference engines or external production rules.
- **Integrations:** allows you to quickly integrate the conversational agent with the most popular messaging services, attendees and chat applications: Facebook Messenger, Slack, Twitter, Kik, Line, Skype, Telegram, Twilio and Viber, Google Assistant, Amazon Alexa, Microsoft Cortana, etc

III. OBJECTIVES AND METHODOLOGY

A. Overall objective

The main objective of this paper is to investigate the feasibility of using natural language processing and machine learning techniques to naturalize and assist the job search process through digital media. Specifically, consist of building a job search tool with an interface implemented through a conversational agent that identifies the user's work profile and filters the database of job offers based on the assigned profile and user preferences.

B. Specific objectives

The specific objectives have been selected to break down the main

objective into bounded tasks that can be planned and addressed more efficiently. Below, the goals that are intended to cover are listed:

- Feature extraction on unstructured data. This type of data usually assumes a high percentage of the amount of data generated by companies (up to 80%) [5] [6].
- Compare the performance of different automatic classification techniques and evaluate the prerequisites to be applied.
- Use filtering and recommendation techniques to refine the classification of categories and to help obtain more accurate results.
- Guide the information extraction process and naturalize communication with the user through the use of conversational agents as the interface to the system.

C. Methodology

The project will started by going to data repositories and selecting a dataset that suits the needs of the job. After evaluating its representativeness, the possibility of expanding the data set through web scraping techniques will be considered.

The next step will consist of extracting features from our unstructured data source. For this task, natural language processing techniques will be used via morphosyntactic analyzers. Later on, filtering will be applied by the use of grammars to extract grammatical constituents that may be more predictive of our data set.



Figure 1. Natural language processing pipeline

The degree of representativeness obtained from the use of noun phrases, verbal phrases or the combination of both will be compared. The frequency of occurrence of the grammatical constituents obtained will be counted and the most frequent ones (frequency matrix) will be selected as input characteristics for the inference engine. Different approaches to refine the feature selection process will be also tested to make them as representative as possible and try to reduce the overlapping between similar categories. The synonymy will be used to cover the different variants that the user can use to transmit the content.

The information extracted from the description of each profession will be used to feed the machine learning algorithms (inference engines) responsible for cataloging the user's work profile. For this purpose, the results of two types of machine learning models will be used and compared: Bayesian networks and neural networks. The first ones have been selected as the preferred technique due to their proven efficacy and widespread use for text classification [26]. The seconds will be used because they are one of the most powerful and flexible models. We will reserve 80% of the data for the training of the models and 20% for the evaluation of them.

The Bayesian models will be evaluated using the *f1* metric, based on the confusion matrix [27]:

$$precision = \frac{VP}{VP+FP} \quad recall = \frac{VP}{VP+FN}$$

$$f1 = \frac{2 \cdot precision \cdot recall}{precision + recall}$$

On the other hand, the neural networks classification performance will be evaluated by the loss function using during training (optimization). The function chosen for this purpose is *Categorical Cross-Entropy*:

$$CE = -\log\left(\frac{e^{S_p}}{\sum_j e^{S_j}}\right)$$

To avoid presenting abundant and unnecessary information to the user, a filter according to the assigned work profile and additional information extracted from the user (such as the salary expected

range, the city in which the user would like to work, etc.) will be applied. To obtain this data, entity recognition (*NER*) will be used on the side of the conversational agent and regular expressions on the side of the filter engine in *Python*.

The conversational agent will be presented as the entry and output point the user. We will focus on defining the dialogue management strategy and verifying that the user has provided the required information with the use of finite state machines and frame-based information retrieval techniques. Strategies to achieve greater variability in the formulation of the questions will be also inspected, with the purpose of bringing naturalness to the conversation.

IV. CONTRIBUTION

The objective of this work is to undertake the complete development cycle of an application based on artificial intelligence techniques. The developed system has been divided into four modules according to the task to be completed..

A. Feature Extraction module

- Target: treatment of unstructured data to obtain predictive indicators for the problem to be solved.
- Techniques: natural language processing.
- Tools: NLTK by *Python*.
- Inputs: text with description of tasks to be performed in each job.
- Outputs: frequency of relevant words in each text.

B. Inference module

- Target: assignment of the most likely job profile.
- Techniques: classification by machine learning.
- Herramientas: *Keras* and *Scikit-Learn* by *Python*.
- Inputs: frequency of relevant words in each text.
- Outputs: categoría laboral.

C. Filter and recommendation module

- Target: selection in the database of offers relevant to the user.
- Techniques: filtering with search keys.
- Herramientas: *Python*.
- Inputs: assigned job category and user preferences.
- Outputs: set of job offers.

D. Conversational interface module

- Target (1): guide the user to ensure that he provides accurate and complete information to perform the search.
- Techniques: frame-based conversation model and dialogue management using finite state machines.
- Tools: *Dialogflow*.
- Target (2): processing unstructured data to obtain user preferences.
- Techniques: regular expressions and entity recognition.
- Tools: *Python* y *Dialogflow*.
- Inputs: user text.
- Outputs:
 - Módulo de inferencia: text entered by the user.
 - Módulo de filtrado y recomendación: user preferences.
 - Usuario: question to request information, informative text and set of job offers found.

V. RESULTS OR EVALUATION

A. Data pre-processing

Originally, the dataset we used the "Data Science for Good: City of Los Angeles" (from now DS_O) published on the Kaggle web platform [2]. When verifying that the dataset was not representative enough, another dataset was obtained using web scrapping techniques (DS_E). The documents contain different sections in text format. The

pre-processing stage collects the relevant information for the classification algorithms: name of the job offer and the text of the 'Duties', 'Salary' sections, which contain the description of the tasks performed in the position offered and the salary range that can be accessed respectively.

B. Dataset labeling

The application of clustering techniques to obtain automatic labeling of work bulletins provides inconsistent results (figure 2).

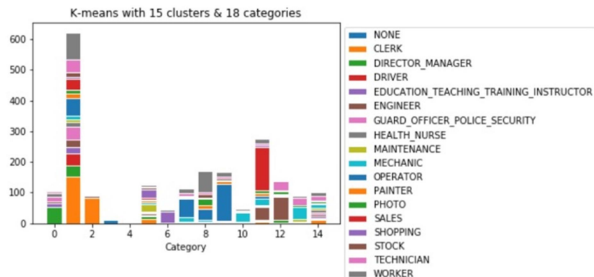


Figure 2. Clustering for automatic labeling

As an alternative, semi-guided labeling was used that led to the definition of 18 categories:

{'NONE': 0, 'CLERK': 1, 'COLOR_PAINTER': 2, 'DIRECTOR_MANAGER': 3, 'DRIVER': 4, 'EDUCATION_TEACHING_TRAINING_INSTRUCTOR': 5, 'ENGINEER': 6, 'GUARD_OFFICER_POLICE_SECURITY': 7, 'HEALTH_NURSE': 8, 'MAINTENANCE': 9, 'MECHANIC': 10, 'OPERATOR': 11, 'PHOTO': 12, 'PRINTER': 13, 'SALES': 14, 'SHOPPING': 15, 'STOCK': 16, 'TECHNICIAN': 17, 'WORKER': 18}

C. Feature selection and classifier performance

The yields obtained by applying different methodologies for the extraction and selection of characteristics are shown.

1) Comparison of datasets and attribute types

Performance between the original (DS_O) and extended (DS_E) dataset for extracting features (features) based on verbs (V) and names (N) is compared.

Dataset	Features	Categories	Test performance	Models
DS_O	V: 44	Hand:22	0.1863	NB
DS_E	V: 283	Leaded:19	0.4879	NB
DS_O	N: 205	Hand:22	0.4412	NB
DS_E	N: 1027	Leaded:19	0.6066	NB

Table 1. Comparison of datasets and attributes when using verbs vs. names

From this moment, all tests are performed on the extended dataset (DS_E) using Bayesian models (NB).

2) Dimensionality Reduction

The results of carrying out the extraction and selection of characteristics on the total of the documents are compared against proceeding by evaluating the documents by groups of categories.

Dataset	Features	Minimum frequency	Features category	Test performance
Completo	N: 1027	20	Sin límite	0.6066
Grupos	N: 316	20	60	0.6352

Table 2. Comparing selection over complete dataset vs. groups

All the tests shown below apply the methodology for selecting characteristics by groups.

3) Characteristics representativeness improvement

The individual performance of each category is analyzed using the metrics obtained by the neural network algorithms (NN).

Category	precision	recall	f1-score	Samples
1	0.74	0.74	0.74	61
2	0.64	0.64	0.64	22
3	0.5	0.62	0.55	34
4	0.56	0.6	0.58	15
5	0.58	0.55	0.56	20
6	0.64	0.55	0.59	51
7	0.72	0.5	0.59	26
8	0.25	0.25	0.25	8
9	0.2	0.08	0.11	13
10	0.44	0.55	0.49	22
11	0.65	0.67	0.66	91
12	0.41	0.41	0.41	17
13	0.72	0.5	0.59	36
14	0.73	0.72	0.73	46
15	0.4	0.4	0.4	10
16	0.12	0.12	0.12	8
17	0.22	0.19	0.2	32
18	0.46	0.66	0.54	56

Table 3. Performance by categories.

The marked rows correspond to categories that have few samples or that share common (coupled) characteristics: *Health_Nurse* (48), *Maintenance* (55), *Mechanic* (142), *Photo* (75), *Shopping* (44), *Stock* (46), *Technician* (154).

First, classes with few samples are eliminated. Then, to try to decouple the similar categories, we have eliminated the characteristics that are frequent and common among several categories. Another approach has been to use the tf-idf matrix. The results obtained by eliminating the categories that are coupled are also added.

Methodology	Features	Categories	Test performance
Sample minimum	N: 298	13	0.7103
Features screening	N: 291	13	0.7051
tf-idf matrix	N: 298	13	0.6974
Eliminating coupled	N: 242	9	0.8513

Table 4. Approaches for improving the representativeness.

4) Word vectors

Outputs of working over dense representations.

Features / vocabulary	Input	Categories	Test performance	Model
N: 298	<i>Tf</i>	13	0.7103	NB
298 (filtered)	<i>Word vectors</i>	13	0.5410	NN
11316 (complete)	<i>Word vectors</i>	13	0.7128	NN

Table 5. Result for models on dense representations.

D. Filtering and recommendation

The database on which the filtering algorithms work is shown below:

	Job_name	Profile	Salary_range	Id
10	AIRPORT CHIEF INFORMATION SECURITY OFFICER	GUARD_OFFICER_POLICE_SECURITY	(125.551, 183.576)	AIRPORT CHIEF INFORMATION SECURITY OFF
11	AIRPORT ENGINEER	ENGINEER	(105.005, 163.532)	AIRPORT ENGINEER 7256 0708
12	AIRPORT GUIDE	GUARD_OFFICER_POLICE_SECURITY	(31.925, 46.687)	AIRPORT GUIDE 0845 0420
13	AIRPORT INFORMATION SPECIALIST	DIRECTOR_MANAGER	(42.991, 78.529)	AIRPORT INFORMATION SPECIALIST 1783 1211
14	AIRPORT LABOR RELATIONS ADVOCATE	DIRECTOR_MANAGER	(93.918, 137.327)	AIRPORT LABOR RELATIONS ADVOCATE 0201
15	AIRPORT MANAGER	DIRECTOR_MANAGER	(99.326, 188.567)	AIRPORT MANAGER 7260 1202
16	AIRPORT POLICE CAPTAIN	GUARD_OFFICER_POLICE_SECURITY	(142.568, 197.001)	AIRPORT POLICE CAPTAIN 3228 0216
17	AIRPORT POLICE LIEUTENANT	GUARD_OFFICER_POLICE_SECURITY	(102.186, 134.028)	AIRPORT POLICE LIEUTENANT 3227 0918
18	AIRPORT POLICE OFFICER	GUARD_OFFICER_POLICE_SECURITY	(51.448, 83.019)	AIRPORT POLICE OFFICER 3225 11090 0901
19	AIRPORT POLICE SPECIALIST	GUARD_OFFICER_POLICE_SECURITY	(0.0, 0.0)	AIRPORT POLICE SPECIALIST 3236 063017

Figure 3. Database of job offers and preferences.

E. Conversational Agent

The following scheme represents the state machine that determines the active production rules (states) and the contexts (arcs) that condition the conversational agent's dialogue flow.

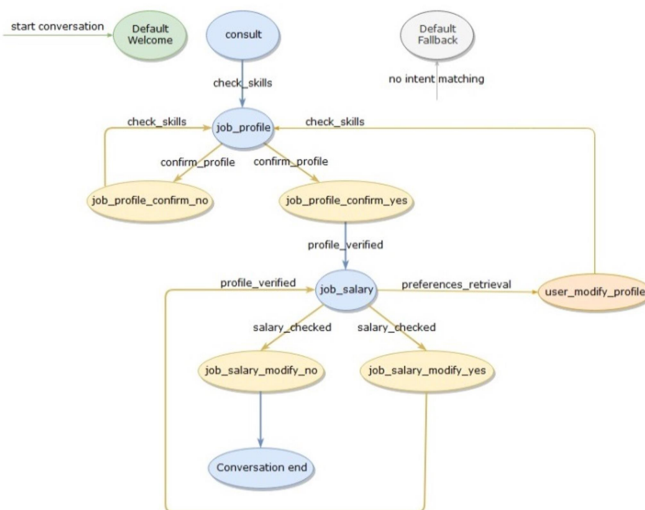


Figure 4. State machine for dialogue flow control.

VI. DISCUSSION O RESULTS ANALYSIS

On this section we will comment some reflections about the specific objectives of the project.

A. Feature extraction

The use of natural language processing techniques has been crucial for the extraction of relevant information from the input data (unstructured). It is here, where learning techniques show their power by capturing patterns and structures in the data; it would have been unfeasible to capture the complexity and casuistry that can occur in the input data through rule-based programming. The characterization of the documents has been based on the frequency words occurrence.

A factor that has proven to be highly determinant is the quantity and quality of the data. The simple fact of moving from one dataset with 676 documents to another with 2271 documents led to a 16% increase in yield (name-based extraction, table 1); increasing from 44% (where the models did not have predictive capacity) to 60%. The rest of the methodologies and considerations have managed to increase an additional 12% to the performance of the system. We mention them below:

- **Use of names as attributes:** the performance of the classifiers has been greater when using names rather than verbs, since these are more diverse and therefore more specific when describing a

work profile.

- **Synonymy:** the use of synonymy has not improved the result rates. However, its use is maintained in the service phase to help capture better the different styles of expression that the user can use.

- **Dimensionality reduction:** instead of making the selection of features over all the documents, this task has been undertaken category by category. This methodology helps to select more descriptive words from each category, which had been displaced (in terms of frequency of occurrence) by more common words when the selection was applied to the total number of documents. Above the slight performance increase (around 3%), it is worth noting that the number of features needed to catalog the documents is significantly reduced. This improves the representativeness of the characteristics obtained and increases the efficiency of the model that will now process a smaller amount of data (in training and inference).

- **Elimination of categories with few examples:** after analyzing the individual performance for the prediction of each category, low accurate results were observed for the labor profiles which had a small number of samples available. After eliminating these categories, the performance reaches 72%, thus highlighting the importance of having a generous number of samples.

- **Screening of generic characteristics:** despite having sufficient samples some categories did not achieve high yields in the classification. This was due to the fact that they shared a large set of the describing characteristics with some other class. To reduce the overlapping, we tried to eliminate those features that were frequently shared among several categories from the set of characteristics. This technique did not improve performance.

- **Matrix tf-idf:** as an alternative to the characteristics screening (reduction of information), it was tried to give more weight to the most specific words of each job profile by constructing the *tf-idf* matrix as a way to characterize the data. The yields remained at the same levels.

- **Elimination of the overlapped categories:** leaving out of the classification the categories that are coupled we can easily obtain a performance of 85%, which can be improved if the number of categories is further reduced.

- **Word vectors:** the use of dense representations to characterize the data allows training of neural networks that reach similar rates to those reached with techniques based on scattered representations (frequencies). However, these types of techniques are not as interpretable as the previous ones. This is the reason why the first ones were chosen for the development of the work.

As closure, it can be concluded that the two main limiting factors to achieve higher yields have been the data shortage (reduces predictability) and the overlapping that some relatively similar job profiles have (reduces accuracy).

B. Comparison of machine learning algorithms

As some authors already advanced [26], it has been corroborated that Bayesian classifiers provide better yields than neural networks in text classification problems. It is necessary to remark that for the supervised training processes it is essential to have a right labeling of the samples. Since clustering algorithms have failed to provide sufficiently consistent labeling, semi-guided labeling was chosen during the process of data acquisition (*web scraping*). In this process a key search is used to automatically find a set of job descriptions belonging to the same group.

It is also interesting to comment that although this work has had to find a way to automatically label the samples, most of the job search web portals already have job offers databases with an enough number of samples that are correctly labeled. Therefore, it would be quite straightforward to adapt the information and get training sets that

provide high outcomes.

C. Conversational agent as user interface

The conversational agent provides an effective guidance for the process of extracting information meanwhile is improving the usability of the system and the user experience by naturalizing communication. The combination of production rules (intents) and contexts have proven to be very flexible techniques to guide the flow of the conversation. On its part, the use of entity recognition techniques is utterly effective for locating and extracting the information that is required from the user.

VII. CONCLUSION

A. Conclusions

As overall, it can be concluded that the use of conversational agents as interface for the user makes the job search process remarkable easier. The user will be able to find job offers well segmented according to their preferences in just a few minutes in the context of a pleasant conversation which is focused on the needs of the task. This ensures that the user does not forget to provide information that is useful to narrow down the search results. Instead of having to fill out forms that may be incomplete and/or ambiguous, the user is dedicated to communicating to the agent their skills and competencies in the most natural way possible, through language. In this way, it is possible to give great visibility to the product and predispose the user to use this type of solutions. This, in turn opens the door to think of such interfaces as the preferred approach when planning tool developments that involve direct information exchange with the user.

Regarding the implementation, it has been demonstrated that is completely feasible to use natural language processing and machine learning techniques to solve the tasks associated with any of the phases of the development and production cycle of the tool. Specifically, natural language processing techniques have been applied for the extraction and selection of characteristics of the input text, learning algorithms (inference) for the classification of the job profile and a mixture of both approaches for the construction of the conversational agent that acts as interface. In relation with this point, it is interesting to highlight how the solution developed combines own developments with third party developments (commercial or research community). The commercially available modules solve in a closed way generic problems such as syntactic analysis, entity recognition, provide architectures and functionalities (frameworks) for the development of conversational agents, etc. The specific needs of each problem must be solved through adaptations, re-training of models or own developments. Therefore, commercial solutions companies have paid special attention to enabling APIs so that developers can adapt or extend these tools according to the specific needs of their projects. This flexibility allows the programmers to reduce development and deployment times while ensuring high robustness because they are using modules that have been developed and tested very rigorously by highly qualified teams.

B. Future work

Regarding the user interface, a direct extension would be to add new roles to the conversational agent that would allow the user to use additional functionalities such as setting up notifications when new job offers associated to a profile comes out, assistance in the preparation of well-organized and complete resumes, virtual job interviews for preparing real interviews, etc. In these cases the agent could work on different application domains, so first of all it would have to detect the task to be carried out to later load the template (frame) and activate the appropriate production rules (intents) for each case. In a more ambitious effort it can be seen that most of the functionalities of web applications could implement their interface

with the user in a pleasant way through the use of conversational agents, especially when conversing by voice. The underlying idea is that each functionality could have a well-defined, complete and flexible API and that the conversational agent is dedicated to filling in with the parameters and input information.

According to the classification of work categories, the investigation of different alternatives that would allow a better decoupling of those job profiles that are similar should be deepened. One possible approach could be to group the samples into more generic categories and let the user decide if they want a more or less specificity when choosing their preferences; for example to distinguish between the profiles of 'engineer' and 'technician'. This methodology would only be possible if two attributes were added when labeling, the main category and a secondary category. Having two categories, it might more interesting to work with serial classifiers. One of them in charge of distinguishing the most generic profiles and then a specialized classifier (for each profile) would try to distinguish the possible subcategories. As the second stage of classification is much more specific, it is likely that similar categories could be decoupled better. On the other hand, it is also proposed to investigate the results of using other text classification techniques, such as LDA [33]. Although, because the algorithm is also based on word frequency, it is likely that the overlapping between job profiles will cause the technique not to be completely effective. It is also possible to find ways to give more or less weight to the data for increasing the difference between similar categories. Algorithms such the one proposed by Lesk [38] could be used for this purpose. Finally, it is proposed to research the combination of NLP techniques with algorithms based on word vectors to try to find some combination that captures better the semantics of texts when they are classified.

REFERENCES

- [1] Lombardero, L. (2015), Trabajar en la era digital: tecnología y competencias para la transformación digital. Lid Editorial
- [2] Neller, T. W. (2017), *AI education: Machine learning resources*. AI Matters, 3(2)
- [3] Jain, A. K. (2010), Data clustering: 50 years beyond K-means. *Pattern recognition letters*. 31(8), 651-666
- [4] Buchta, C., Kober, M., Feinerer, I., & Hornik, K. (2012), *Spherical k-means clustering*. *Journal of Statistical Software*, 50(10), 1-22
- [5] Blumberg, R., & Atre, S. (2003), *The problem with unstructured data*. *Dm Review*, 13(42-49), 62
- [6] Buneman, P., Davidson, S., Fernandez, M., & Suciu, D. (1997, January), *Adding structure to unstructured data*. In *International Conference on Database Theory* (pp. 336-350). Springer, Berlin, Heidelberg
- [7] Miller, G. A. (1995), *WordNet: a lexical database for English*. *Communications of the ACM*, 38(11), 39-41
- [8] Martin, J. H., & Jurafsky, D. (2009), *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition*. Upper Saddle River: Pearson/Prentice Hall.
- [9] Bird, S., Klein, E., & Loper, E. (2009), *Natural language processing with Python: analyzing text with the natural language toolkit*. O'Reilly Media, Inc..
- [10] Peres-Neto, P. R., Jackson, D. A., & Somers, K. M. (2005), *How many principal components? Stopping rules for determining the number of non-trivial axes revisited*. *Computational Statistics & Data Analysis*, 49(4), 974-997.
- [11] Crespo, R. G., Martínez, O. S., Lovelle, J. M. C., García-Bustelo, B. C. P., Gayo, J. E. L., & De Pablos, P. O. (2011), *Recommendation system based on user interaction data applied to intelligent electronic books*. *Computers in Human Behavior*, 27(4), 1445-1449.
- [12] Belkin, N. J., & Croft, W. B. (1992), Information filtering and information retrieval: two sides of the same coin. In *Communications of the ACM*.
- [13] Resnick, P., & Varian, H. R. (1997), *Recommender systems*.

Communications of the ACM, 40(3), 56-59.

- [14] Adomavicius, G., & Tuzhilin, A. (2005), *Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions*. IEEE Transactions on Knowledge & Data Engineering, (6), 734-749.
- [15] Linden, G., Smith, B., & York, J. (2003), *Amazon.com recommendations: Item-to-item collaborative filtering*. IEEE Internet computing, (1), 76-80.
- [16] Hofmann, T. (2003, July), *Collaborative filtering via gaussian probabilistic latent semantic analysis*. In Proceedings of the 26th annual international ACM SIGIR conference on Research and development in information retrieval (pp. 259-266). ACM.
- [17] Grosz, B. J., & Sidner, C. L. (1986), *Attention, intentions, and the structure of discourse*. Computational linguistics, 12(3), 175-204.
- [18] Chomsky, N. (1956), *Three models for the description of language*. IRE Transactions on information theory, 2(3), 113-124.
- [19] Padró, L., Collado, M., Reese, S., Lloberes, M., & Castellón, I. (2010), *Freeling 2.1: Five years of open-source language processing tools*. In 7th International Conference on Language Resources and Evaluation.
- [20] Baeza-Yates, R. (2004, February), *Challenges in the interaction of information retrieval and natural language processing*. In International Conference on Intelligent Text Processing and Computational Linguistics (pp. 445-456). Springer, Berlin, Heidelberg.
- [21] Russell, S. J., & Norvig, P. (2016), *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited.
- [22] Fortnow, L., & Homer, S. (2003), *A short history of computational complexity*. Boston University Computer Science Department.
- [23] Goodfellow, I., Bengio, Y., & Courville, A. (2016), *Deep learning*. MIT press.
- [24] Sutton, R. S., & Barto, A. G. (2011), *Reinforcement learning: An introduction*. MIT press.
- [25] Fiuza Pérez, M., & Rodríguez Pérez, J. C. (2000), *La regresión logística: una herramienta versátil*. Nefrología, 20(6), 495-500.
- [26] Yang, Y., & Liu, X. (1999, August), *A re-examination of text categorization methods*. In Sigir (Vol. 99, No. 8, p. 99).
- [27] Hossin, M., & Sulaiman, M. N. (2015), *A review on evaluation metrics for data classification evaluations*. International Journal of Data Mining & Knowledge Management Process, 5(2), 1.
- [28] Sagirolu, S., & Sinanc, D. (2013, May), *Big data: A review*. In 2013 International Conference on Collaboration Technologies and Systems (CTS), (pp. 42-47). IEEE.
- [29] Vargiu, E., & Urru, M. (2013), *Exploiting web scraping in a collaborative filtering-based approach to web advertising*. Artif. Intell. Research, 2(1), 44-54.
- [30] Myllymaki, J. (2002), *Effective web data extraction with standard XML technologies*. Computer Networks, 39(5), 635-644.
- [31] Moreno, I., Moreda, P., & Romá-Ferri, M. (2012), *Reconocimiento de entidades nombradas en dominios restringidos*. In Actas del III Workshop en Tecnologías de la Informática (pp. 41-57).
- [32] Robertson, S. (2004), *Understanding inverse document frequency: on theoretical arguments for IDF*. Journal of documentation, 60(5), 503-520.
- [33] Blei, D. M., Ng, A. Y., & Jordan, M. I. (2003), *Latent dirichlet allocation*. Journal of machine Learning research, 3(Jan), 993-1022.
- [34] Beel, J., Breitingger, C., & Langer, S. (2017), *Evaluating the CC-IDF citation-weighting scheme: how effectively can 'Inverse Document Frequency'(IDF) be applied to references*. Proceedings of the 12th iConference.
- [35] Turing, A. M. (2004), *Computing machinery and intelligence (1950), The Essential Turing: The Ideas that Gave Birth to the Computer Age*. Ed. B. Jack Copeland. Oxford: Oxford UP, 433-64.
- [36] Bobrow, D. G., Kaplan, R. M., Kay, M., Norman, D. A., Thompson, H., & Winograd, T. (1977), *GUS, a frame-driven dialog system*. Artificial intelligence, 8(2), 155-173.
- [37] Reiter, E., & Dale, R. (2000), *Building natural language generation systems*. Cambridge university press.
- [38] Banerjee, S., & Pedersen, T. (2002, February), *An adapted Lesk algorithm for word sense disambiguation using WordNet*. In International conference on intelligent text processing and computational linguistics (pp. 136-145). Springer, Berlin, Heidelberg.