



Tesis

Automatización de lectura de Curriculum Vitae
para selección de personal en el sector IT

Calonge, Federico Matias
calongefederico@gmail.com

22 de junio de 2022

Resumen

En la Tesis de Ingeniería en Informática que se presenta se diseña un *sistema de lectura automática de Curriculum Vitae* accesible vía Web. La finalidad del mismo es ayudar al reclutador laboral a elegir a los mejores candidatos para los puestos laborales de IT que tenga disponible. Esta elección se realiza mediante el uso de algoritmos de *machine learning* y basándose, principalmente, en una medición de similitud entre textos: Curriculum Vitae de los candidatos por un lado, y descripciones de los puestos laborales de IT por el otro. El sistema esta desarrollado utilizando el lenguaje de programación Python, permitiendo verificar la teoría desarrollada.

Abstract

This Computer Engineering Thesis introduces an *automatic Curriculum Vitae reading system* accesible via the Web. The purpose of it is to help the job recruiter to choose the best candidates for the available IT job positions. This choice is made through the use of *machine learning* algorithms and based mainly on a measurement of similarity between texts: Curriculum Vitae of the candidates on the one hand, and IT job descriptions on the other hand. The system is developed using the Python programming language allowing to verify the developed theory.

Índice

1. Introducción.	6
1.1. Algoritmos y técnicas utilizadas.	8
1.2. Objetivos del Proyecto.	9
1.2.1. Objetivo general.	9
1.2.2. Objetivos específicos.	9
1.3. Alcance del Proyecto.	10
1.4. Organización.	10
2. Reclutamiento y selección laboral.	12
2.1. Introducción.	12
2.2. Reclutamiento vs selección.	12
2.3. Evolución de los procesos de reclutamiento y selección laboral.	14
2.4. Cribado o screening.	15
2.4.1. Screening manual vs screening automatizado	15
2.5. Sistemas de screening: Estado del arte.	16
2.6. Enfoque del Proyecto.	18
3. Algoritmos de Machine Learning.	19
3.1. Introducción.	19
3.2. Machine Learning (ML).	19
3.2.1. Aprendizaje supervisado y no supervisado.	20
3.2.1.1. Regresión.	21
3.2.1.2. Clasificación.	22
3.2.1.3. Agrupación (clustering).	23
3.2.1.4. Algoritmos más conocidos.	24
3.2.2. Aprendizaje transductivo	25
3.2.3. Separación de los datos.	26
3.2.4. ¿Cómo implementar un modelo de ML?	27
3.2.4.1. Cross Validation.	31
3.2.4.2. Los Problemas de ML: Overfitting y Underfitting.	32
3.3. K-Nearest Neighbor (KNN).	33
3.3.1. Principal limitación KNN.	34

3.3.2.	Funcionamiento y ejemplo de KNN.	34
3.3.3.	Métrica de distancia a emplear.	36
3.3.4.	Eligiendo el valor de k: overfitting y underfitting.	38
3.4.	K-means.	40
3.4.1.	Funcionamiento y ejemplo de K-means.	40
3.4.2.	Objetivo de k-means y su función de coste.	44
3.4.3.	Posición inicial de los centroides.	45
3.4.4.	Limitaciones K-means.	47
3.4.4.1.	Obtención del k mediante Elbow Method.	48
3.4.4.2.	Inicialización de los centroides: k-means++.	49
3.5.	Redes Neuronales Artificiales (ANN).	51
3.5.1.	Perceptrón simple.	51
3.5.2.	Desventaja del Perceptrón simple.	53
3.5.3.	Perceptrón multicapa (MLP).	54
3.5.4.	Funciones de activación en MLP.	56
3.5.4.1.	Softmax en la capa de salida.	57
3.5.4.2.	Sigmoide en la capa de salida.	58
3.5.5.	Entrenamiento en una red neuronal.	59
3.5.5.1.	Hiper-parámetro n.	61
4.	Natural Language Processing.	63
4.1.	Introducción.	64
4.2.	Preprocesamiento de textos.	65
4.3.	Similitud entre textos.	66
4.4.	Técnicas para medir Similitud entre textos.	67
4.4.1.	Cosine Similarity.	67
4.4.2.	Word Mover's Distance (WMD).	68
4.5.	Técnicas de vectorización.	69
4.5.1.	TF-IDF.	69
4.5.2.	Word Embeddings.	70
4.5.2.1.	¿Cómo entrenar Word Embeddings?	71
5.	Implementación.	72

5.1.	Obtención del modelo de clasificación.	73
5.1.1.	Introducción.	73
5.1.2.	Esquema.	73
5.1.3.	Obtención de sets de datos.	74
5.1.3.1.	Curriculum Vitae.	74
5.1.3.2.	Descripciones Puestos Laborales.	74
5.1.4.	Preprocesamiento de textos.	75
5.1.5.	Cantidad final del set de datos y su uso en las distintas etapas. . . .	76
5.2.	Comparando textos y obteniendo similitudes.	78
5.2.1.	TF-IDF & Cosine Similarity.	79
5.2.2.	Word embeddings (Word2vec) & WMD.	79
5.2.2.1.	Elección de hiper-parámetros Word2vec.	79
5.2.2.2.	Word Mover's Distance (WMD).	79
5.3.	Armado del modelo de clasificación KNN.	79
5.4.	Clasificación de nuevas muestras y resultados obtenidos.	80
5.5.	Integración al Sistema Web.	81
5.5.1.	Base de datos.	82
5.5.2.	Secciones del sistema	84
5.5.3.	Manejo de los datos.	87
5.5.3.1.	Modelado.	88
5.5.3.2.	Filtrado.	89
5.5.3.3.	Visualización.	90
5.6.	Pipeline Flow final del Sistema.	91
5.7.	Conclusiones.	92
5.8.	Caso de Uso.	93
5.9.	Limitaciones del sistema.	94
6.	Próximos pasos.	95
7.	Anexos.	96
7.1.	Funciones de activación.	96
7.2.	Ejemplo de obtención de Word Embeddings mediante skipgram y softmax. .	97
7.3.	Palabras repetidas en nuestro Corpus y palabras polisémicas.	105

1 Introducción.

Los procesos de *reclutamiento y selección laboral* se han vuelto cruciales para el manejo de recursos humanos en el mundo moderno. Con las transformaciones digitales de las empresas y del mercado laboral en general, identificar los perfiles más acordes a las necesidades de la empresa se convirtió en uno de los retos más ambiciosos de Recursos Humanos, en especial cuando hablamos del *Sector IT*, donde año tras año se van generando nuevos puestos de trabajo y estos mismos van creciendo en demanda. Este crecimiento de la demanda en distintos puestos del Sector IT lo podemos evidenciar, a modo de resumen, en la figura 1.1.

En estos últimos años se implementaron una gran cantidad de herramientas de Software que utilizan algoritmos inteligentes y que permiten automatizar y gestionar información de los candidatos de una manera mucho más intuitiva[12, 13, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]. La *automatización* se transformó en un factor clave e indispensable para que el reclutador pueda conseguir a los candidatos más relevantes para los puestos que ofrece la empresa en un tiempo muy corto. Se espera que para el año 2025 muchas tareas realizadas actualmente por humanos, sean automatizadas por máquinas: esto lo podemos evidenciar en la figura 1.2.

↗ Increasing demand		↘ Decreasing demand	
1	Data Analysts and Scientists	1	Data Entry Clerks
2	AI and Machine Learning Specialists	2	Administrative and Executive Secretaries
3	Big Data Specialists	3	Accounting, Bookkeeping and Payroll Clerks
4	Digital Marketing and Strategy Specialists	4	Accountants and Auditors
5	Process Automation Specialists	5	Assembly and Factory Workers
6	Business Development Professionals	6	Business Services and Administration Managers
7	Digital Transformation Specialists	7	Client Information and Customer Service Workers
8	Information Security Analysts	8	General and Operations Managers
9	Software and Applications Developers	9	Mechanics and Machinery Repairers
10	Internet of Things Specialists	10	Material-Recording and Stock-Keeping Clerks
11	Project Managers	11	Financial Analysts
12	Business Services and Administration Managers	12	Postal Service Clerks
13	Database and Network Professionals	13	Sales Rep., Wholesale and Manuf., Tech. and Sci.Products
14	Robotics Engineers	14	Relationship Managers
15	Strategic Advisors	15	Bank Tellers and Related Clerks
16	Management and Organization Analysts	16	Door-To-Door Sales, News and Street Vendors
17	FinTech Engineers	17	Electronics and Telecoms Installers and Repairers
18	Mechanics and Machinery Repairers	18	Human Resources Specialists
19	Organizational Development Specialists	19	Training and Development Specialists
20	Risk Management Specialists	20	Construction Laborers

Figura 1.1: Top 20 demanda de roles laborales en aumento y disminución para el año 2020, por participación de las empresas encuestadas por el *Foro Económico Mundial*[2].

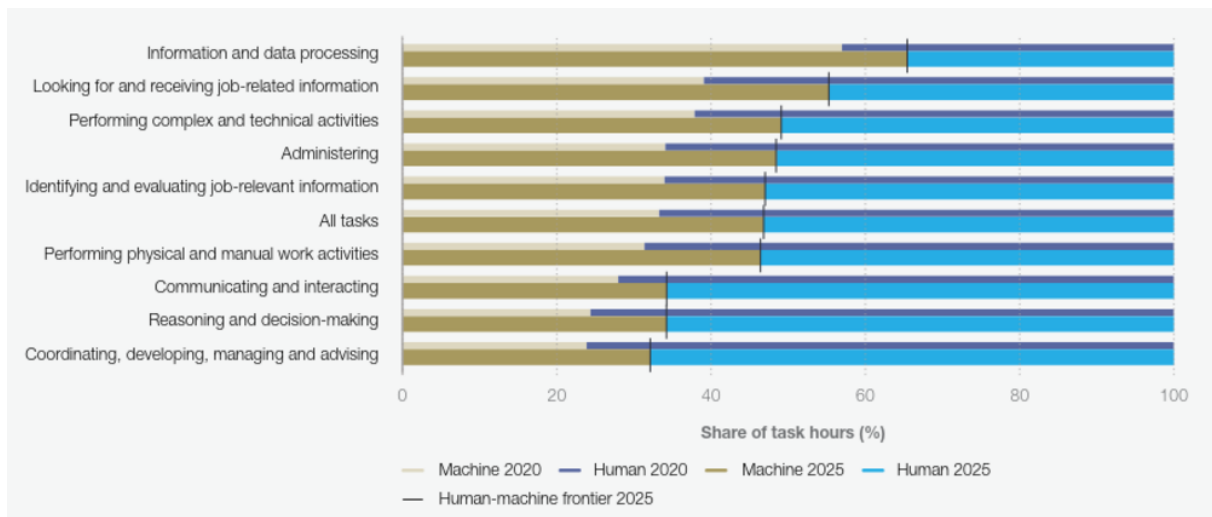


Figura 1.2: Porcentaje de tareas realizadas por humanos frente a máquinas, 2020 y 2025 (previsto), por participación de las empresas encuestadas por el *Foro Económico Mundial*[2].

El tema de este Proyecto de Tesis será desarrollar un *sistema de lectura automática de Curriculum Vitae* accesible vía Web. La finalidad del mismo es ayudar al reclutador laboral a elegir a los mejores candidatos para los puestos laborales de IT que tenga disponible. Esta elección se realiza mediante el uso de algoritmos de machine learning y basándose, principalmente, en una medición de similitud entre textos: Curriculum Vitae de los candidatos por un lado, y descripciones de los puestos laborales de IT por el otro.

La medición de similitudes entre documentos de texto (text similarity measurement) es uno de los problemas más cruciales del *Procesamiento del Lenguaje Natural (NLP)*. Encontrar similitudes entre documentos se utiliza en varios dominios de NLP, tales como en sistemas de recomendación, de information retrieval (IR), de análisis de sentimientos, etc.[1]

Para que las máquinas puedan describir esta similitud entre documentos, se necesita definir una forma de medir matemáticamente la similitud, la cual debe ser comparable para que la máquina pueda identificar qué documentos son más (o menos) similares. Previamente a esto necesitamos representar el texto de los documentos en una forma cuantificable (que suele ser en forma vectorial), de modo que podamos realizar los posteriores cálculos de similitud sobre él. Una distancia pequeña entre los vectores significa un alto nivel de similitud, mientras que grandes distancias significan un bajo nivel de similitud[1].

Por lo tanto, resumidamente los pasos necesarios para poder medir similitudes entre documentos son[1]:

1. Convertir cada uno de los documentos en un objeto matemático (vector).
2. Definir y emplear una métrica de distancia que será utilizada como nuestra medida de similitud entre los textos.

En el sistema desarrollado, una vez obtenidas estas mediciones de similitud entre los Curriculum Vitae de los candidatos y las descripciones de los puestos laborales de IT mediante distintos algoritmos y técnicas de machine learning, estos valores se utilizarán para alimentar y generar *un modelo de clasificación*, el cual nos servirá para lograr, en base a los valores de similitud de nuevos candidatos, clasificar qué tan similares son dichos candidatos con respecto a la descripción de un puesto de IT: similitud escasa, similitud media, similitud alta, similitud muy alta.

1.1 Algoritmos y técnicas utilizadas.

A continuación nombraremos resumidamente los algoritmos y técnicas de machine learning utilizadas para el desarrollo del *sistema de lectura automática de Curriculum Vitae*.

Como se mencionó en *Introducción*, el primer paso para lograr nuestra medición de similitudes entre documentos consiste en convertir los documentos en vectores: para esto se utilizaron las técnicas de vectorización **TF-IDF** y **Word Embeddings**.

Para el segundo paso, definir y emplear métricas de distancia, se decidió emplear una combinación de las técnicas **Cosine Similarity** y **Word Mover's Distance (WMD)**. En la sección 4.5.3-¿Por qué decidimos utilizar Cosine similarity y WMD? se detallan las razones por las cuales se decidieron elegir estos métodos; las cuales están justificadas en base al análisis realizado en *Técnicas para medir Similitud entre textos*, el cual detalla las numerosas técnicas existentes para obtener similitudes entre textos.

Como tercer y último paso, una vez obtenidas estas mediciones de similitud entre los Curriculum Vitae de los candidatos y las descripciones de los puestos laborales de IT, estos valores se utilizaron para alimentar un **algoritmo de clustering K-means** que a su vez, con sus datos de salida (4 clusters), alimentan a un **modelo de clasificación K-Nearest Neighbor (KNN)**.

Finalmente, con este modelo KNN logramos, en base a los valores de similitud de nuevos candidatos, clasificar qué tan similares son dichos candidatos con respecto a la descripción de un puesto de IT.

Estos algoritmos, técnicas y modelos utilizados serán detallados en las secciones posteriores, tanto teóricamente como en su implementación.

1.2 Objetivos del Proyecto.

1.2.1 Objetivo general.

El objetivo de este Proyecto de Tesis es lograr un desarrollo, tanto teórico como práctico, de un *sistema de lectura automática de Curriculum Vitae* accesible vía Web. La finalidad del mismo es ayudar al reclutador laboral a elegir a los mejores candidatos para los puestos laborales de IT que tenga disponible. Esta elección se realiza mediante el uso de algoritmos de machine learning y basándose, principalmente, en una medición de similitud entre textos:

- los Curriculum Vitae de los candidatos por un lado,
- descripciones de los puestos laborales de IT por el otro.

1.2.2 Objetivos específicos.

Los objetivos específicos de este Proyecto de Tesis son:

- Describir el estado del arte actual de los Sistemas de lectura y análisis de Curriculum Vitae en las fases de reclutamiento y selección laboral.
- Implementar un Sistema de lectura automática de Curriculum Vitae basado en la comparación y medición de similitudes entre textos, para finalmente obtener una visualización de los mejores candidatos para un puesto laboral de IT determinado.
- Aprender los conceptos y técnicas principales utilizadas dentro del procesamiento de lenguaje natural (NLP) aplicando técnicas de preprocesamiento y limpieza de textos.
- Implementar diferentes técnicas para medir similitudes entre los textos (Cosine Similarity y Word Mover's Distance -WMD-) y diferentes técnicas de vectorización (TF-IDF y Word Embeddings), analizando su funcionamiento tanto teórica como matemáticamente, ventajas y desventajas.
- Conocer, implementar e integrar el algoritmo de clustering K-means junto al algoritmo de clasificación KNN para obtener un modelo de clasificación de candidatos en base a las medidas de similitud entre los textos.
- Evaluar los Frameworks disponibles para tener una UI ¹ accesible vía web e integrar el mismo al Sistema.
- Almacenar datos de candidatos, reclutadores y puestos laborales en una base de datos.

¹La interfaz de usuario o user interface (UI) de una página web refiere a todo aquello tangible con lo que los usuarios interactúan de forma directa en la misma.

1.3 Alcance del Proyecto.

El alcance de esta Tesis de Grado de Ingeniería incluye el desarrollo de conceptos de análisis de datos y machine learning, procesamiento de lenguaje natural, técnicas de preprocesamiento y limpieza de los datos, técnicas de vectorización y técnicas para medir la similitud entre textos, algoritmos de clasificación y clustering, integración con frameworks, visualización de datos, y gestión de Base de Datos, de acuerdo a lo enunciado en los objetivos específicos.

1.4 Organización.

Este Proyecto de Tesis fue organizado para trabajarlo en tres secciones:

1. Análisis e investigación inicial.

Esta sección abarca principalmente la parte teórica del trabajo, haciendo hincapié en el análisis e investigación de:

- El estado del arte (actual y pasado) de los sistemas de lectura y análisis de Curriculum Vitae.
- Técnicas usadas para el procesamiento del lenguaje natural (NLP).
- Técnicas para medir similitudes entre textos: Cosine Similarity y WMD.
- Técnicas de vectorización: TF-IDF y Word Embeddings.
- Algoritmos de machine learning para tareas de clasificación (KNN) y clustering (K-means).

Esta primera sección abarca los capítulos *Reclutamiento y selección laboral*, *Algoritmos de Machine Learning* y *Natural Language Processing* de este Informe de Tesis.

2. Implementación de distintas técnicas y algoritmos para la obtención del modelo de clasificación KNN.

Esta sección hace referencia a la aplicación práctica dentro del marco teórico desarrollado en la primera sección, mediante la realización de una serie de análisis en documentos de Jupyter Notebook² utilizando Python³, para la obtención final de nuestro modelo de clasificación KNN capaz de clasificar, en base a los valores de similitud de nuevos candidatos, qué tan similares son dichos candidatos con respecto a la descripción de un puesto de IT: similitud escasa, similitud media, similitud alta, similitud muy alta.

²Aplicación cliente-servidor que permite crear documentos web en formato JSON que siguen un esquema versionado y una lista ordenada de celdas de entrada y de salida. Estas celdas albergan, entre otras cosas, código, texto (en formato Markdown), fórmulas y ecuaciones matemáticas. Estos documentos que se generan funcionan en cualquier navegador estándar.

³Lenguaje de programación interpretado y multiplataforma de código abierto, popularizado en los últimos años por su facilidad para trabajar con inteligencia artificial, big data, machine learning y data science, entre muchos otros campos en auge.

Los items que abarca esta sección son:

- Obtención de sets de datos: curriculums vitae y descripciones laborales.
- Preprocesamiento de los textos.
- Comparación entre textos y obtención de similitudes entre los mismos mediante el uso de las técnicas para medir distancias y obtener dichas similitudes (WMD y Cosine Similarity) y las técnicas de vectorización (TF-IDF y Word Embeddings).
- Obtención del modelo de clasificación KNN utilizando como datos de entrada los clusters devueltos por el algoritmo K-means obtenidos en base a las mediciones de similitud previamente realizadas.
- Análisis y primeras visualizaciones de los resultados.

Esta sección abarca el capítulo *Implementación* (desde *Obtención del modelo de clasificación* hasta *Clasificación de nuevas muestras y resultados obtenidos*) de este Informe de Tesis.

3. Integración al sistema web.

Esta última sección hace referencia a la reutilización de las funciones que contienen la lógica de los distintos algoritmos utilizados junto con el modelo de clasificación KNN obtenidos previamente en la sección 2, para integrar todo este conjunto en el sistema web que, a su vez, está integrado a una base de datos relacional. De esta manera, el sistema cuenta con una interfaz gráfica permitiendo interactuar entre candidatos y reclutadores y, principalmente, permitiendo que el reclutador sea capaz de obtener un listado con los N candidatos más similares a un puesto determinado, y ordenados de mayor a menor de acuerdo a esta *similitud*. Dicha *similitud* representa el resultado obtenido de la clasificación por nuestro modelo KNN.

Los items principales de esta sección son:

- Definición de los usuarios que accederán al sistema.
- Definición de los datos que se almacenarán.
- Integración de frameworks y bases de datos.
- Modelado, filtrado y visualización de los datos.
- Reutilización e integración al sistema de los algoritmos y del modelo KNN utilizados en la fase previa.
- Evaluación del funcionamiento de todo el Sistema integrado.

Esta última sección abarca el capítulo *Implementación* (desde *Integración al Sistema Web* hasta el final del capítulo) de este Informe de Tesis.

2 Reclutamiento y selección laboral.

En este capítulo se va a realizar una introducción a los procesos de reclutamiento y selección laboral, sus diferencias y diferentes tareas involucradas. Por último, se llevará a cabo un análisis del Estado de Arte actual de los sistemas de cribado (o más conocidos como sistemas de *screening*) y se detallará el enfoque utilizado para este Proyecto.

2.1 Introducción.

Los procesos de *reclutamiento y selección laboral* se han vuelto cruciales para el manejo de recursos humanos en el mundo moderno. Con las transformaciones digitales de las empresas y del mercado laboral en general, identificar los perfiles más acordes a las necesidades de la empresa se convirtió en uno de los retos más ambiciosos de Recursos Humanos, en especial cuando hablamos del *Sector IT*, donde año tras año se van generando nuevos puestos de trabajo y estos mismos van creciendo en demanda.

Los *reclutadores*, dentro de un departamento de recursos humanos, son los encargados de llevar a cabo los procesos de *reclutamiento y selección* laboral. Uno de sus objetivos principales es buscar talento humano para cubrir los puestos de trabajo vacantes que tenga la empresa.

2.2 Reclutamiento vs selección.

El **reclutamiento** es el proceso de atracción, búsqueda, recolección e identificación de candidatos que encajan con la oferta de trabajo y, en definitiva, con la empresa. Como mencionan Anwar y Abdullah, “*el reclutamiento es el proceso de descubrir y capturar candidatos calificados o apropiados para ocupar el puesto vacante*”[9].

El objetivo del reclutamiento es *atraer, buscar e identificar* a los candidatos más adecuados y mejor calificados para el puesto disponible, según las necesidades de la empresa.

El proceso de reclutamiento incluye las siguientes actividades[9]:

- Identificación de las necesidades del puesto a cubrir.
- Análisis de la descripción y especificaciones del puesto.
- Identificación de posibles fuentes de candidatos cualificados para el puesto.
- Publicación del puesto vacante en dichas fuentes.
- Atracción de candidatos para aplicar al puesto.
- Manejo apropiado en las respuestas y en los escrutinios a las postulaciones.

En cambio, la **selección** es un proceso posterior al reclutamiento, donde se evalúa más detalladamente y se entrevista a los candidatos para el trabajo en particular. Como mencionan Rahman y Abdullah, *“la selección es un proceso de evaluar y entrevistar a los candidatos para un trabajo en particular y seleccionar a la persona adecuada para el puesto correcto”*[9].

El objetivo de la selección es *elegir y hacer efectiva la contratación* del candidato más adecuado y mejor calificado para el puesto disponible, según las necesidades de la empresa. Este procedimiento particiona a los candidatos en dos secciones: a los que se les ofrecerán el trabajo, y a los que se descartarán.

El proceso de selección incluye las siguientes actividades[9]:

- Recepción de la aplicación al puesto.
- Cribado o Screening de los candidatos, lo que permite avanzar con los candidatos adecuados y descartar a los no adecuados para el puesto.
- Entrevistas a los candidatos.
- Manejo de tests a los candidatos, tales como tests médicos o psicológicos.
- Manejo de exámenes a los candidatos, tales como exámenes técnicos, de aptitud, inteligencia, performance, etc.
- Evaluación de las referencias de los candidatos.
- Decisión final acerca de la contratación o no contratación del candidato.

Como conclusión, podemos decir que en la fase de **reclutamiento** se trata de encontrar muchos candidatos que cumplan con los requisitos de la oferta; mientras que en la etapa de **selección** se debe elegir al mejor candidato para las necesidades de la empresa.

2.3 Evolución de los procesos de reclutamiento y selección laboral.

Los procesos de reclutamiento y selección laboral fueron evolucionando a lo largo del tiempo[22]:

En los modelos de reclutamiento y selección de **primera generación**, las empresas anunciaban sus vacantes de puestos laborales en diarios, revistas, radio y en televisión. Los candidatos enviaban sus currículums por correo postal y los mismos se clasificaban manualmente: algo muy tedioso y que llevaba mucho tiempo en realizar. Una vez preseleccionados los candidatos, los reclutadores llamaban a los mismos para realizar las rondas de entrevistas.

Luego pasamos a la **segunda generación**. En esta época las empresas comenzaron a crecer y también lo hicieron las necesidades de reclutamiento y selección. Las empresas empezaron a subcontratar sus procesos de reclutamiento y selección, naciendo de esta manera las consultoras o agencias de contratación. Estas consultoras requerían que los candidatos cargaran sus currículums en sus sitios web en formatos particulares. Luego, las consultoras revisaban los datos de los candidatos y preseleccionaban a los mismos para la empresa. El gran inconveniente de este proceso fue que habían numerosas consultoras y cada una tenía su propia y única forma de selección, no era un proceso uniforme.

Para intentar superar los problemas anteriores, se llegó a una **tercera generación**, en la que estamos actualmente. En esta generación se crearon, y siguen creándose, una gran cantidad de herramientas de Software que utilizan algoritmos inteligentes y permiten automatizar y gestionar información de los candidatos de una manera mucho más intuitiva. Estos sistemas ayudan a los reclutadores dentro de las empresas y consultoras a analizar la información de cualquier Curriculum Vitae y clasificarlos o listarlos en función de los puestos disponibles. De esta manera, cuando el reclutador publica una oferta de trabajo, estos sistemas clasifican o listan a los currículums basándose en distintas métricas (por ejemplo palabras clave) mostrando así los candidatos más relevantes para la empresa o consultora.

2.4 Cribado o screening.

Como vimos anteriormente en *Reclutamiento vs selección*, el screening (o también conocido como cribado) es una etapa del proceso de selección. En esta etapa los reclutadores revisan los Currículums Vitae que fueron recibiendo por parte de los candidatos, y preseleccionan a los que mejor se adapten a los requisitos de la oferta de empleo de la empresa. Este proceso es muy importante dentro del proceso de selección, ya que permite valorar en ese momento si el candidato es apto para continuar en el proceso de selección o, en caso contrario, si el mismo se descarta por no considerarlo adecuado para el puesto.

2.4.1 Screening manual vs screening automatizado

Si el proceso de screening sigue el modelo tradicional y se realiza manualmente, es un proceso muy tedioso y que lleva mucho tiempo por parte del reclutador, el cual tiene que evaluar una gran cantidad de Currículums Vitae. Existe un estudio[11] realizado en 2018 por Ladders Inc., empresa líder en sitios de carrera, donde se estudió cuánto tiempo en promedio tarda un reclutador en dar una primera vista rápida a un Curriculum Vitae de un candidato. Se descubrió que este tiempo es, en promedio, de 7.4 segundos. Este es el tiempo en el que el reclutador decide inicialmente si el candidato sigue (o no) con el proceso de selección.

Sin embargo, este tiempo es engañoso: ya que este estudio fue realizado con la aplicación de muchos Currículums vitae que no cumplían con los criterios mínimos para calificar a los puestos; es por esto que los reclutadores realizaban una primer vista rápida y los descartaban (o no) a los pocos segundos. Si las aplicaciones provinieran de Currículums Vitae que cumplan con los requisitos mínimos del puesto, este tiempo de observación sería mucho mayor, ya que el reclutador examinaría con mayor detalle los curriculums.

Adicionalmente, este estudio tampoco tiene en cuenta el tiempo consumido por el reclutador al comparar el Curriculum Vitae del candidato con la descripción o requisitos del puesto laboral, por lo que el tiempo en realizar un screening manual se incrementaría significativamente.

Otra de las desventajas al considerar el screening como un proceso manual, es que muchos son los factores que pueden influir al reclutador en el momento de tomar la decisión de descartar o selección del candidato, ya sea cansancio por el volumen de los curriculums ya revisados, la estructura, elementos discriminatorios o información incompleta dentro de los mismos, etc. Es por esto que las probabilidades de descartar prematuramente a un candidato válido son muy altas.

Frente a estas desventajas del modelo tradicional y poco eficiente del screening manual existe una solución: la *automatización*. Actualmente existen sistemas automatizados y basados en Inteligencia Artificial que permiten realizar el screening de un volumen importante de curriculums en unos pocos segundos (Ver *Sistemas de screening: Estado del arte*). De esta manera, el proceso de screening no estaría afectado a factores externos que puedan influir en la decisión del reclutador, y además la herramienta sería capaz de entregar un listado justificado de los mejores candidatos para la siguiente fase del proceso en un tiempo relativamente corto.

Uno de los beneficios potenciales de utilizar sistemas automatizados e inteligentes en los procesos de screening es el acortamiento de los ciclos de contratación, lo que hace que la organización responda mejor a los solicitantes y sea más capaz de competir con otras organizaciones por los mejores candidatos[8].

Muchos gerentes de recursos humanos depositan sus esperanzas en la tecnología y las herramientas automatizadas e inteligentes: desde el aumento de la eficiencia y la reducción de costos hasta el aumento de las aplicaciones de candidatos y la estandarización de todos sus sistemas de selección[8].

2.5 Sistemas de screening: Estado del arte.

Tal como mencionamos en la sección anterior, un sistema de screening automatizado e inteligente nos permite entregar al reclutador una shortlist con los candidatos que mejor se adaptan a los requisitos de la oferta de empleo de la empresa. De esta manera, al realizar dicha preselección de candidatos, el reclutador podrá seguir con las etapas posteriores de selección o podrá realizar otro screening manual para descartar a otros candidatos.

A continuación detallaremos una serie de trabajos de relevancia (tanto de implementación como de investigación) donde se utilizaron distintos enfoques para desarrollar o sugerir un desarrollo de un sistema de screening de candidatos.

En [12] utilizaron el *algoritmo KNN* para clasificar los curriculums de los candidatos en diferentes categorías, y luego utilizaron la métrica de similitud *Cosine Similarity* para averiguar qué tan cerca está el curriculum del candidato con respecto a la descripción de los puestos, y realizar un ranking acorde a estos resultados.

En [13] utilizaron una herramienta inteligente llamada *EXPERT mapping-based candidate screening* que utiliza *ontology mapping*⁴ para crear un sistema automatizado para el screening inteligente de candidatos.

En [15] y [16] también se utilizan sistemas basados en *ontologías* para extraer datos de curriculums y realizar macheos con los puestos disponibles.

En [17] sugieren un método de matching basado en un *modelo probabilístico* para ser utilizado en la selección y recomendación de candidatos.

En [18] se propuso un sistema automatizado de screening de currículums que usa *Vector Space Model*⁵ para machear cada curriculum con la descripción del puesto correspondiente y *cosine similarity* como medida de similitud.

⁴La *ontología (ontology)* permite la especificación explícita de un dominio de discurso, que permite acceder y razonar sobre el conocimiento de un agente. Las ontologías elevan el nivel de especificación del conocimiento, incorporando semántica a los datos. El *mapeo de ontologías (ontology mapping)* es el proceso mediante el cual dos ontologías se relacionan semánticamente a nivel conceptual y las instancias de ontología de origen se transforman en entidades de ontología de destino de acuerdo con esas relaciones semánticas.[14]

⁵Modelo algebraico para representar documentos de textos en lenguaje natural de una manera formal mediante el uso de vectores de identificadores. Es utilizado para la recuperación, filtrado, indexado y cálculo de relevancia de información.

En [19] y [20] se propuso un sistema para el screening de candidatos, para el cual se utiliza *cosine similarity*, *KNN* y un *sistema de recomendación basado en contenido* (*Content Based Filtering*, *CBF*) para buscar los curriculums más cercanos a la descripción del puesto.

En [21] se propuso un sistema para el ordenamiento y clasificación de curriculums utilizando el concepto de inteligencia artificial (sin especificar qué algoritmos o técnicas específicas se deberían usar). Lo que se propuso fue un esquema de trabajo para luego seguir con la implementación del sistema; el cual permitiría clasificar a todos los curriculums de acuerdo con los requisitos de la empresa y los enviaría a Recursos Humanos para su posterior consideración.

En [22] se diseñó un sistema que ayuda a los reclutadores a seleccionar los curriculums basados en la descripción del trabajo. Ayuda en un proceso de contratación fácil y eficiente al extraer los requisitos automáticamente. Este sistema utiliza un *modelo NER* (*Named entity recognition*)⁶.

En [23] y [24] se implementan sistemas basados en *semantic annotation*⁷ y *ontologías*, permitiendo realizar matcheos entre las ofertas de trabajo y los curriculums de los candidatos.

En [25] se utiliza *relevance feedback*⁸ para la implementación de un sistemas que intenta encontrar a los mejores candidatos para las distintas ofertas de trabajo.

Paul Resnick y *Hal R. Varian* fueron pioneros en los *sistemas de recomendación*⁹[28]. En [26] y [27] se utilizaron dichos *sistemas de recomendación* junto a otros algoritmos para implementar aplicaciones de screening y clasificación de candidatos. Esto es debido a que, dado que el perfil del candidato es necesario para un puesto en particular, la forma en que se comparan los curriculums de los candidatos es muy similar a un sistema de recomendación.

⁶Subtarea de *information extraction (IE)* que permite buscar y categorizar entidades específicas en un cuerpo o cuerpos de textos.

⁷Proceso de etiquetado de documentos con conceptos relevantes. Los documentos se enriquecen con metadatos que permiten vincular el contenido a los conceptos, descritos en un gráfico de conocimiento. Esto hace que el contenido no estructurado sea más fácil de encontrar, interpretar y reutilizar.

⁸Característica de algunos sistemas de *information retrieval (IR)*. La idea de *relevance feedback* es tomar los resultados que se devuelven inicialmente de una consulta/query determinada, recopilar los comentarios de los usuarios y usar información para evaluar si esos resultados son relevantes o no para realizar una nueva consulta/query.

⁹Un *sistema de recomendación* es una subclase de los sistemas de *Information filtering (IF)* que busca predecir la calificación o la preferencia que un usuario le puede dar a un artículo. En palabras simples, es un algoritmo que sugiere artículos relevantes para los usuarios.

2.6 Enfoque del Proyecto.

Como vimos en la sección anterior, hay cientos de enfoques y combinaciones posibles para realizar un sistema de screening automático e inteligente.

El enfoque elegido para este Proyecto de Tesis fue realizar un sistema híbrido que utilice técnicas de mediciones de similitud entre los curriculums de los candidatos y las descripciones de los puestos disponibles, junto con técnicas de machine learning (de clustering y clasificación).

Como métricas de medición de similitud se utilizó una de las más comúnmente usadas, *similitud del coseno*, y una métrica que **no se usó en ninguno de los trabajos anteriores** pero, que sin embargo, es muy reciente y prometedora, y es la principal técnica utilizada para la medición semántica de la distancia entre textos, *Word Mover's Distance (WMD)*. Estas métricas serán explicadas más en detalle en las secciones posteriores (Ver *Técnicas para medir Similitud entre textos*).

Una vez utilizadas estas mediciones de similitud se utilizaron algoritmos de machine learning: un algoritmo de clustering K-means, y luego un modelo de clasificación KNN.

Además del uso de *Word Mover's Distance*, otro aspecto distintivo de este sistema es que se realizó una **integración a una interfaz web**, la cual la mayoría de los sistemas descriptos anteriormente no poseen. De esta manera, mediante la interfaz web se logra una interacción entre usuarios candidatos y reclutadores y, principalmente, permite que el reclutador sea capaz de visualizar un listado con los N candidatos más similares a un puesto determinado.

Por último, cabe destacar que los trabajos anteriormente mencionados no tienen un fácil acceso a los datasets ni al código que utilizaron para dichos sistemas, por lo que seguir el trabajo de ellos aplicando las mejoras que mencionan en sus artículos es una tarea casi imposible. En cambio, **este sistema será de código abierto**: los datasets, modelos y códigos utilizados estarán disponibles en Github y Git LFS¹⁰.

¹⁰https://github.com/FedericoCalonge/automatic_reading_of_CVs_using_text_similarity

3 Algoritmos de Machine Learning.

3.1 Introducción.

Continuando con el marco teórico de esta Tesis, es necesario explicar qué es *machine learning* (ML) y cómo se pueden clasificar a los distintos algoritmos y modelos según su tipo de aprendizaje, haciendo énfasis en los algoritmos K-Nearest Neighbor (KNN) y K-means, y el modelo de Redes Neuronales. Estas técnicas son las que se utilizarán en la implementación del proyecto.

Además, detallaremos la importancia de la separación de los datos y mencionaremos cuáles son los pasos a seguir para implementar un modelo de ML teniendo en cuenta algunos de los problemas clásicos que pueden afectar a nuestros resultados.

3.2 Machine Learning (ML).

La **Inteligencia Artificial (IA)** es una disciplina dentro de las ciencias de la computación que consiste en el desarrollo de algoritmos¹¹ que imiten el razonamiento humano, teniendo la capacidad de solucionar problemas que comúnmente resuelve la inteligencia natural pero de la manera más eficiente posible. Esencialmente, la IA permite que las máquinas puedan actuar e implementar distintas tareas que están fuera del alcance de los humanos [29].

Machine Learning (ML) o **Aprendizaje Automático** es un subconjunto de la Inteligencia Artificial que se encarga de construir algoritmos que aprenden a hacer algo útil a partir de los datos. Como ML es un subconjunto de IA, esto implica que todos los algoritmos de ML son técnicas de inteligencia artificial, pero no todos los métodos de inteligencia artificial califican como algoritmos de ML.

El objetivo principal de ML es permitir que las computadoras aprendan sin intervención o asistencia humana. Esencialmente, los algoritmos de ML aprenden *patrones* ocultos basados en datos históricos de entrada, los cuales posteriormente utilizan para aprender a clasificar la información o realizar predicciones relacionadas con el problema que se quiera resolver.

Los algoritmos de ML hoy en día se usan en todo tipo de aplicaciones: en Amazon para recomendarnos qué productos comprar, en Twitter para recomendarnos qué usuarios seguir, en Google para predecir qué páginas son las más relevantes para una consulta, en Facebook para reconocer qué fotos contienen rostros de personas, en el mercado inmobiliario para predecir los precios de las propiedades, en el mercado de valores para predecir el precio de las acciones, etc. Son tantas las aplicaciones de ML en la industria que se ha producido una verdadera explosión del tema en los últimos años[10].

¹¹Un algoritmo informático es un conjunto de instrucciones definidas, ordenadas y acotadas para resolver un problema, realizar un cálculo o desarrollar una tarea.

3.2.1 Aprendizaje supervisado y no supervisado.

Podemos dividir a ML en dos grandes categorías: aprendizaje supervisado y aprendizaje no supervisado. Resumidamente, la diferencia entre estas dos categorías es la existencia de etiquetas (labels) en el subconjunto de datos de entrenamiento (Ver *Separación de los datos*).

- En el **aprendizaje supervisado**, el modelo es entrenado utilizando datos que están “etiquetados”. Esto quiere decir que el set de datos que es utilizado para enseñar al algoritmo tiene una etiqueta (o label) que define la respuesta/salida correcta.

Analizándolo matemáticamente, este proceso puede ser representado por una variable de entrada x y una conocida variable de salida y , donde se usa un algoritmo f para que aprenda a predecir resultados para aquellos nuevos datos que no tienen etiqueta.

$$y = f(x)$$

Los algoritmos de aprendizaje supervisado se utilizan para resolver problemas de **clasificación** y de **regresión**.

- El **aprendizaje no supervisado**, en cambio, se refiere al proceso de entrenamiento de un modelo de machine learning sin un set de datos etiquetado. Este tipo de aprendizaje es una rama muy importante en Data Science ya que, al trabajar solo con datos sin necesidad de tener un “label” para los mismos, únicamente necesitamos los datos en crudo y estos en general son mucho más fáciles de conseguir que datos ya previamente clasificados. Por ejemplo, las aplicaciones de redes sociales tienen grandes cantidades de datos sin etiquetar.

El objetivo de este método es experimentar con el conjunto de datos (los cuales contienen muchas características) y luego aprender propiedades útiles de la estructura de dicho conjunto de datos. Matemáticamente se cuenta con un dataset de entrenamiento de variables de entrada x pero no se conoce su salida.

$$¿? = f(x)$$

Los algoritmos de aprendizaje no supervisado se utilizan para resolver problemas de **clustering**.

En la Figura 3.1 se detalla un resumen de la división en las categorías de ML y su sub-división en modelos de clasificación, regresión y agrupación (clustering).

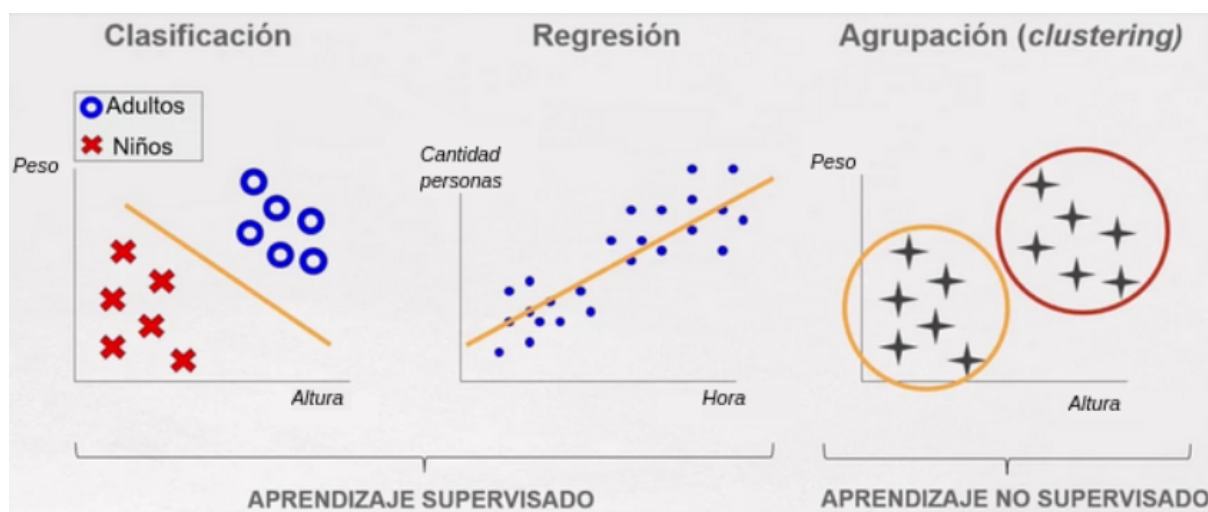


Figura 3.1: Tipos de aprendizaje en ML¹²

3.2.1.1 Regresión.

En un problema de regresión se intenta predecir el valor de una variable numérica y continua a partir de un cierto conjunto de datos.

En general contamos con un set de entrenamiento en el cual conocemos el valor de la variable que queremos predecir. El objetivo es entonces construir un modelo que nos permita predecir el valor de nuestra variable de salida a partir de datos nuevos. Dicho de otra forma, lo que se hace es buscar una función que represente los datos de entrenamiento y que permita generalizar correctamente¹³.

El caso más simple es la regresión lineal (observado previamente en la Figura 3.1), en el cual nuestro modelo es una recta, la recta que mejor se ajusta a los puntos de nuestro set de entrenamiento. En este ejemplo se intenta predecir la cantidad de personas en un lugar a una hora determinada.

Los problemas de regresión pueden usarse para realizar distintas predicciones, ya sea el valor de las acciones en el mercado de valores, predecir el costo de una propiedad, estimar las ganancias de un negocio, etc.

Las claves para identificar un problema de regresión[10] son las siguientes:

- Queremos predecir una variable que es numérica y continua.
- Contamos con un set de entrenamiento para el cual conocemos el valor de dicha variable.

¹²Sitio Web. *Modelos de Machine Learning*. <https://openwebinars.net/blog/modelos-de-machine-learning/>. (Consultado el 30 marzo de 2022).

¹³Generalizar es la capacidad de nuestro modelo de ML de obtener buenos resultados con datos nuevos reconociendo patrones generales de los datos de entrenamiento en lugar de reconocer particularidades específicas.

3.2.1.2 Clasificación.

En los problemas de clasificación la variable que se intenta predecir no es continua sino discreta, frecuentemente tiene pocos valores posibles y en muchos casos los valores posibles son solo dos: clasificación binaria¹⁴. La idea general es la misma que antes, para este caso contamos con un set de entrenamiento en el cual para cada dato conocemos la clase a la cual pertenece el mismo, y queremos construir un modelo que nos permita clasificar automáticamente datos nuevos cuya clase desconocemos.

Por ejemplo, en la Figura 3.1, observamos que tenemos una serie de personas, de las cuales contamos con su peso y altura, y queremos clasificarlas en adultos o niños. Lo que va a hacer un modelo de clasificación es aprender dónde están estos puntos y crear un clasificador que, para nuevos datos de entrada, consiga segmentarlos correctamente en adultos o niños.

Otras aplicaciones, por ejemplo, podrían ser analizar si las reviews de un producto son buenas o malas, medir la actitud del público en general ante determinadas noticias por los comentarios que existen en redes sociales, clasificar si un email es spam o no lo es, etc.

Las claves para reconocer un problema de clasificación[10] son:

- Queremos determinar la clase / grupo a la que pertenece cada dato.
- La clase es una variable discreta con un set de valores posibles limitado y definido.
- Contamos con un set de entrenamiento para el cual conocemos los datos y a qué clase pertenecen.

¹⁴Un caso típico de la clasificación binaria es un problema de análisis de sentimiento, donde queremos saber si un cierto texto es positivo o negativo, es decir si habla bien o mal de un cierto tema. Como set de entrenamiento entonces deberíamos contar con textos para los cuales ya conocemos su sentimiento.

3.2.1.3 Agrupación (clustering).

En un problema de clustering contamos con datos que queremos dividir en grupos de forma automática. Estos datos no tienen etiquetas, no sabemos a qué grupo pertenecen. Lo que hace un algoritmo de clustering es intentar buscar agrupaciones en los datos, creando de esta forma clusters¹⁵ con características similares. En algunos casos la cantidad de clusters la debemos indicar previamente y en otros el algoritmo es capaz de determinarla por sí mismo.

En el ejemplo de la Figura 3.1, podemos ver que se agrupan los puntos de la misma forma que en el problema de clasificación. La diferencia es que en este caso no sabemos si se trata de adultos o niños, sino que es el propio algoritmo el que va a identificar que hay dos grupos, y nosotros somos los que tenemos que interpretar los resultados.

Otros ejemplos de clustering podrían ser agrupar películas automáticamente de forma que queden juntas las que son de un mismo género; o la detección de comunidades en una red social, el cual es un típico problema de clustering en donde los puntos son los usuarios y queremos agruparlos automáticamente en comunidades. De esta forma podemos descubrir grupos de usuarios que tienen un cierto interés en común aun sin saber exactamente cuál es dicho interés.

Las claves para reconocer un problema de clustering[10] son:

- Contamos con un set de datos y queremos agruparlos en clusters/grupos.
- No son necesarios labels.

¹⁵Cluster, o grupo, es un conjunto de objetos que son "similares" entre ellos y "diferentes" de los objetos que pertenecen a los otros grupos. La palabra cluster viene del inglés y significa agrupación. Desde un punto de vista general, el cluster puede considerarse como la búsqueda automática de una estructura o de una clasificación en una colección de datos no etiquetados.

3.2.1.4 Algoritmos más conocidos.

Resumidamente, en la Figura 3.2 se detallan los algoritmos más conocidos para las distintas clasificaciones de machine learning previamente explicadas.

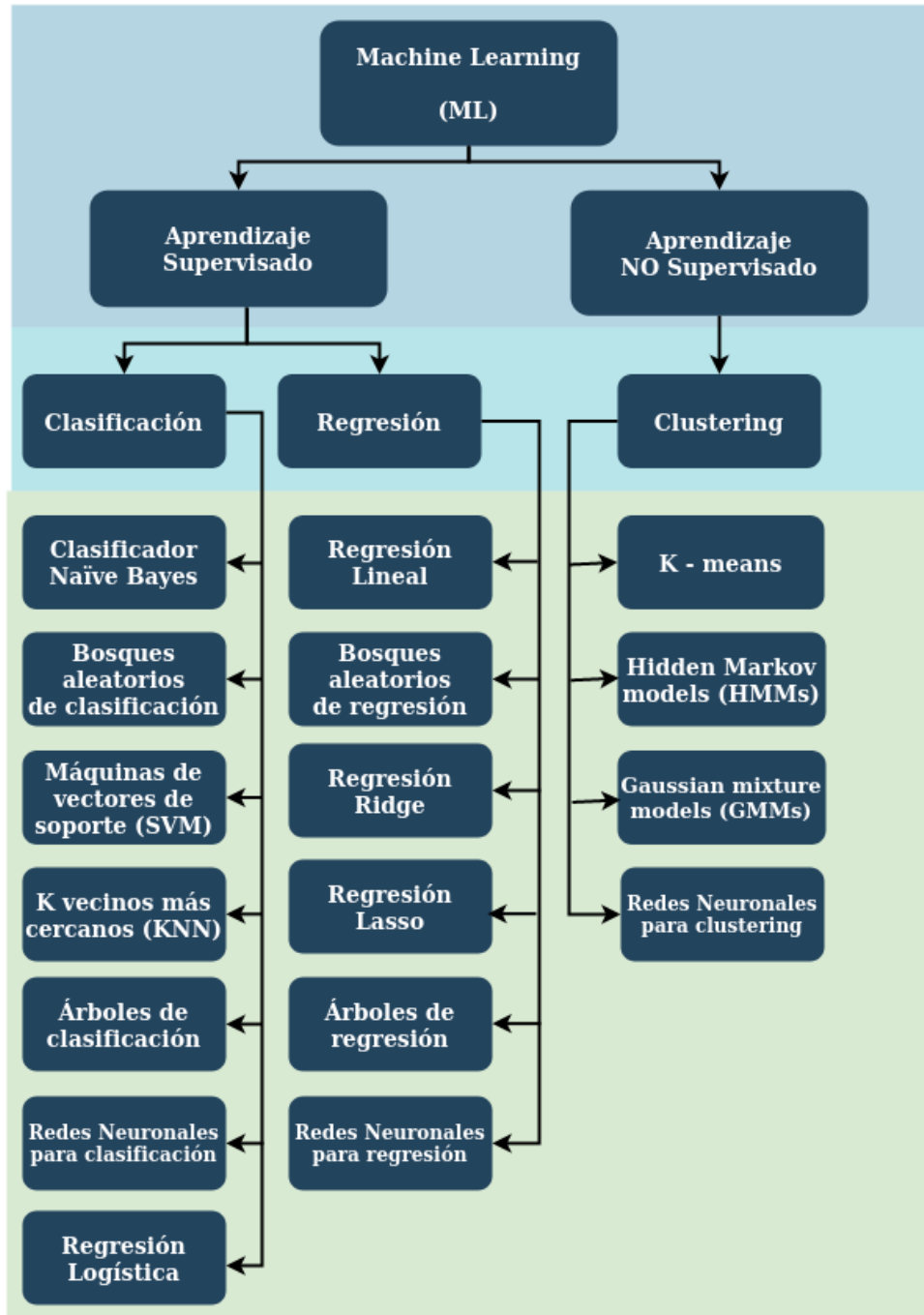


Figura 3.2: Algoritmos de machine learning más conocidos

Cabe destacar que algunos algoritmos se pueden adaptar para resolver problemas de otro tipo al cual están inicialmente categorizados. Por ejemplo, KNN que está categorizado como un algoritmo de clasificación, también se puede utilizar para resolver problemas de regresión. Otro caso conocido es Support Vector Regression (SVR), que utiliza las bases de Máquinas de vectores de soporte (SVM) para resolver problemas de regresión.

3.2.2 Aprendizaje transductivo

Existe una cierta relación entre los problemas de clustering y de clasificación, por ejemplo dado un problema de clasificación podríamos aplicar clustering primero y luego clasificar a cada punto de acuerdo al cluster al cual pertenece en base a la clase mayoritaria de dicho cluster. Este procedimiento no es muy frecuente pero es conveniente tenerlo en cuenta porque permite entender el funcionamiento de ciertos algoritmos que combinan las propiedades de un problema de clustering y uno de clasificación.

Una aplicación que combina clustering (aprendizaje no supervisado) y clasificación (aprendizaje supervisado) es la que denominamos **Aprendizaje transductivo**[30][10]. De esta manera, para predecir nuevos datos no etiquetados se utilizan los datos previamente etiquetados, así como los datos sin etiquetar, como forma de ayuda a un clasificador tradicional.

Consideremos el ejemplo de la Figura 3.3. Aquí tenemos solo dos puntos clasificados, uno clasificado como “blanco” y el otro como “negro”. Sin mayor información el punto marcado con el signo de pregunta, cuya clase desconocemos, quedaría clasificado como “blanco” ya que está mucho mas cerca del punto blanco que del punto negro.



Figura 3.3: Aprendizaje transductivo

En la Figura 3.4 al agregar puntos cuya clase desconocemos, vemos que en nuestros datos existen dos clusters. De esta manera, si tenemos que asociar cada cluster con un color entonces el de arriba es “negro” y el de abajo es “blanco”, ya que el punto negro pertenece al cluster de arriba y el blanco al de abajo. De este modo, el punto que inicialmente consideramos blanco, con esta nueva información de clusters, sería considerado negro.

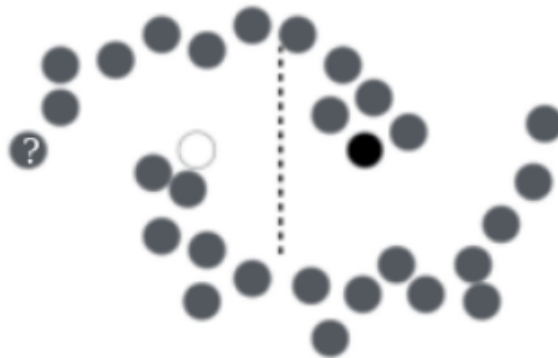


Figura 3.4: Aprendizaje transductivo

El aprendizaje transductivo es un área muy nueva dentro de Data Science y que sin dudas merece ser explorada.

En este proyecto de Tesis se utilizará el aprendizaje transductivo para lograr inicialmente una clusterización de nuestros candidatos en 5 grupos mediante un algoritmo de clustering (K-means), y luego se realizará una clasificación mediante un algoritmo de clasificación (KNN).

3.2.3 Separación de los datos.

Dos conceptos importantes que mencionaremos a lo largo del informe son: los “datos de entrenamiento” y los “datos de prueba”.

Como mencionamos anteriormente, los algoritmos de machine learning aprenden de los datos con los que los entrenamos. A partir de ellos, intentan encontrar o inferir el patrón que les permita predecir el resultado para un nuevo caso. Pero, para poder calibrar si un modelo funciona, necesitaremos probarlo con un conjunto de datos diferente. Por ello, en todo proceso de aprendizaje automático, los datos de trabajo se deben dividir mínimamente en dos partes:

- Los **datos de entrenamiento** son los datos que usamos para entrenar un modelo. La calidad de nuestro modelo de aprendizaje automático va a ser directamente proporcional a la calidad de estos datos. Por ello son muy importantes las tareas de limpieza y preprocesamiento de los mismos.
- Los **datos de prueba o evaluación** son los datos que nos “reservamos” para comprobar si el modelo que hemos generado a partir de los datos de entrenamiento “funciona”. Es decir, si las respuestas predichas por el modelo para un caso totalmente nuevo son acertadas o no.

Es importante que el conjunto de datos de prueba tenga un volumen suficiente como para generar resultados estadísticamente significativos, y a la vez, que sea representativo del conjunto de datos global. Normalmente el conjunto de datos se suele dividir en un **70 %/80 % de datos de entrenamiento** y un **30 %/20 % de datos de prueba**, pero se puede variar la proporción según el caso[10].

3.2.4 ¿Cómo implementar un modelo de ML?

Resumidamente, los pasos a seguir para implementar un modelo de ML y utilizarlo para realizar predicciones sobre los datos son los siguientes:

- Paso 1: Recolección de Datos.
- Paso 2: Preparación y preprocesamiento de los datos.
- Paso 3: Elección del modelo de ML.
- Paso 4: Entrenamiento del algoritmo.
- Paso 5: Evaluación del modelo.
- Paso 6: Parameter Tuning o configuración de parámetros.
- Paso 7: Utilizando nuestro modelo.

A continuación detallaremos en mayor detalle los mencionados pasos.

- Paso 1: Recolección de Datos.

Dada la problemática que deseamos resolver mediante algoritmos de ML, nuestro primer paso será recolectar los datos que utilizaremos posteriormente para “alimentar” a dicho algoritmo.

En este paso hay que tener muy en cuenta la calidad y cantidad de información que consigamos ya que impactará directamente en lo bien o mal que luego funcione nuestro modelo. Estos datos los podemos sacar de bases de datos, planillas de cálculo, utilizando técnicas de web scraping¹⁶ o mediante APIs¹⁷ para recopilar información de manera automática de diversas fuentes de Internet, etc.

- Paso 2: Preparación y preprocesamiento de los datos.

En este paso generalmente se realizan visualizaciones de los datos y se revisa si existen correlaciones entre las distintas “features”¹⁸ de nuestros datos. Al pre-procesar nuestros datos nos referimos a normalizar los mismos: eliminando duplicados y realizando distintas correcciones de errores. El preprocesamiento de datos usualmente tiene un impacto significativo en la performance de generalización de nuestro algoritmo de machine learning[32].

¹⁶Proceso dentro de Data Science que se utiliza para la extracción de datos de sitios web simulando cómo navegaría un ser humano por los mismos.

¹⁷API o interfaz de programación de aplicaciones, es un conjunto de métodos o funciones que ofrece cierta biblioteca para ser utilizada por otro software como una capa de abstracción.

¹⁸Features generalmente son las columnas de nuestro dataframe, archivo o base de datos -dependiendo cómo almacenamos nuestros datos-.

En esta etapa, además, es importante **separar** nuestros datos en dos grupos:

- un set de entrenamiento.
- un set de prueba.

Como mencionamos previamente, el set de test en general es un 20 % o 30 % del set de entrenamiento. Esta partición de los datos en estos dos conjuntos diferenciados permite generar el modelo a partir de los datos de entrenamiento para después comprobar su eficiencia con los datos reservados para test.

■ Paso 3: Elección del modelo de ML.

Una vez obtenidos y preprocesados estos datos, lo que se hace es **elegir el modelo de ML**¹⁹ de acuerdo al objetivo que tengamos o problema que deseemos resolver.

De esta manera, utilizaremos algoritmos de clasificación, regresión o clustering para construir nuestro modelo de ML a partir de los datos, de forma tal de luego poder usar dicho modelo para predecir nuevos datos.

■ Paso 4: Entrenamiento del algoritmo.

El siguiente paso es **entrenar** a nuestro algoritmo de ML. En este proceso mediante una serie de iteraciones nuestro algoritmo detecta patrones en nuestros datos que luego nos servirán para poder realizar predicciones con los nuevos datos que se incorporen al sistema.

La idea es entrenar a nuestro algoritmo con el set de entrenamiento (mediante una función *fit()*) para luego, en las etapas posteriores, aplicarlo al set de prueba (mediante una función *predict()*). De esta forma, los datos para los cuales queremos probar el algoritmo (set de test) nunca fueron vistos por el mismo en la etapa de entrenamiento, lo cual permite saber si el modelo fue capaz de generalizar correctamente.

■ Paso 5: Evaluación del modelo[10].

Hacer predicciones correctas sobre datos futuros suele ser el principal problema que queremos resolver al utilizar algoritmos de ML. Luego de entrenar el modelo se tiene que **evaluar** el mismo. Evaluar un modelo, resumidamente, es estimar su rendimiento para saber qué tan bien se desempeñará / predecirá para datos nuevos no vistos por el mismo.

Para poder evaluar un modelo correctamente, tenemos que contar con la separación de nuestros datos en set de entrenamiento y set de prueba que realizamos en los pasos previos. Esto lo hacemos, ya que evaluar la precisión predictiva de un modelo de ML con los mismos datos que se han utilizado para el entrenamiento no es útil, ya que compensa a los modelos que pueden “recordar” los datos de entrenamiento en lugar de generalizar.

¹⁹Un modelo de machine learning es la salida de información que se genera cuando se entrena un algoritmo de ML con datos. Después del entrenamiento, al proporcionar un modelo con una entrada, se le dará una salida. Por ejemplo, un algoritmo predictivo creará un modelo predictivo.

De esta manera, luego de haber entrenado nuestro modelo de ML, resumidamente lo que hacemos en esta etapa es comparar las predicciones realizadas sobre set de pruebas devueltas por el modelo de ML contra el valor de destino conocido para el mismo set de pruebas; y por último generar alguna **métrica de evaluación** que nos permite verificar la performance de nuestro modelo indicando la efectividad de las predicciones.

Dependiendo del tipo de modelo que tengamos, podemos utilizar distintas métricas de evaluación para verificar su performance. Como observación, se detallan algunas de las métricas más conocidas en la Tabla 1.

Modelos de regresión	Modelos de clasificación	Modelos de clustering
Mean square error (MSE)	Matriz de confusión o error	Inertia
Root MSE (RMSE)	Accuracy (Exactitud)	Homogeneity
Normalized RMSE (NRMSE)	Precision (Precisión)	Majority-representation
Mean absolute error (MAE)	Recall (Sensibilidad o TPR)	Adjusted Rand Index
Mean absolute percentage error (MAPE)	FP Rate (Especificidad o TNR)	Silhouette coefficient
		Dunn index

Tabla 1: Métricas para evaluar distintos tipos de modelos[33, 34, 35, 36].

Por ejemplo, si queremos verificar la performance de un modelo de clasificación podemos utilizar el **accuracy**. Esta métrica mide el % de aciertos: es el ratio de las predicciones correctas sobre el número total de instancias evaluadas. Por ejemplo, si el Accuracy es menor o igual al 50 % este modelo no será útil ya que sería como lanzar una moneda al aire para tomar decisiones. Si alcanzamos un 90 % o más podremos tener una buena confianza en los resultados que nos otorga el modelo.

■ Paso 6: Parameter Tuning o configuración de parámetros[10].

Si durante la evaluación no obtuvimos buenas predicciones y nuestra métrica de evaluación no logró ser la mínima deseada, es posible que tengamos problemas de overfitting (ó underfitting) y deberemos retornar al paso de entrenamiento (Paso 4) haciendo antes una nueva configuración de hiper-parámetros de nuestro modelo.

Cada modelo tiene un conjunto de parámetros e hiper-parámetros que necesita para funcionar:

- Los **parámetros** son los valores obtenidos por el propio algoritmo a partir de los datos, no son indicados manualmente.
- Los **hiper-parámetros**, en cambio, son datos que debemos pasarle al algoritmo para funcionar.

Como ejemplos de parámetros tenemos los coeficientes en una regresión lineal o logística, o los pesos en una red neuronal artificial. Y como ejemplos de hiper-parámetros, tenemos al 'k' en KNN, o los 'EPOCHs' que nos permiten incrementar la cantidad de veces que iteramos sobre nuestros datos de entrenamiento en una red neuronal artificial.

Para encontrar los hiper-parámetros óptimos, es decir, aquellos que mejor funcionan para nuestro set de datos, lo que se hace es, nuevamente, realizar un entrenamiento y una evaluación de nuestro modelo de ML seleccionando e iterando sobre los distintos hiper-parámetros que tengamos: a esto se le conoce como “*Parameter Tuning*”. Esta iteración puede ser de manera aleatoria (método conocido como *random search*) o completa (*grid search*). Luego de realizar esta iteración se obtiene la configuración de hiper-parámetros más óptima.

Algo a tener en cuenta es que, para realizar la evaluación, no debemos usar los datos del set de prueba, ya que podríamos caer en un caso de overfitting seleccionando los parámetros que funcionan mejor para los datos del set de pruebas, pero tal vez no los parámetros que generalicen mejor. Para evaluar el modelo y realizar dicho “*Parameter Tuning*” lo que necesitamos es dividir el set de entrenamiento original en dos: un set de entrenamiento y un **set de validación**.

La idea es entrenar el modelo con el set de entrenamiento y luego probarlo con dicho set de validación (NO con el set de pruebas) a efectos de encontrar los mejores hiper-parámetros. El set de validación en general es un 20-30 % del set de entrenamiento original.

Por último, una vez hallados los hiper-parámetros más óptimos, lo que haremos es realizar una **evaluación final** sobre el set de pruebas con los hiper-parámetros que encontramos, y de esta manera nos dará el accuracy (o la métrica de evaluación que elegimos) para los datos que el modelo no vió.

No obstante, existen 2 problemas con el esquema anterior:

- Siendo el set de validación siempre el mismo podemos caer en el problema de que los hiper-parámetros que encontremos solo sean óptimos para nuestro set de validación (el cual es un pequeño conjunto de nuestros datos).
- La división del set de entrenamiento en un nuevo set de entrenamiento y un set de validación hace que haya menos datos disponibles para el entrenamiento. Esto resulta un problema, especialmente para conjuntos de datos pequeños, ya que siempre es mejor utilizar el mayor número de datos posible para el entrenamiento.

Para evitar estos 2 problemas utilizamos el método de **Cross Validation** (o validación cruzada), el cual es prácticamente universal para optimizar algoritmos de ML. Este método nos permite evaluar modelos de ML solucionando los 2 problemas mencionados anteriormente previniendo el overfitting. Para mayor detalle de su funcionamiento ver *Cross Validation*.

■ Paso 7: Utilizando nuestro modelo.

Una vez que obtuvimos buenas predicciones en la etapa de evaluación y nuestra métrica de evaluación llegó a ser la mínima deseada, ya podemos afirmar que estamos en condiciones de utilizar nuestro modelo de machine learning entrenado para realizar predicciones con nuevos datos que están fuera del set de entrenamiento y del set de test.

3.2.4.1 Cross Validation.

El método de **Cross Validation**, o validación cruzada[10][31], consiste en entrenar modelos de ML en subconjuntos de los datos de entrada disponibles y evaluarlos con un subconjunto complementario de los datos.

En la la Figura 3.5 podemos observar el funcionamiento del enfoque básico de Cross Validation: **k-fold Cross Validation**. Este método comienza particionando el set de entrenamiento en k bloques (o “folds”). Luego, se realizan varias iteraciones en las cuales entrenamos nuestro algoritmo con $k - 1$ bloques como set de entrenamiento y lo validamos con el bloque restante (el cual sería nuestro set de prueba), para obtener el valor de alguna métrica de evaluación, como por ejemplo el accuracy. Este proceso se repite k veces con el objetivo de que todos los bloques de datos hayan participado alguna vez del set de pruebas.

El resultado de la métrica de evaluación final devuelta por k -fold Cross Validation es el **promedio** de los resultados de dicha métrica de evaluación calculada dentro de las k iteraciones del algoritmo. Esto hay que hacerlo además por cada valor posible para nuestros hiper-parámetros, por lo que, dependiendo de los datos, puede resultar un proceso costoso.

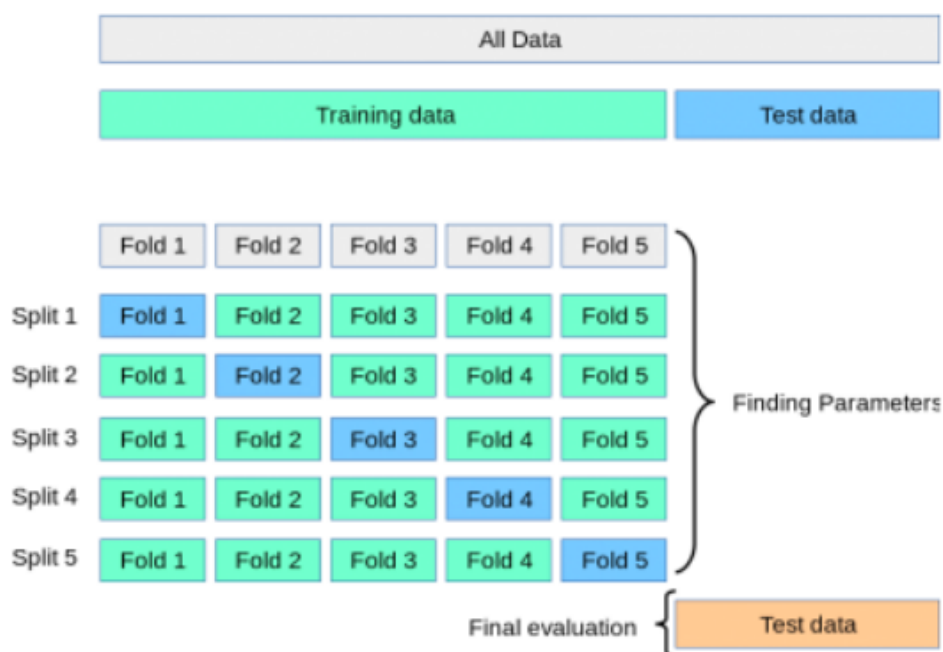


Figura 3.5: k-fold Cross Validation²⁰

En conclusión, luego de realizar dichas iteraciones mediante k -fold Cross Validation, obtendremos la métrica de evaluación final para cada uno de las combinaciones de nuestros hiper-parámetros, y elegiremos los hiper-parámetros más óptimos (el que mejor métrica de evaluación nos haya dado).

Una vez hallados los mismos, el último paso, como comentamos con el primer esquema, es realizar una evaluación final sobre el set de pruebas con los hiper-parámetros que

²⁰Librería Scikit-learn. *Cross-Validation: evaluating estimator performance*. https://scikit-learn.org/stable/modules/cross_validation.html. (Consultado el 30 marzo de 2022).

encontramos, y de esta manera nos dará el accuracy (o la métrica de evaluación que elegimos) para los datos que el modelo no vió.

Un ejemplo de la utilización de estos esquemas para la evaluación de modelos, lo podemos observar en *Armado del modelo de clasificación KNN*.

3.2.4.2 Los Problemas de ML: Overfitting y Underfitting.

En esta sección vamos a nombrar y explicar algunos de los problemas clave en tareas de ML: overfitting y underfitting. Podemos observar un ejemplo de estos conceptos en la Figura 3.6.

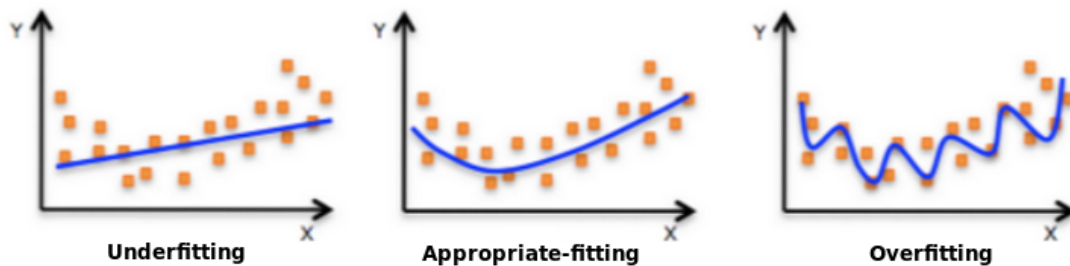


Figura 3.6: Overfitting y Underfitting[10]

Overfitting, o sobre-ajuste, es un problema clave en las tareas de aprendizaje automático supervisado. Es el fenómeno que se detecta cuando un algoritmo de aprendizaje se ajusta/entrena tan bien al set de datos de entrenamiento que se memorizan el ruido y peculiaridades específicas de los datos de entrenamiento. Entonces, se vuelve difícil para el modelo generalizar a nuevos ejemplos que no estaban en el conjunto de entrenamiento. Es por esto que la precisión del modelo cae cuando se prueba en un conjunto de datos desconocido. De esta manera, termina siendo mayor la precisión en el entrenamiento que la precisión sobre los set de pruebas.

La cantidad de datos utilizados para el proceso de aprendizaje es fundamental en este contexto. Los conjuntos de datos pequeños son más propensos al overfitting que los conjuntos de datos grandes. El ajuste excesivo de los datos de entrenamiento conduce al deterioro de las propiedades de generalización del modelo y da como resultado un rendimiento o precisión del mismo poco fiable[37].

El concepto de overfitting está asociado a la complejidad del modelo. Un modelo excesivamente complejo puede ajustar tan bien como queramos al set de entrenamiento pero funcionar muy mal para el set de test[10].

En cambio **Underfitting**, o sub-ajuste, es el opuesto de Overfitting. Esto ocurre cuando el modelo es incapaz de capturar la variabilidad de los datos. Se produce cuando el modelo es demasiado simple. Si nuestros puntos están distribuidos en forma curva, un modelo lineal es demasiado simple, no tiene el poder expresivo necesario para representar correctamente el set de entrenamiento[10].

Por lo tanto, podemos decir que el modelo óptimo es aquel que tiene la complejidad necesaria para capturar lo que los datos expresan pero no más.

3.3 K-Nearest Neighbor (KNN).

K-Nearest Neighbor o K Vecinos más cercanos (KNN), es un algoritmo ML que se puede usar tanto en tareas de regresión como de clasificación. En nuestra implementación, lo usaremos para resolver un problema de clasificación (ver *Clasificación*). Como mencionamos anteriormente, en los problemas de clasificación la variable que se intenta predecir es discreta. En este tipo de problemas contamos con un set de entrenamiento en el cual para cada dato conocemos la clase a la cual pertenece el mismo, y queremos construir un modelo que nos permita clasificar automáticamente datos nuevos cuya clase desconocemos.

Como lo dice su nombre, el algoritmo KNN se basa en encontrar para un determinado punto m , sus K -vecinos más cercanos. Esto es asumiendo que nuestro set de datos está formado por un conjunto de m puntos en n dimensiones siendo todos los valores numéricos.

Para poder utilizar KNN hay que definir previamente dos hiper-parámetros:

- La métrica a usar para calcular las distancias.
- El valor de k , es decir cuantos vecinos vamos a considerar.

Algo a destacar, es que KNN es un tipo de aprendizaje basado en la memoria, también llamado aprendizaje basado en instancias, que pertenece al aprendizaje perezoso (“lazy learning”). Esto quiere decir que KNN **no tiene una fase de entrenamiento**. ¿Qué quiere decir que KNN no tenga un proceso de entrenamiento?, y entonces en *Armado del modelo de clasificación KNN* ¿qué se hace realmente al utilizar la función `.fit()` provista por sklearn?. A continuación se contestarán estas preguntas:

- **A nivel conceptual**, entrenar un clasificador significa tomar un conjunto de datos como entrada, y obtener a la salida un modelo clasificador identificado con distintos parámetros, los cuales son obtenidos en esta etapa de entrenamiento mediante iteraciones realizando cálculos numéricos o resolviendo problemas de optimización (como por ejemplo la obtención del mejor hiperplano en SVM o el ajuste de los pesos en redes neuronales artificiales). En el caso de KNN, el clasificador no se obtiene luego de iterar y obtener dichos parámetros, sino que se identifica por los propios datos de entrenamiento. Entonces, conceptualmente, entrenar un clasificador KNN simplemente requiere almacenar el conjunto de entrenamiento.
- **A nivel de implementación**, evaluar un clasificador KNN en un nuevo punto de datos requiere buscar sus vecinos más cercanos en el conjunto de entrenamiento, lo que puede ser una operación costosa cuando nuestro conjunto de entrenamiento es grande. Existen varios métodos para acelerar esta búsqueda, que generalmente funcionan creando estructuras de datos basadas en el conjunto de entrenamiento. La idea general, es que parte del trabajo computacional necesario para clasificar nuevos puntos es común en todos los puntos. Por lo tanto, este trabajo se puede hacer con anticipación y luego reutilizarse, en lugar de repetirse para cada nueva instancia. De esta manera, en la fase de entrenamiento, al llamar a la función `.fit()` de sklearn, lo que se hace internamente es guardar todo el set de entrenamiento

completo mediante estructuras de datos que nos permitan minimizar los futuros cálculos de medición de distancias. Estas estructuras de datos suelen ser árboles kd (*KD tree*) o árboles de bolas (*Ball tree*). De esta manera, posteriormente a utilizar `.fit()` utilizaremos `.predict()`, y en este paso el cálculo de medición de distancias para el nuevo punto a clasificar será mucho más rápido.

3.3.1 Principal limitación KNN.

Una de las principales limitaciones de KNN[38] es la **complejidad computacional requerida al trabajar con datasets de gran tamaño**.

Esto ocurre debido a que, como mencionamos previamente, cada vez que se va a realizar una predicción para un nuevo punto del set de test, es necesario calcular las distancias entre este punto y todos los demás puntos del set de entrenamiento completo. Dichos cálculos llevan un gran costo computacional si se trata de un dataset de grandes dimensiones.

Además, KNN requiere almacenar todos los datos de entrenamiento para funcionar, por lo que también existe un gran costo computacional en términos de almacenamiento. Es por esto que se recomienda utilizar datasets de pocas dimensiones para que el clasificador KNN complete su ejecución rápidamente.

3.3.2 Funcionamiento y ejemplo de KNN.

Luego de haber entrenado / construido nuestro modelo KNN habiendo previamente determinado el valor de k y la métrica de distancia a utilizar, el funcionamiento del algoritmo KNN para la predicción o clasificación de nuevas muestras es el siguiente:

- Paso 1: Dado un nuevo punto de entrada del set de pruebas, se calculan las distancias entre este nuevo punto y todos los puntos del set de entrenamiento.
- Paso 2: Se ordenan las distancias y se determinan los K vecinos más cercanos basándose en los valores mínimos de dichas distancias.
- Paso 3: Se analiza la clase de esos vecinos y se asigna una clase para ese punto de entrada basado en el voto de la mayoría.
- Paso 4: Se retorna la clase predicha.

A modo de ejemplo, veamos un caso en el cual usamos KNN para clasificación usando la distancia euclidiana como métrica para calcular las distancias.

Como podemos ver en la Figura 3.7, el conjunto de datos está etiquetado: uno es un cuadrado azul y otro es un triángulo rojo. El círculo verde es el punto que necesitamos clasificar.

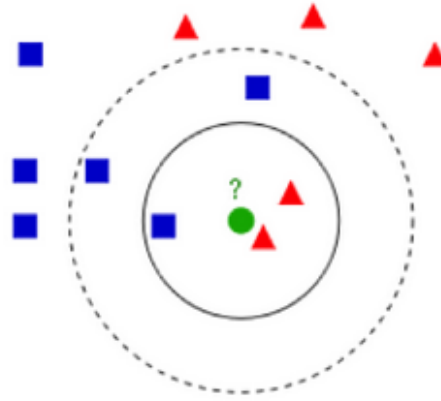


Figura 3.7: Ejemplo de Clasificación con KNN con $K=3$ (círculo con línea sólida) y $K=5$ (círculo con línea punteada), usando la distancia euclidiana como métrica de cálculo de distancias[39].

Viendo la Tabla 2 observamos que:

- Si $K = 1$, tomamos únicamente el punto más cercano del círculo verde. Este punto es un triángulo rojo. Como es el único punto que vota, el punto verde a clasificar pertenece al triángulo rojo. De esta manera la probabilidad de que sea rojo es 1 y de que sea azul es 0.
- Si $K = 3$, entonces hay 2 triángulos rojos y 1 cuadrado azul más cercano al punto verde. Estos 3 puntos votan, por lo que el punto verde a clasificar pertenece al triángulo rojo. De esta manera la probabilidad de que sea rojo es $2/3$ y de que sea azul es $1/3$.
- Si $K = 5$, entonces hay 2 triángulos rojos y 3 cuadrados azules más cercanos al punto verde. Estos 5 puntos votan, por lo que el punto verde a clasificar pertenece al cuadrado azul. De esta manera la probabilidad de que sea rojo es $2/5$ y de que sea azul es $3/5$.

K	P(Azul)	P(Rojo)
1	0	1
3	$1/3$	$2/3$
5	$3/5$	$2/5$

Tabla 2: Probabilidad de pertenecer a cada clase para diferentes valores de k .

Por lo tanto, podemos ver que KNN se basa esencialmente en un método estadístico. Cuando queremos simplemente clasificar un punto cuya clase no conocemos no hacen falta las probabilidades; podemos simplemente asignarlo a la clase con mayoría entre los k -vecinos del punto.

3.3.3 Métrica de distancia a emplear.

La función de distancia entre dos vectores x e y es una función $d(x, y)$ que define la distancia entre ambos vectores como un número real no negativo. Esta función es considerada como una métrica si satisface las siguientes 4 propiedades[39]:

1. Valor no-negativo: La distancia entre x e y siempre es un valor mayor o igual a cero.

$$d(x, y) \geq 0$$

2. Identidad de los indiscernibles: La distancia entre x e y es igual a cero si y sólo si x es igual a y .

$$d(x, y) = 0, \text{ si } x = y$$

3. Simetría: La distancia entre x e y es igual a la distancia entre y y x .

$$d(x, y) = d(y, x)$$

4. Desigualdad triangular: Considerando la presencia de un tercer punto z , la distancia entre x e y es siempre menor o igual que la suma de la distancia entre x y z y la distancia entre y y z .

$$d(x, y) \leq d(x, z) + d(y, z)$$

De esta manera, puede usarse cualquier métrica para medir las distancias en KNN, siempre y cuando cumpla con las propiedades descritas anteriormente.

Dos de las distancias más utilizadas en la implementación de KNN son la distancia euclidiana y la distancia Manhattan. En Figura 3.8 podemos observar gráficamente las diferencias entre las mismas.



Figura 3.8: Distancia Manhattan vs Distancia Euclidiana[10].

La **distancia euclidiana**, o también conocida como Norma L2 o distancia de regla, es una extensión del Teorema de Pitágoras. Esta distancia representa la raíz de la suma al cuadrado de las diferencias absolutas entre los valores opuestos de los vectores. Considerando x e y como vectores a los cuales queremos calcular su distancia euclidiana, su cálculo se obtiene mediante la Fórmula 1.

$$d(x, y) = \sqrt{\sum_{i=1}^n |x_i - y_i|^2} \quad (\text{Ver [39]}) \quad (1)$$

Donde $x = (x_1, x_2, \dots, x_n)$, $y = (y_1, y_2, \dots, y_n)$, y n = dimensiones de los puntos.

En la Fórmula 2 y Figura 3.9 podemos observar el cálculo de la distancia euclidiana entre x e y considerando un espacio bidimensional. Este es el cálculo que se realiza en la implementación de nuestro KNN, ya que utilizamos la distancia euclidiana como métrica, y únicamente contaremos con 2 ejes.

$$d(x, y) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2)$$

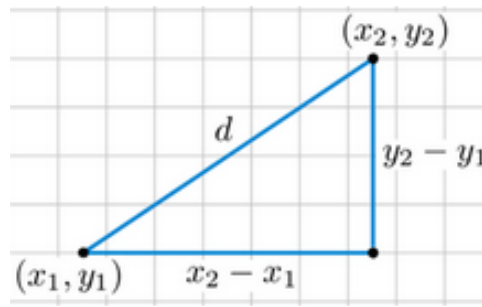


Figura 3.9: Distancia Euclidiana en 2 dimensiones.

En cambio, la **distancia Manhattan**²¹, o también conocida como Norma L1 o distancia rectilínea, se calcula como la sumatoria de la diferencia absoluta entre los valores opuestos de los vectores. Considerando x e y como vectores a los cuales queremos calcular su distancia Manhattan, su cálculo se obtiene mediante la Fórmula 3.

$$d(x, y) = \sum_{i=1}^n |x_i - y_i| \quad (\text{Ver [39]}) \quad (3)$$

Donde $x = (x_1, x_2, \dots, x_n)$, $y = (y_1, y_2, \dots, y_n)$, y n = dimensiones de los puntos.

²¹Su nombre es debido a que es la forma de calcular distancias en una ciudad con forma de grilla en la cual solo nos podemos mover por las calles en forma horizontal y vertical

3.3.4 Eligiendo el valor de k : overfitting y underfitting.

En el caso de KNN el único hiper-parámetro que manejamos -además de la métrica usada para la distancia-, es k . La elección del valor k tendrá un impacto significativo en los resultados del algoritmo.

Para determinar el valor óptimo para k lo que se hace es probar diferentes valores de k y ver cual es el que nos da mejores resultados. Aquí hay que tener un cierto cuidado ya que hay que entender qué implica aumentar o disminuir la cantidad de vecinos más cercanos. Veamos en la Figura 3.10 qué pasa cuando usamos $k = 1$, es decir cuando a cada punto lo clasificamos únicamente en base al punto más cercano.

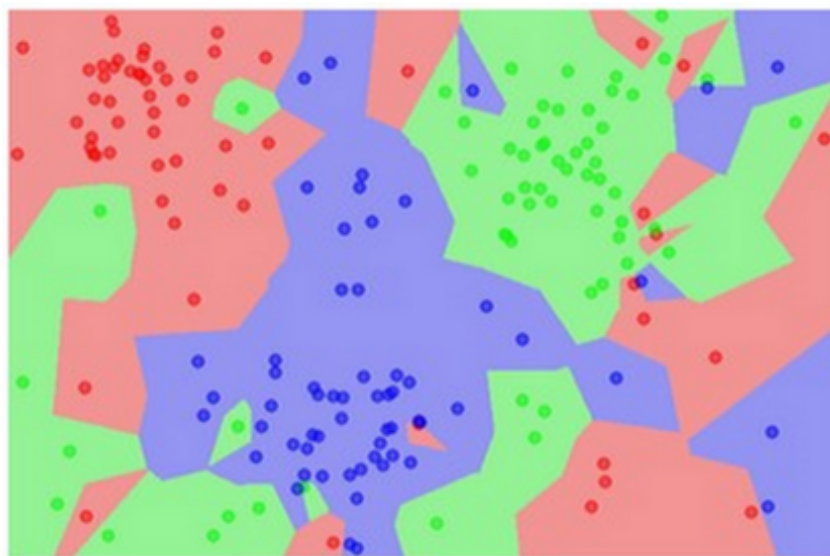


Figura 3.10: KNN con $k = 1$. [10]

En este caso nuestro set de entrenamiento en KNN es simplemente perfecto porque dado un registro cualquiera, el más parecido es el registro mismo y, por lo tanto, vamos a tener 100 % de precisión para el set de entrenamiento. Sin embargo, esto no quiere decir que el algoritmo generalice bien y para el set de pruebas los resultados pueden ser catastróficos.

Lo que podemos ver es que la distribución que aprende nuestro clasificador no es homogénea, es decir que hay puntos de distintas clases mezclados en zonas en las cuales predomina otra clase, esto implica que nuestro clasificador va a funcionar muy bien para el set de entrenamiento pero no aprendió a generalizar y esto quiere decir que no va a ser muy bueno para predecir la clase de puntos nuevos que no hayamos observado en el set de entrenamiento. Esta es la definición de overfitting que discutimos en la sección previa (Ver *Los Problemas de ML: Overfitting y Underfitting*): cuando un algoritmo funciona bien para el set de entrenamiento y mal para datos nuevos. El concepto es que aprender a predecir el set de entrenamiento no es lo mismo que aprender a generalizar.

Ahora veamos en la Figura 3.11 qué sucede con $k = 5$.

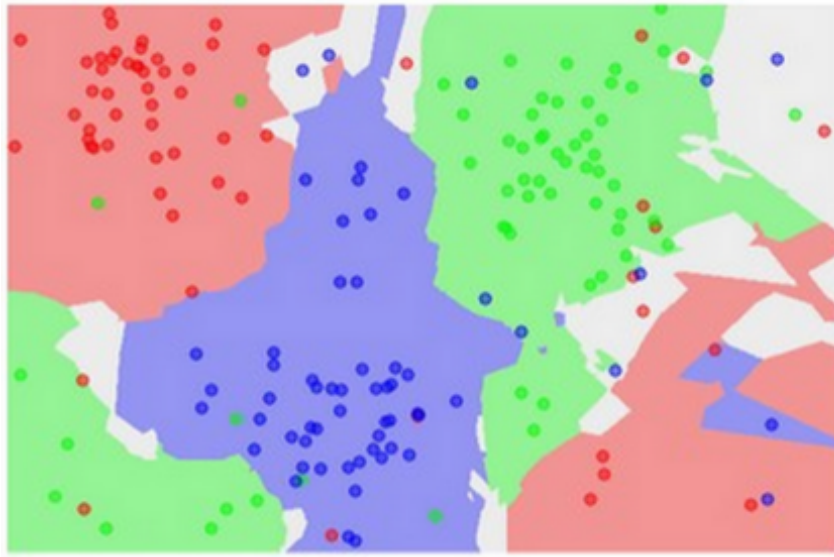


Figura 3.11: KNN con $k = 5$. [10]

Como podemos ver en la Figura 3.11 el clasificador aprendió a generar áreas más suaves y por consiguiente es un clasificador que generalizará mejor para predecir puntos que no estaban en el set de entrenamiento. Podríamos pensar entonces que es conveniente usar valores de k grandes como $k = 1000$ o incluso $k = n$ (donde n es la cantidad de puntos de datos del set de entrenamiento). Sin embargo, al aumentar el valor de k estamos dando cada vez mayor peso a las clases que tienen mayor cantidad de puntos en el set de entrenamiento, en el extremo si $k = n$ vamos a predecir para todos los puntos la clase que mayor cantidad de puntos tiene en el set de entrenamiento lo cual no es bueno.

En resumen:

- Cuando k es muy bajo, el algoritmo toma muy pocos puntos para clasificar un punto nuevo. Es un caso de overfitting. El modelo clasifica los puntos nuevos en base a muy poca evidencia. Nuestro algoritmo alucina, las fronteras se vuelven muy complejas, demasiado complejas para el set de datos que tenemos.
- Cuando k es un número muy grande, KNN considera demasiados puntos para clasificar un punto nuevo. Este es un caso de underfitting, donde nuestro algoritmo tiene visión borrosa por estar mirando demasiados puntos, las fronteras entre nuestras clases se vuelven difusas y el poder expresivo del algoritmo es pobre.

En definitiva, el k óptimo en KNN es aquel que nos de un buen desempeño en cuanto a la precisión de clasificación para el mayor k posible. Una de las mejores formas para seleccionar nuestro k óptimo, es utilizando el método de **Cross Validation** (ó validación cruzada): ver *Cross Validation*.

3.4 K-means.

K-means o K-medias, o también conocido como algoritmo de Lloyd, es un algoritmo de ML de clustering (ver *Agrupación (clustering)*). Como mencionamos anteriormente, en problemas de clustering contamos con datos que queremos dividir en grupos de forma automática. Estos datos no tienen etiquetas, no sabemos a qué grupo pertenecen. K-means nos permite encontrar cómo repartir m puntos en n dimensiones dentro de k clusters.

El nombre de K-means viene porque representa cada uno de los clusters por la media de sus puntos, es decir, por su centroide. Cada clúster, por tanto, es caracterizado por su centroide que se encuentra en el centro de los elementos que componen el clúster.

Para este algoritmo, el único hiper-parámetro que se necesita definir previamente es k , el número de clusters. Este k se puede obtener de distintas formas, en nuestra implementación utilizaremos el *método del codo* para calcularlo. Antes de explicar en qué consiste este método, detallaremos el funcionamiento de k-means, su objetivo, sus desventajas y las mejoras que se pueden aplicar.

3.4.1 Funcionamiento y ejemplo de K-means.

Resumidamente, habiendo definido previamente nuestro k , el número de clusters, y teniendo como entrada nuestros puntos de datos x_1, \dots, x_m , el funcionamiento de K-means consta de los siguientes 4 pasos [42]:

- Paso 1: Se inicializan aleatoriamente los centroides C_1, \dots, C_k colocándose en posiciones al azar. Cada centroide C_j pertenece a un cluster j específico.
- Paso 2: Se asignan los puntos de datos a su cluster más cercano: Para esto, lo que se hace es, para cada punto de datos x_i , se calculan las distancias euclidianas con cada uno de los centroides C_j . Luego, se asigna cada punto de datos x_i al cluster j , cuyo centroide C_j es el más cercano al punto x_i basándose en la mínima distancia euclidiana. Matemáticamente, se utiliza la Fórmula 1 para calcular la distancia euclidiana entre cada uno de nuestros puntos de datos x_i , y cada uno de nuestros centroides C_j , y obtener el mínimo valor. Reformulando dicha ecuación, nos quedaría la Fórmula 4.

$$d(x, C_j) = \sqrt{\sum_{i=1}^d |x_i - C_{ji}|^2} \quad (\text{Ver [40]}) \quad (4)$$

Donde d = dimensiones de nuestros puntos de datos, $x_i = (x_1, x_2, \dots, x_d)$ la posición de un punto x de nuestro set de datos para la dimensión i , y $C_{ji} = (C_{11}, C_{12}, \dots, C_{kd})$ la posición de nuestro centroide C del cluster j para la dimensión i .

En la Fórmula 4, podemos visualizar que los centroides tienen igual cantidad de dimensiones que nuestros puntos y, además, pueden o no coincidir con puntos de los datos. Un centroide que es también un punto de los datos se lo llama “clusteroid”. En general, no se pide que los centroides sean también puntos sino que se permite que tomen cualquier posición dentro del espacio de los datos.

- Paso 3: Se actualizan las posiciones de los C_j (centroides C pertenecientes a los clusters j). De esta manera, se recalculan los centroides como el promedio de todos sus puntos (los que pertenecen a su cluster) de forma tal de minimizar la distancia desde el centroide a los puntos asignados al mismo. Como resultado, los centroides se mueven al centro promedio de los puntos a los que se les asignaron. Matemáticamente, se utiliza la Fórmula 5 para calcular, para cada cluster $j = (1, \dots, k)$ la nueva posición del centroide C_j , el cual se calcula como el promedio de todos los puntos x_i asignados al cluster j en el paso anterior.

$$C_j(a) = \frac{1}{m_j} \sum_{x_i \in C_j} x_i(a), \text{ para } a = (1, \dots, d) \quad (5)$$

Donde d = dimensiones de nuestros puntos de datos y centroides, m_j = cantidad de puntos de nuestro set de datos pertenecientes al cluster j , $x_i(a)$ = posición de nuestro punto del set de datos x_i para la dimensión a , y $C_j(a) = (C_1(1), C_1(2), \dots, C_K(D))$ la posición de nuestro centroide C del cluster j para la dimensión a .

- Paso 4: Se repiten los pasos 2 y 3 hasta que los centroides convergen. La convergencia puede verificarse mediante la diferencia entre los centroides entre el paso anterior y el actual. Cuando los centroides prácticamente ya no cambian de posición o cuando los puntos de datos no cambian de cluster se declara la **convergencia** del algoritmo.

A modo de ejemplo, veamos un caso en el cual usamos K-means para clusterizar nuestros datos hasta llegar a su convergencia.

En el ejemplo de la Figura 3.12, nuestros datos tienen 2 dimensiones. Definiendo previamente nuestro $k=2$ (2 clusters), en el ítem (1) observamos dicha distribución de datos. Además, en este ítem se realiza la inicialización aleatoria de nuestros 2 centroides (triángulos rojo y amarillo). Todo esto involucra al **paso 1** del algoritmo k-means.

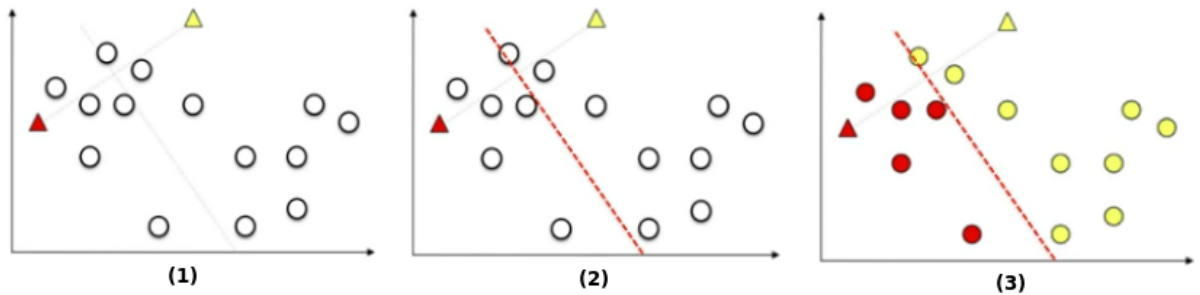


Figura 3.12: Algoritmo K-means.

En el **paso 2**, lo que se tiene que hacer es, para cada uno de nuestros puntos de datos, averiguar a cuál centroide se asignará, basándonos en la distancia más cercana al mismo. Al considerar la distancia euclidiana para este objetivo, en (2) dibujamos una línea roja imaginaria que divide en iguales distancias nuestros 2 centroides: partiendo de esta línea la distancia a ambos centroides es la misma. De esta manera, cada punto ubicado a la derecha de la línea pertenecerá al cluster amarillo (por tener menor distancia al centroide

triángulo amarillo), y cada punto ubicado a la izquierda de la línea pertenecerá al cluster rojo (por tener menor distancia al centroide triángulo rojo). Esta asignación la observamos en el ítem (3), que representa nuestra asignación inicial de nuestros puntos en los clusters.

La línea roja es simplemente una referencia utilizada para una mejor comprensión gráfica, pero internamente esta asignación de puntos se realiza en base al cálculo de las distancias euclidianas mediante la Fórmula 4. descrita anteriormente para el paso 2 de k-means.

Posteriormente pasamos al **paso 3** de k-means: la actualización de las posiciones de los centroides de los clusters. Lo que hacemos en (4) es mover los centroides al centro promedio de los puntos a los que se les asignaron. Para esto se utiliza la Fórmula 5 descrita anteriormente. Podemos observar las nuevas posiciones de los centroides en (5). De esta forma, finalizamos la primera iteración de nuestro algoritmo.

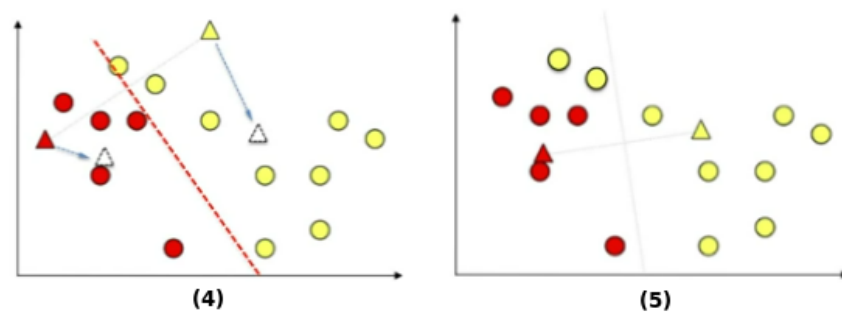


Figura 3.13: Algoritmo K-means.

En la próxima iteración, tenemos la misma disposición de los puntos, solo que ahora tenemos dos nuevas posiciones para nuestros centroides. Ignoramos las clases que se asignaron para los puntos de nuestros datos ya que esta era una asignación del paso anterior: ahora nuestros centroides cambiaron y la asignación de clases también lo hará. Nuevamente trazamos la línea imaginaria (6) y realizamos la nueva asignación: los únicos puntos que cambiaron con respecto al paso anterior son los 2 amarillos ubicados en la parte superior, los cuales pasan a ser rojos (7). Debido a que cambiaron de clase, entonces nuevamente vamos a actualizar las posiciones de nuestros centroides (8), quedándonos distribuidos finalmente como (9). De esta manera finalizamos la segunda iteración.

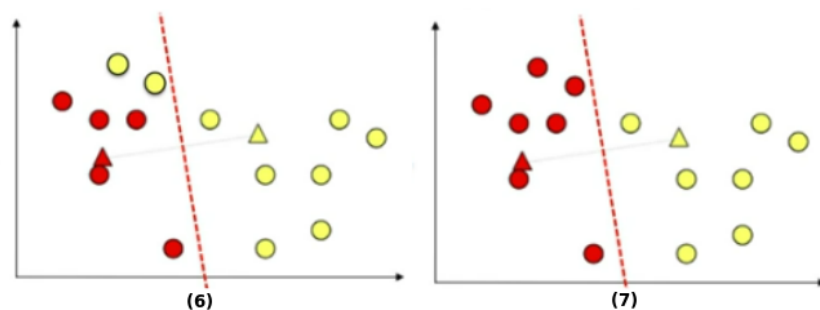


Figura 3.14: Algoritmo K-means.

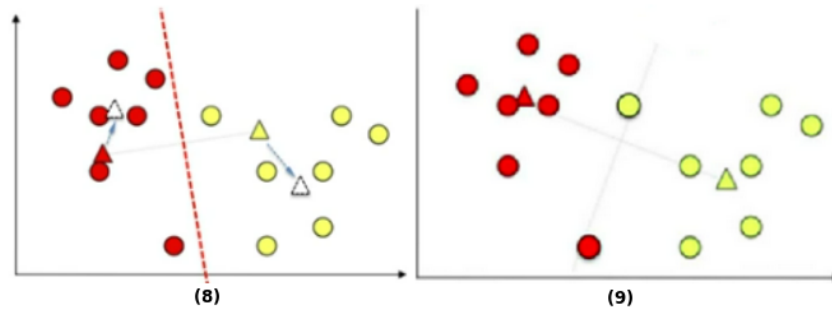


Figura 3.15: Algoritmo K-means.

Para la tercera iteración se vuelven a repetir los pasos 2 y 3: trazamos la línea de división de nuestros datos **(10)**, realizamos la nueva asignación de puntos **(11)** y observamos que cambiaron 2 puntos de clase. Debido a que cambiaron de clase, entonces re-ubicamos los centroides nuevamente **(12)**, quedándonos finalmente como **(13)**. De esta manera finalizamos la tercera iteración.

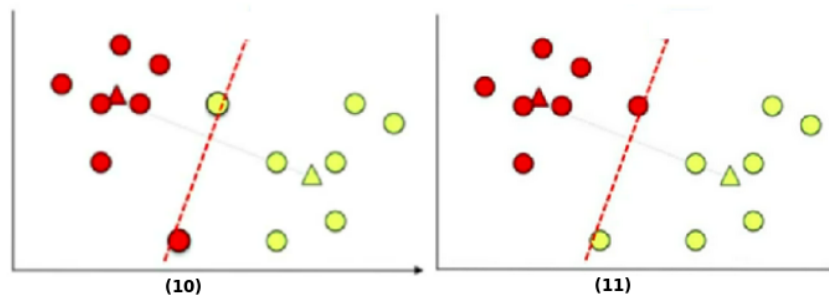


Figura 3.16: Algoritmo K-means.

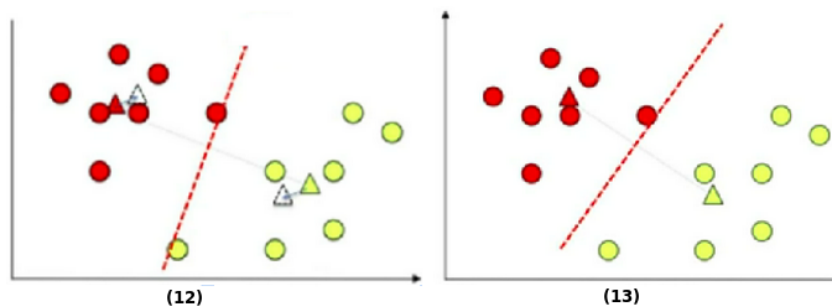


Figura 3.17: Algoritmo K-means.

Finalmente, en el ítem **(13)**, si ahora intentamos asignar nuevamente nuestros puntos de datos, observamos que ninguno cambiaría de color/clase. Los puntos rojos ya están del lado izquierdo de la línea, y los puntos amarillos ya están del lado derecho de la línea. Esto significa que si seguimos iterando nada cambiará, por lo que llegamos a la **convergencia** de nuestro algoritmo.

3.4.2 Objetivo de k-means y su función de coste.

El objetivo de K-means es encontrar cómo repartir m puntos en n dimensiones dentro de k clusters de la “mejor forma posible”.

¿A qué llamamos “de la mejor forma posible”? se puede definir a cada cluster a partir de su centro (centroide), en cuyo caso la mejor distribución posible es aquella que minimiza la distancia entre cada punto del cluster y el centroide que se le asignó. Esto es lo que se conoce como **Inercia** (Inertia) o **WCSS** (within-cluster sum-of-squares, suma de cuadrados intra cluster), y es lo que busca *minimizar* K-means.

A este criterio de Inercia también se lo considera como la **función de coste** J^{22} de K-means, la cual se intenta *minimizar*.

Dicho de otra manera, el objetivo de k-means es encontrar la ubicación de los centroides que minimicen la inercia, esta será la posición óptima de los mismos.

La Inercia[40] se puede reconocer como una medida de la coherencia interna de los clústeres, permitiendo medir qué tan bien K-means realiza la agrupación del conjunto de datos. La misma se calcula como la sumatoria de las diferencias de las distancias (generalmente euclidiana) al cuadrado entre cada uno de los puntos de datos x_i pertenecientes a un cluster j y el centroide al que fue asignado dicho punto, C_j : Fórmula 6.

$$J = Inertia = WCSS = \sum_{C_j}^{C_k} \left(\sum_{x_i \in C_j}^{x_m} ||x_i - C_j||^2 \right) \quad (6)$$

Donde k = número de centroides ó clusters, m = cantidad de puntos de nuestro set de datos, C_j = centroide de cluster j , y x_i = punto i de nuestro set de datos asignado a C_j . Además, el término $||x_i - C_j||$ se corresponde a la distancia euclidiana entre cada punto del set de datos, x_i , y el centroide que tiene asignado, C_j . Este cálculo se corresponde a la Fórmula 4 que obtuvimos anteriormente.

Considerando una inicialización aleatoria de nuestros centroides, y por ejemplo, definiendo que nuestro número de clusters es 3 (k en kmeans = 3), entonces la Fórmula 6 se podría reescribir como la Fórmula 7. Podemos observar que al variar la inicialización de nuestros centroides, las posiciones de nuestros C_j cambiarían, por lo que nuestra WSS sería distinta.

$$WCSS = \sum_{x_i \in C_1} ||x_i - C_1||^2 + \sum_{x_i \in C_2} ||x_i - C_2||^2 + \sum_{x_i \in C_3} ||x_i - C_3||^2 \quad (\text{Ver [41]}) \quad (7)$$

²²La función de coste o también llamada función de distorsión u objetivo, es una función que trata de determinar la diferencia o el error entre el valor estimado / predicho por un modelo de machine learning y el valor real, con el fin de optimizar los parámetros en dicha función y obtener un error mínimo. En el caso de k-means, nuestros valores estimados son las posiciones de los centroides -los cuales tenemos que descubrir y encontrar los más óptimos / los que minimicen nuestra J - y nuestros valores reales son nuestros puntos del cluster.

Es importante entender la Fórmula 6 para comprender el problema genérico de clustering. Se sabe que hay k centroides y que cada uno de estos centroides puede estar en cualquier punto del espacio. El objetivo de k -means es **encontrar la posición óptima para estos centroides de forma tal de minimizar la distancia total entre los puntos y los centroides que se le han asignado**. Es evidente que cada punto debe estar asignado a su centroide más cercano para minimizar la distancia por lo que el problema puede resumirse a **encontrar la posición óptima para los k centroides**.

Obtener el mínimo valor de nuestro $WCSS$, o que es lo mismo que encontrar el mínimo global, es un proceso muy complejo debido a la inmensa cantidad de formas en las que nuestros m puntos de datos se pueden repartir en k clusters. En lugar de esto, k -means trata de encontrar una solución que, aun no siendo la mejor de entre todas las posibles, sea buena (óptimo local) y garantice un agrupamiento en el que los clusters sean poco dispersos y se encuentren separados entre sí.

Es de destacar que, suponiendo que tenemos m puntos de datos y especificamos que este m sea nuestro número de clusters ($k = m$), entonces $WCSS$ será 0 ya que cada punto de nuestros datos actuará como centroide de sí mismo y la distancia euclidiana entre ellos será nula. De esta forma, cada cluster contendrá solo un punto. Esto idealmente es un cluster perfecto, pero no tiene ningún sentido tener tantos clusters como puntos de datos tengamos. Por lo tanto, existe un valor de umbral para k que podemos encontrar utilizando el *método del codo*, el cual veremos más adelante en *Obtención del k mediante Elbow Method*).

3.4.3 Posición inicial de los centroides.

Al ser un algoritmo que encuentra un mínimo local, K -means es muy sensible a la posición inicial de los centroides. Una posible solución para no caer en un mínimo local malo es realizar varias ejecuciones de K -means con inicializaciones aleatorias para los centroides, idealmente ejecuciones en paralelo, y computar la función de coste J para cada resultado final quedándonos con la mínima.

La Figura 3.18 muestra un ejemplo donde hay 4 clusters y varios puntos aislados en un plano bidimensional. La Figura 3.19 muestra el resultado del mejor y peor costo. Se puede ver que el resultado del peor costo es realmente muy malo mientras que el resultado del mejor costo es lo que se esperaba obtener.

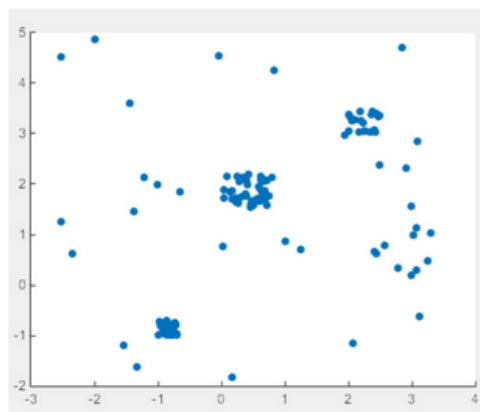


Figura 3.18: Ejemplo K-means. [10]

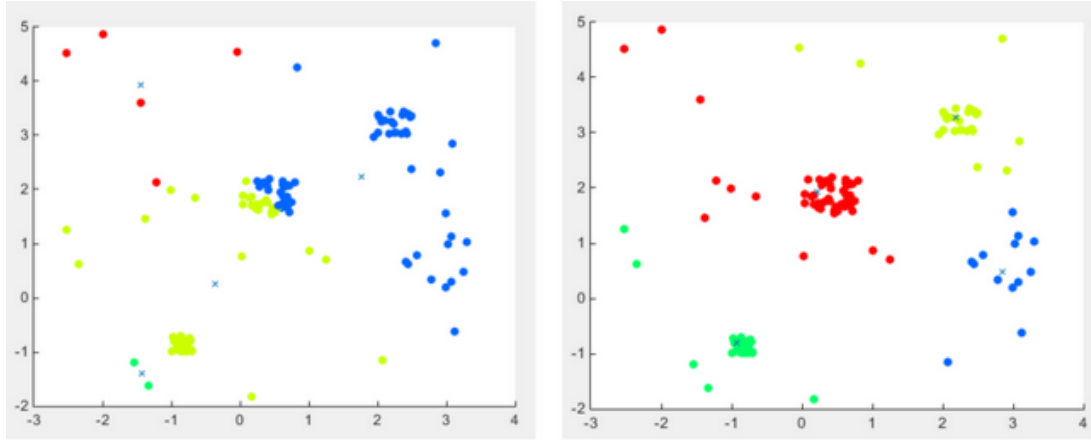


Figura 3.19: Ejemplo K-Means peor (izquierda) y mejor (derecha) costo. [10]

El gráfico de la Figura 3.20 muestra el resultado de la función costo para 100 ejecuciones de K-Means, con un k predefinido, y variando las inicializaciones de nuestros centroides. En el eje x se representan las 100 ejecuciones de k-means, y en el eje y encontramos la función de coste J obtenida para cada ejecución.

En la Figura 3.21 podemos observar un histograma representando la cantidad de veces en porcentajes (eje y) que se observaron diferentes rangos de costo (eje x). Como se puede ver, afortunadamente los costos bajos son amplia mayoría, es decir que únicamente con algunas pocas inicializaciones desafortunadas K-means llega a un mínimo global muy malo. Esto quiere decir que una sola iteración de K-Means nos dará un buen resultado con una probabilidad alta y hacer un ciclo con unas pocas ejecuciones es más que suficiente para llegar a un excelente resultado.

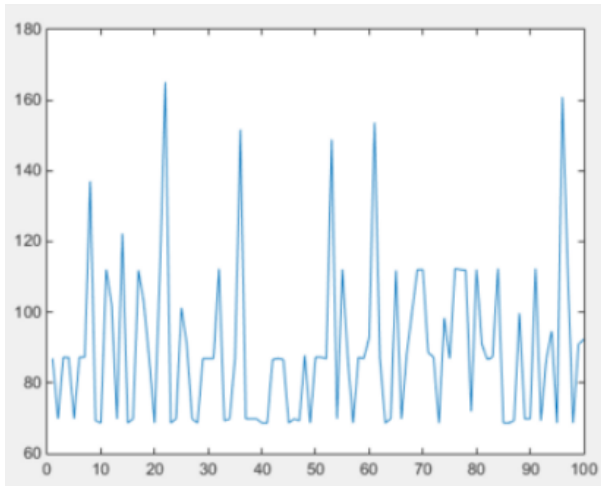


Figura 3.20: Costo en K-Means. [10]

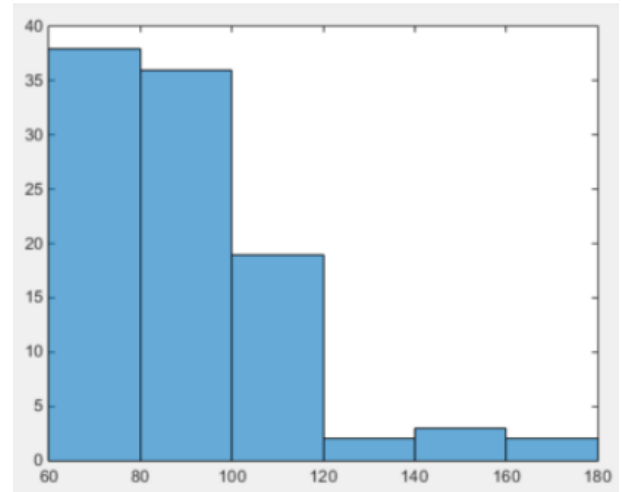


Figura 3.21: Histograma J en K-Means. [10]

3.4.4 Limitaciones K-means.

K-means tiene tres principales limitaciones, las cuales describiremos a continuación.

1. **Outliers:** Un outlier es una observación atípica dentro de una muestra de datos que es notablemente diferente del resto. Los outliers representan errores en la medición, mala recolección de datos, o simplemente muestran variables no consideradas al recolectar los datos. En k-means los outliers pueden incrementar la función de coste o Inercia. Varios investigadores observaron que, cuando los datos contienen outliers, existe una variación en el resultado que significa que no hay un resultado estable al realizar diferentes ejecuciones de k-means con los mismos datos[42]. Por esta razón, es muy importante eliminar los outliers de nuestro conjunto de datos. Los valores atípicos se pueden eliminar aplicando técnicas de preprocesamiento en el conjunto de datos original.
2. **Número de los k clusters:** Uno de los principales inconvenientes de K-means es la necesidad de determinar el número óptimo de los k clusters por adelantado. Esto perjudica la eficacia del algoritmo ya que en la práctica, no se conoce a priori el número de clusters final. Este defecto lo perjudica al compararlo con otros algoritmos, ya que en muchos la inicialización del número de clusters no es necesaria. Todavía es un problema saber cuál es el número correcto de k clusters que debemos asignar[42].
3. **Inicialización de los centroides:** Como vimos en la sección *Posición inicial de los centroides*, la selección inicial de los centroides dirige el proceso de K-means y las particiones resultantes y su efectividad están condicionadas a la elección de estos centroides[40]. Existen numerosos estudios[43] que muestran que la performance de K-means es fuertemente dependiente de la estrategia de inicialización de la ubicación de los centroides a utilizar.

Todas estas limitaciones descritas anteriormente fueron abordadas y resueltas en la implementación de k-means realizada para este proyecto de la siguiente forma:

- Para el primer problema, los outliers, se aplicó un pre-procesamiento de nuestros datos que permitió removerlos: ver *Armado del modelo de clasificación KNN*.
- Para el problema de la determinación óptima y por adelantado del número de los k clusters, se aplicó el *método del codo*: ver *Obtención del k mediante Elbow Method*.
- Para el último caso de inicialización de los centroides, como mencionamos anteriormente, una de las soluciones para no caer en un mínimo local malo es realizar varias ejecuciones de K-means con inicializaciones aleatorias para los centroides, computar la función de distorsión J para cada resultado final y quedarnos con el mínimo.

En la implementación de esta Tesis, se utilizó una estrategia más novedosa, inicializar nuestros centroides utilizando K-means++: ver *Inicialización de los centroides: k-means++*. La ventaja de utilizar esta estrategia es que generalmente funciona mejor y más rápido que la detallada anteriormente, por lo que actualmente es uno de los métodos estándar para inicializar nuestros centroides en k-means. [44]

3.4.4.1 Obtención del k mediante Elbow Method.

Como se mencionó anteriormente, es necesario encontrar el valor óptimo de k . Un buen modelo k-means es uno que tenga la menor Inertia y un bajo número de clusters k . Sin embargo esto no es posible, ya que si k aumenta, Inertia decrece.

Elbow method o *método del codo* nos permite determinar el óptimo valor de k , o dicho de otra forma, el óptimo número de clusters en los cuales nuestros puntos de datos serán clusterizados. Mediante este método lo que se hace es graficar cómo varía la función de costo o Inercia en función de k : recordemos que la Inercia se calcula mediante la Fórmula 6.

Luego de obtener este gráfico, se encuentra el punto “del codo”, el cual indicará nuestro k óptimo. Este es el punto después del cual la Inercia comienza a disminuir de forma lineal, o dicho de otra forma, el punto “del codo” indica que después de este punto, el cambio en la disminución del valor de la Inercia no es significativo.

Veamos el siguiente ejemplo donde obtendremos nuestro k óptimo en base al *método del codo*: para los puntos de datos observados en la Figura 3.22, podemos observar visualmente 3 clusters separados, por lo que el número óptimo de clusters debería ser 3. En este caso es bastante sencillo observar esto, pero hay muchos casos donde no es fácil visualizar a simple vista el número de k óptimo, y es por esto que el *método del codo* es de gran utilidad.

Utilizando esta técnica, graficamos la Inercia en función de k : iteramos valores de k desde 1 hasta 9 y calculamos los valores de inercia para cada valor de k dentro de ese rango, obteniendo el gráfico de la Figura 3.23.

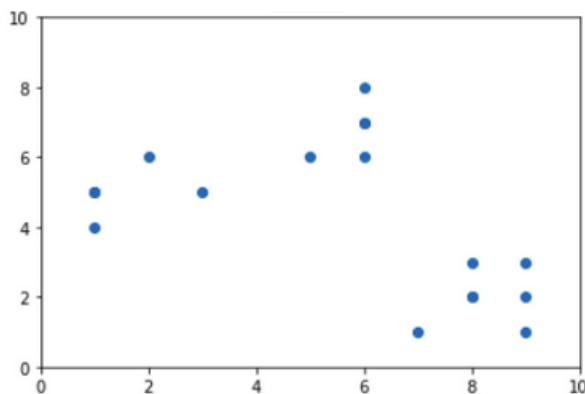


Figura 3.22: Puntos de datos.

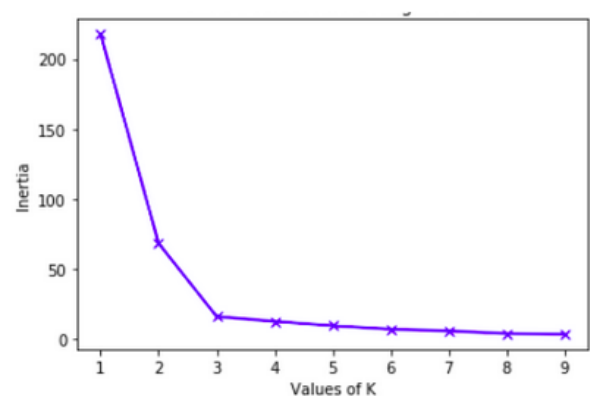


Figura 3.23: Método del codo usando Inercia.

De esta manera, concluimos que, para estos datos, viendo el gráfico de la Figura 3.23, el punto “del codo” está en $k = 3$, siendo 3 nuestro k óptimo.

En la Figura 3.24 podemos visualizar los puntos de datos clusterizados para diferentes valores de k , observando de esta manera que con $k = 3$ obtenemos la clusterización más óptima y lógica.

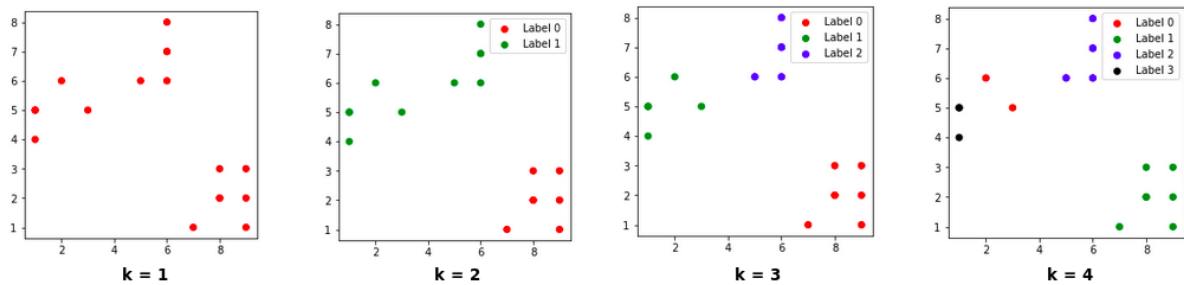


Figura 3.24: Puntos de datos clusterizados para diferentes valores de k .

3.4.4.2 Inicialización de los centroides: k-means++.

K-Means++[44] es una variante de K-Means en donde lo único que cambia con respecto a este es la forma en que se inicializan los centroides (el paso 1 de k-means descrito en la sección *Funcionamiento y ejemplo de K-means*). Actualmente es uno de los métodos estándar para inicializar los centroides en k-means.

De esta manera, habiendo predefinido nuestro k , los pasos para inicializar nuestros centroides con k-means++ son:

1. Se elige un punto al azar como primer centroide. Este paso es el mismo que ocurre en k-means, solo que en este caso únicamente el primer centroide será elegido al azar.
2. Calculamos la distancia $D(x)$ entre cada punto de nuestros datos x y su centroide.
3. Uno de nuestros puntos de datos x será elegido como el nuevo centroide. Este nuevo centroide será el que tenga la mayor distancia al cuadrado $D(x)^2$ con respecto a su centroide, es decir, el punto x que tenga la distancia más lejana con su centroide.
4. Repetimos los pasos 2 y 3 hasta que nuestros k centroides sean asignados.

La idea de K-means++ es asignar los centroides de forma espaciada, de esta forma el óptimo local obtenido por K-Means tiene una mayor probabilidad de estar cerca del óptimo global.

En la secuencia de la Figura 3.25 podemos observar un ejemplo de asignación de centroides por medio de k-means ++, habiendo previamente definido $k = 3$. Observamos que, a partir de la muestra de datos (1), obtendremos la inicialización de nuestros centroides (6).

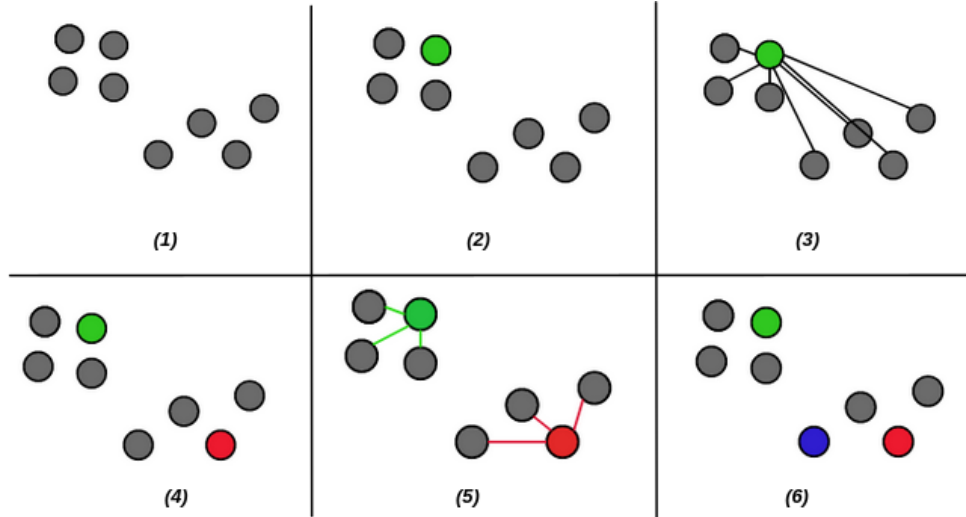


Figura 3.25: Inicialización de centroides mediante K-means.

En (2) se elige aleatoriamente un punto al azar como centroide inicial (en este caso el punto verde). En (3) se calcula la distancia $D(x)$ entre cada uno de los puntos x y su centroide. Para este caso únicamente tenemos el centroide verde. En (4) se encuentra al segundo centroide (rojo) de manera que esté alejado del centroide verde. Este segundo centroide es el punto x que tiene la mayor distancia al cuadrado $D(x)^2$ a su centroide. En (5) se calculan nuevamente las distancias $D(x)$ entre cada uno de los puntos y su centroide más cercano. Por último, en (6) finalizamos la inicialización de nuestros centroides encontrando al último y tercer centroide (azul). El mismo fue elegido ya que su punto x es el que tiene la mayor distancia al cuadrado $D(x)^2$ a su centroide con respecto a cualquier otro punto y su centroide.

De esta manera, concluimos que K-means++ genera una inicialización más dispersa de los centroides iniciales comparándolo con el método de K-means estándar de inicialización aleatoria. El problema de K-Means++ es que para datos realmente masivos necesita hacer k iteraciones sobre los datos para elegir los centroides, tarea que es todavía más ineficiente cuando k es un número largo. Sin embargo, en general, K-Means++ genera una inicialización mejor para K-Means y esto permite no solo obtener una mejor solución final sino que también acelera la velocidad de convergencia del algoritmo.

3.5 Redes Neuronales Artificiales (ANN).

Las redes neuronales artificiales (o artificial neural network, ANN) son técnicas de Machine Learning que permiten imitar o reproducir la capacidad de aprender propia del comportamiento de un cerebro humano. Las ANN están compuestas por un conjunto de elementos simples (neuronas artificiales) que se interconectan masivamente en paralelo y con organización jerárquica[45].

En esta sección detallaremos el funcionamiento básico y los componentes de una red neuronal; la cual será utilizada como base en la implementación del modelo Word2vec para obtener word embeddings: Ver *Word embeddings (Word2vec) & WMD*.

Existen muchísimos modelos de redes neuronales, los cuales pueden ser entrenados para resolver distintas tareas, tanto de clasificación o de regresión, como de clustering. En esta sección se explicarán dos de ellos, el modelo más básico y simple de una red neuronal, conocido como *Perceptrón simple*, y su modelo predecesor, *Perceptrón Multicapa* (o también conocido como Multi Layer Perceptron, *MLP*). Ambos modelos son considerados como algoritmos de clasificación, formando parte de la rama de aprendizaje supervisado.

3.5.1 Perceptrón simple.

El Perceptrón Simple es un algoritmo de clasificación lineal que surge a fines de la década del 50 y fue desarrollado por Frank Rosenblatt. En aquel tiempo se buscaba crear un algoritmo que imitase el funcionamiento del cerebro humano. El perceptrón puede verse biológicamente como una neurona, siendo la unidad básica de procesamiento que se encuentra en una red neuronal. La teoría era que un perceptrón (o neurona) podía programarse como una salida binaria que dependía del resultado de la combinación lineal entre las entradas y pesos asignados a las mismas. De esta forma, la neurona respondía de manera binaria (0 ó 1) según si el resultado de la combinación lineal fuese mayor o menor a un cierto umbral mediante el uso de una “función de activación”[10], la cual detallaremos más adelante.

En resumen, esta neurona (o perceptrón) tiene conexiones con nodos de entrada, a partir de los cuales recibe estímulos externos (valores de entrada). Con estos valores la neurona realiza un procesamiento interno y genera un valor de salida, decidiendo de esta manera a cuál de las dos clases posibles pertenece la observación[45]. Este procesamiento depende de las distintas funciones / cálculos matemáticos llevados a cabo mediante la *función de entrada* y la *función de activación* de la neurona (o también llamada *función de transferencia*)[46]. Podemos observar un diagrama de la arquitectura del Perceptrón Simple en la Figura 3.26.

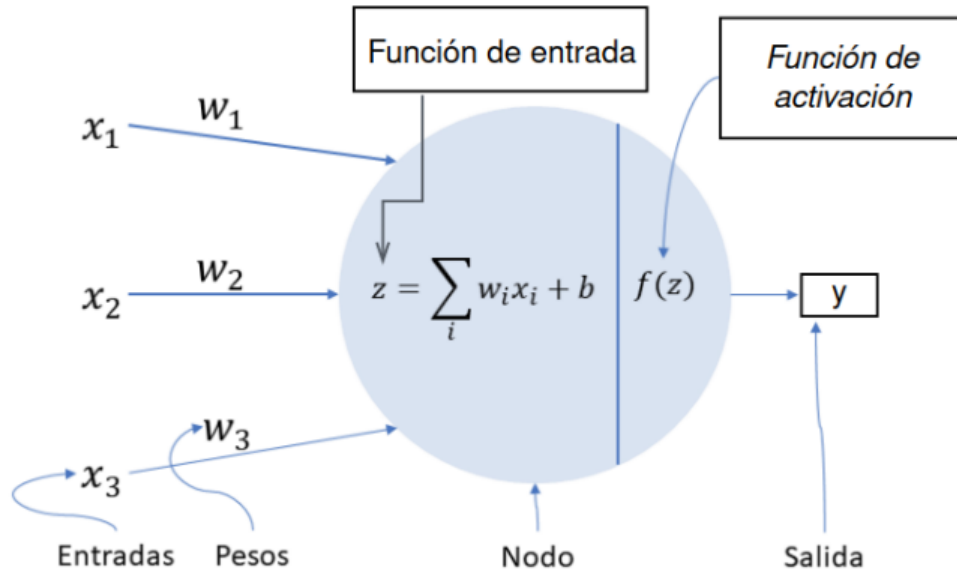


Figura 3.26: Modelo Perceptrón Simple

Mencionaremos las partes involucradas en dicho esquema:

- Entradas (x_1, x_2, x_3, x_n): Las *entradas* son los valores de entrada a la neurona que provienen de estímulos externos.
- Pesos (w_1, w_2, w_3, w_n): Los *pesos* son los coeficientes que determinan la intensidad o fuerza de la conexión de las señales de entrada registradas por la neurona. Si el peso de una entrada x_1 es mayor al peso de la entrada x_2 , entonces la entrada x_1 tiene mayor influencia en la neurona. Si el peso es positivo se habla de una *excitación* de la entrada, en cambio si el peso es negativo se habla de una *inhibición* de la misma.
- Función de entrada: En esta parte del esquema se aplica una función de entrada a nuestros valores de entrada, sus pesos y el bias -representado en la figura como b^{23} -. Existen distintas funciones de entrada, tales como la sumatoria de las entradas ponderadas, la productoria de las entradas ponderadas o el máximo de las entradas ponderadas[47]. En el ejemplo descrito en la figura 3.26 se utilizó la función de entrada más común: la *sumatoria de las entradas ponderadas*, que representa la suma de todos los valores de entrada a la neurona, multiplicados por sus correspondientes pesos. De esta manera, mediante la Fórmula 8 obtenemos la salida Z .

$$Z = \sum_i w_i x_i + b \quad (8)$$

Donde x_i es cada una de las entradas externas a la neurona/perceptrón, w_i son los pesos asociados a cada entrada y b es el sesgo descrito anteriormente.

²³El bias o sesgo es el término independiente que permite cambiar o disparar la función de activación para garantizar un aprendizaje exitoso. En otras palabras, permite controlar qué tan predispuesta está la neurona a disparar un 1 o un 0 independientemente de los pesos. Un sesgo alto hace que la neurona requiera una entrada más alta para generar una salida de 1. Un sesgo bajo lo hace más sencillo.

- Función de activación $f(Z)$:

Las neuronas artificiales tienen diferentes estados de activación, algunas de ellas solamente dos -activa (excitada) o inactiva (no excitada)-, pero otras pueden tomar cualquier valor dentro de un conjunto determinado.

La función activación calcula dichos estados de activación de la neurona, transformando la entrada global z en un valor (estado) de activación o salida y , cuyo rango normalmente va de (0 a 1) o de (-1 a 1). Esto es así, porque una neurona puede estar totalmente inactiva (0 o -1) o activa (1)[47].

Resumiendo, la función de activación tiene como objetivo determinar si la neurona es activada o no aplicando una *función no lineal* f a Z , obteniendo así la salida de la neurona y : ver la Fórmula 9. ¿Por qué esta función de activación tiene que ser una *función no lineal*? Esto lo explicaremos en la sección *Funciones de activación en MLP*.

$$y = f(z) = \begin{cases} 1, & \text{si } z > \text{umbral} \\ 0, & \text{si } z \leq \text{umbral} \end{cases} \quad (9)$$

Por ejemplo, viendo la Figura 3.26, si utilizamos la *función escalón* (o *step*) como nuestra función de activación, entonces nuestro umbral sería = 0 y nuestra salida se podría representar mediante la Fórmula 10.

$$y = f(z) = \begin{cases} 1, & \text{si } \sum_i w_i x_i + b > \text{umbral} \\ 0, & \text{si } \sum_i w_i x_i + b \leq \text{umbral} \end{cases} \quad (10)$$

De esta manera si es 1 se activa la neurona, y si es 0 no se activa. Observemos que este resultado depende de los pesos y las entradas a la neurona, si variamos los mismos nuestras salidas seguramente también lo harán.

Se pueden aplicar otras funciones de activación, las cuales detallaremos en el Anexo: ver sección *Funciones de activación*.

- Salida y : son los valores que permiten retornar los resultados procesados por la red neuronal (en este caso formada por un único perceptrón).

3.5.2 Desventaja del Perceptrón simple.

La desventaja de este esquema es que el Perceptrón Simple es un algoritmo que encuentra un hiperplano separador y no más que esto, por lo que dista bastante de imitar la capacidad de aprender propia de un cerebro humano.

Observando la Figura 3.27, podemos ver que la neurona de tipo perceptrón simple con una función de activación lineal (función identidad) únicamente permite discriminar entre dos clases linealmente separables mediante una única recta o hiperplano (dependiendo del número de entradas), permitiendo solucionar por ejemplo la función OR (que es linealmente separable). Sin embargo, un Perceptrón simple no permite solucionar funciones no lineales, como OR-exclusiva (o XOR), debido a que en esta función no existe ninguna recta que separe los patrones de una clase de los de la otra. Para esto es necesario

que se introduzca una capa intermedia (o capa oculta, “hidden layer”) compuesta por dos neuronas que determinen dos rectas en el plano (parte derecha de la Figura 3.27)[48]).

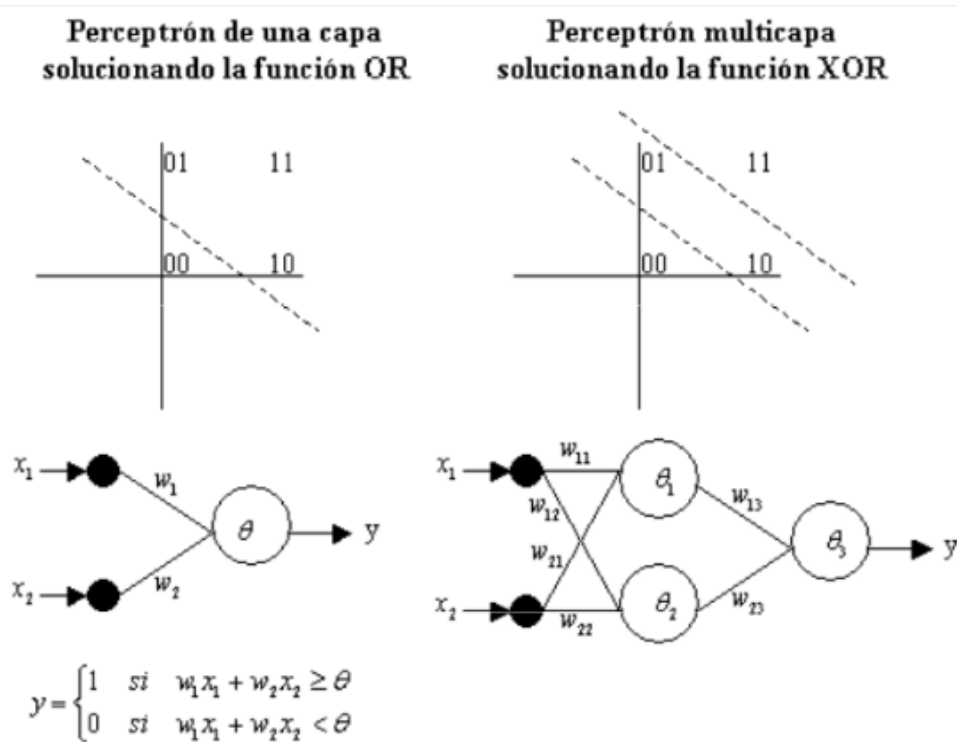


Figura 3.27: Perceptrones solucionando la función OR y XOR[48].²⁴

Esta limitación al utilizar una única neurona es una de las principales razones por la cual surgió el esquema de Perceptrón multicapa, el cual se detalla en la siguiente sección.

3.5.3 Perceptrón multicapa (MLP).

El modelo de Perceptrón multicapa (MLP) es el modelo más conocido y utilizado de Redes Neuronales. Al igual que el modelo de Perceptrón Simple, MLP es un modelo de clasificación. Generalmente el modelo MLP se utiliza para describir el funcionamiento de una red neuronal, y es el esquema que utilizaremos para nuestra implementación del modelo Word2vec. En el modelo MLP la distribución de neuronas dentro de la red se realiza formando niveles o capas, con un número determinado de dichas neuronas en cada una de ellas.

A continuación mencionaremos las tres capas existentes en el modelo MLP, cada una con distintas funciones.

²⁴Desde el punto de vista lógico la salida de la función OR es verdadera si algunos de los argumentos x_1 y x_2 son verdadero, y es falsa si todos los argumentos son falsos: entonces, la combinación $x_1 = 0$ y $x_2 = 0$ (en el gráfico de la izquierda representado por 00) da una salida falsa; y las combinaciones 01, 10, 11 verdadera. Lo que hacemos mediante el Perceptrón de una capa es separar estos resultados. En cambio, la salida de la función XOR es verdadera si las entradas no son iguales, y falso si son iguales; de esta manera 01 y 10 dan una salida verdadera, y 11 y 00 falsa. En este caso el Perceptrón multicapa puede separar estos dos resultados.

- Capa de entrada: contiene las variables de entrada; reciben directamente la información proveniente de las fuentes externas de la red.
- Capas ocultas: pueden haber 1 o N capa/s oculta/s. La longitud de N determina la profundidad de la red, dando origen al término de aprendizaje profundo (o “deep learning”). Colocando neuronas en forma secuencial permite que cada neurona reciba información procesada por una neurona anterior, lo que significa que la red puede generar conocimiento jerarquizado: de esta manera la red podría aprender conocimiento básico en las primeras capas y mientras va pasando la información a las neuronas de las siguientes capas, va generando conocimiento más abstracto y complejo.
- Capa de salida: permite retornar los resultados procesados por la red neuronal. Si estamos resolviendo un problema de clasificación esta capa puede estar formada por 2 neuronas (si se trata de una clasificación binaria) o N neuronas (en caso de clasificación multiclase).

Cabe destacar que en nuestro modelo MLP, una única neurona de nuestra capa oculta o de nuestra capa de salida puede representarse como un Perceptrón Simple (Figura 3.28). En cambio, una neurona de nuestra capa de entrada NO puede representarse como un Perceptrón simple, ya que en esta capa no existe ningún procesamiento interno. Con “procesamiento interno” nos referimos a la función de entrada y función de activación dentro de la neurona: por esta razón, las funciones de activación también residen en las neuronas de la capa oculta y/o en las neuronas de la capa de salida (las neuronas de la capa de entrada no poseen funciones de activación).

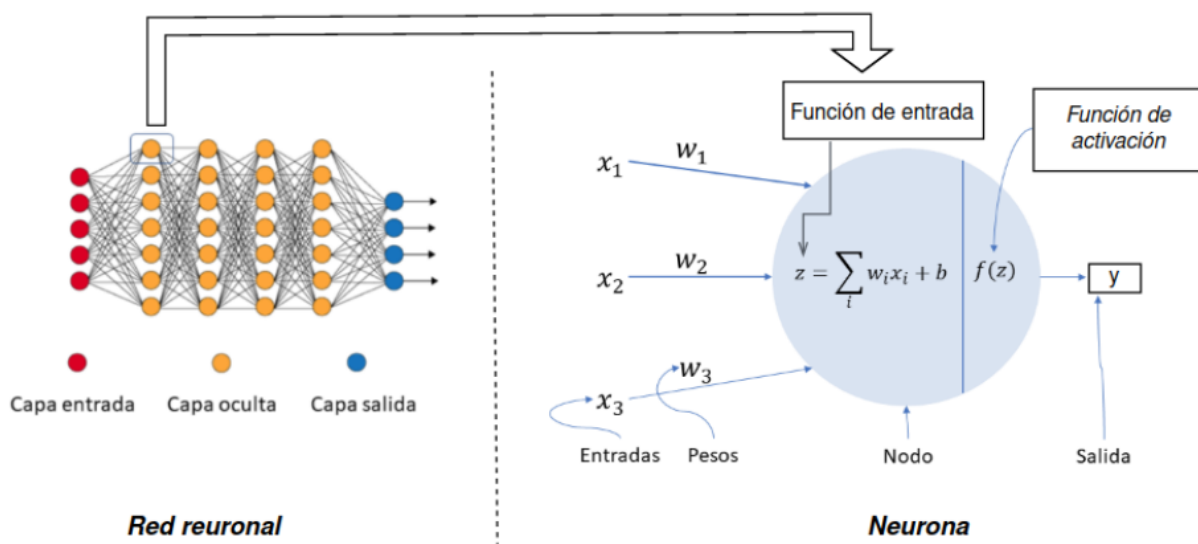


Figura 3.28: Modelo Perceptrón Multicapa -MLP- (izquierda) y Perceptrón Simple (derecha).

Observando la parte izquierda de la Figura 3.28, podemos apreciar que cada una de las capas pueden estar compuesta por una o más neuronas, lo que significa que cada capa es un vector de neuronas. La dimensión del vector determina el ancho del modelo de una red neuronal. Cada capa recibe los datos de salida de la capa anterior. Luego, cada

neurona de la capa actual realizará sus procesamientos en base a esta entrada (aplicando su función de entrada y función de activación), y su salida la pasará a cada neurona de la capa siguiente. Este tipo de redes se denominan *feedforward*, debido a que el flujo de información va de una capa a la que sigue, sin bucles de retroalimentación.

A continuación, explicaremos el modelo MLP desde un punto de vista matemático[49]. Como mencionamos previamente, con el modelo de MLP es posible combinar varios perceptrones en una capa de procesamiento, con lo cual obtenemos una función $f : R^m \rightarrow R^n$ para cada capa (oculta o de salida), donde m es el tamaño de la entrada, y n es el número de perceptrones, que a su vez determinan el tamaño de la salida. De esta manera, dada una entrada x , la función f procesa dicha entrada mediante la Formula 11.

$$f(x) = \phi(Wx + b) \quad (\text{Ver [49]}) \quad (11)$$

Donde ϕ es la función de activación que elegimos usar para esa capa, W es la matriz de pesos, y b es el vector que contiene los sesgos.

Además, como mencionamos previamente, con el modelo MLP podemos componer varias de estas capas, con lo cual obtenemos una *red*, donde los perceptrones de una capa reciben como entrada a los de la capa anterior y alimentan a los de la capa que sigue. Dada una red f y una entrada x , a la salida $f(x)$ se la denomina, naturalmente, capa de salida, mientras que a la entrada x se la suele denominar capa de entrada. Cuando tenemos una red con varias capas, todas las capas que no son de salida ni de entrada se denominan capas ocultas.

Anteriormente vimos que la red f depende de los parámetros W y b , que en su conjunto vamos a denotar con θ , y estará formada por una *composición* sucesiva de varias capas f^i , donde el superíndice indica el número de la capa. De esta manera, cada capa cuenta con sus propios parámetros θ y procesa la información que recibe de acuerdo a la Fórmula 12.

$$f^i(a^{i-1}, \theta^i) = f^i(a^{i-1}, w^i, b^i) = \phi(W^i a^{i-1} + b^i) \quad (\text{Ver [49]}) \quad (12)$$

Donde a^{i-1} es la salida, o activación, de la capa anterior, W^i es la matriz de pesos de la capa i y b^i es el vector de sesgos.

3.5.4 Funciones de activación en MLP.

Como mencionamos previamente, las funciones de activación residen en las neuronas de la capa oculta y/o en las neuronas de la capa de salida (las neuronas de la capa de entrada no poseen funciones de activación).

Algo que vale la pena mencionar es que todas las capas ocultas suelen utilizar la misma función de activación. La capa de salida normalmente utiliza una función de activación diferente de las capas ocultas y depende del tipo de predicción requerida por el modelo.

El propósito de la función de activación es introducir una *no-linealidad* dentro de la red neuronal, lo que le permite a la misma obtener una variable de salida y que varíe de forma no lineal con respecto a sus variables de entrada x . ¿Por qué esta función de activación tiene que ser una función no lineal?:

- Esto es debido a que si una red neuronal únicamente posee funciones de activación lineales (también conocidas como *función identidad*), la red se comportará como si fuera un perceptrón de una sola capa: esto es debido a que si sumamos todas las capas obtendremos otra función lineal. Independientemente de cuán compleja sea la arquitectura de la red, una función de activación lineal solo nos permite modelar una relación lineal entre las entradas y salidas, siendo efectiva solo en una capa de profundidad y para problemas de regresión lineal (por ejemplo al predecir precios de la vivienda), y no siendo útil para la mayoría de las aplicaciones del mundo real que son altamente no lineales. Resumidamente, que la función de activación sea no lineal permite a la red apilar múltiples capas de neuronas para crear lo que se conoce como una red neuronal profunda, que se requiere para aprender conjuntos de datos complejos.

En nuestra implementación de red neuronal para el modelo Word2vec (ver sección *Word embeddings (Word2vec) & WMD*), no utilizamos funciones de activación en las neuronas de la capa oculta, este modelo *únicamente utiliza la función de activación para la capa de salida*. El modelo Word2vec, en su esquema básico, utiliza una función de activación softmax para la capa de salida. En nuestra implementación se realizó una mejora a dicho esquema, introduciendo lo que se conoce como *muestreo negativo* (el cual está explicado en $VER_{SECCION_MUESTREO_NEGATIVO}$), que implementa una función de activación sigmoide para la capa de salida. En las siguientes secciones se detallarán dichas funciones de activación.

3.5.4.1 Softmax en la capa de salida.

La función Softmax comprime las salidas de cada neurona para que estas sean de 0 y 1 de tal forma que la suma de las salidas sea igual a 1. La función Softmax produce salidas que se asemejan a probabilidades. Esto es, cada salida de la red produce una probabilidad entre 0 y 1, mientras que la suma de estas probabilidades es 1. Consecuentemente, esta función es usada en la capa de salida en ANN para resolver problemas de clasificación, ya que en estos problemas se requiere separar los datos de entrada en grupos. En la Fórmula 13 podemos observar la representación matemática de la función softmax; mientras que en la Figura 3.29. podemos observar la aplicación de esta función a la capa de salida de una red neuronal.

$$Z_k = \frac{e^{y_k}}{\sum_{k=1}^M e^{y_k}} \quad (\text{Ver [48]}) \quad (13)$$

Donde $0 \leq Z_k \leq 1$, mientras que y_k es la salida de la neurona k , la cual pasa a la función softmax para obtener la salida final Z_k , y M es la cantidad total de neuronas en la capa de salida.

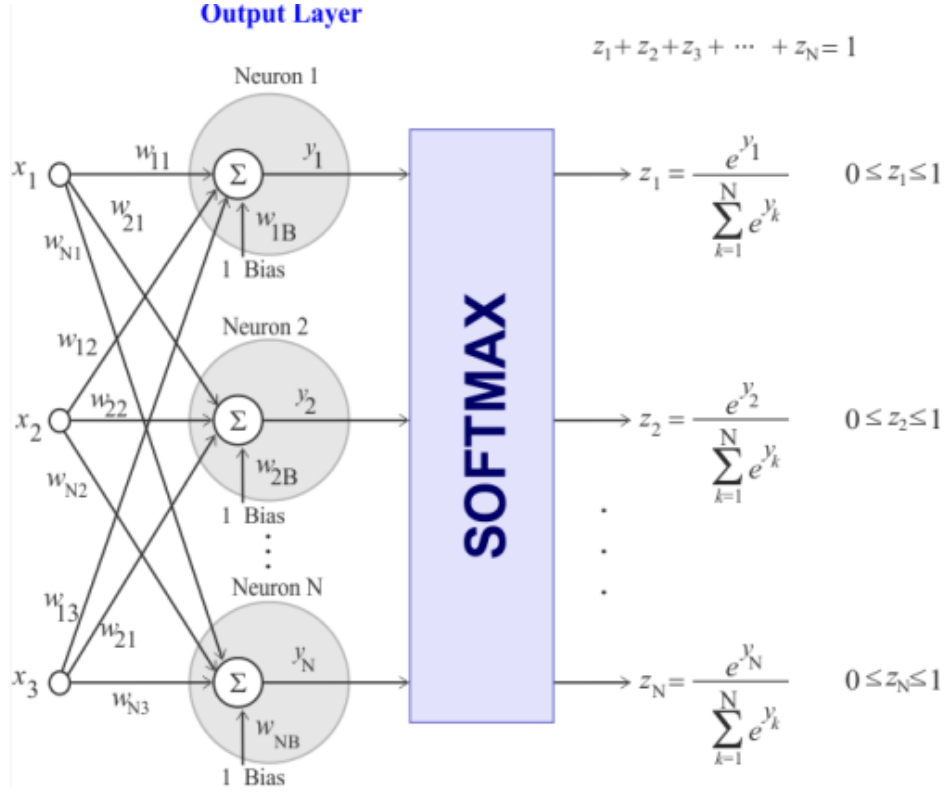


Figura 3.29: Función Softmax aplicada a la capa de salida²⁵.

3.5.4.2 Sigmoide en la capa de salida.

La función sigmoide, o también llamada curva logística, es una aproximación suave de la función escalón. Como resultado de aplicar la función sigmoide a la salida y_k de cada neuronal, al igual que en la Softmax, como salida obtenemos un valor Z_k entre 0 y 1. Pero, a diferencia de la Softmax, la suma de las salidas (de Z_1 a Z_M) no dan 1 como resultado, ya que en el denominador no se consideran a todas las salidas y_K para realizar el cálculo de la función. Utilizando la función de activación sigmoide simplemente se aplica una función $\sigma(y_k)$ por cada y_k salida de la respectiva neurona. Matemáticamente podemos observar la función sigmoide en la Fórmula 14). Mientras que en la Figura 3.30 podemos observar la representación gráfica de la función sigmoide comparándola con la función escalón.

$$\sigma(y_k) = \frac{\exp(y_k)}{1 + \exp(y_k)} = \frac{1}{1 + \exp(-y_k)} \quad (\text{Ver [49]}) \quad (14)$$

Donde $0 \leq \sigma(y_k) \leq 1$.

²⁵Obtenida del sitio web <http://sintesis.ugto.mx/WintemplaWeb>. (Consultado el 30 de abril de 2022).

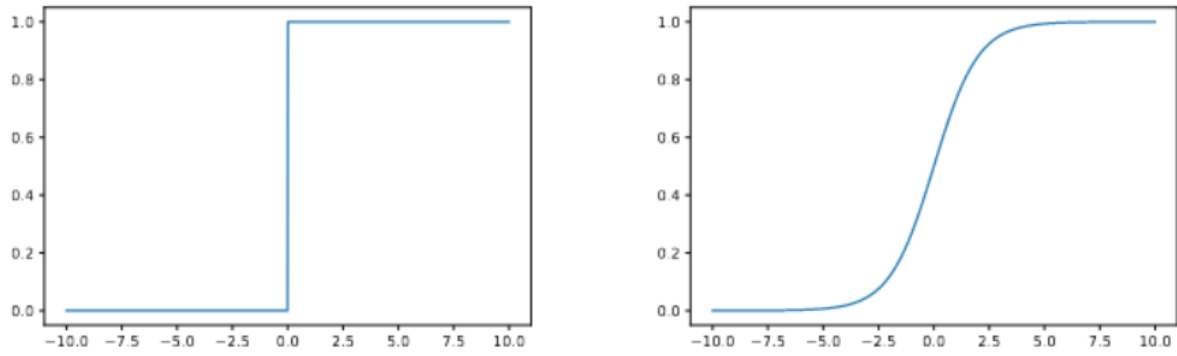


Figura 3.30: Funciones de activación escalón y sigmoide[49].

3.5.5 Entrenamiento en una red neuronal.

Como se comentó anteriormente, el modelo MLP de Redes Neuronales es un modelo de clasificación, lo que significa que realiza un aprendizaje supervisado. De esta manera, a la red neuronal durante el proceso de entrenamiento se le muestra un gran número de datos de entrenamiento para que la red ajuste / modifique sus pesos de manera iterativa y de este modo, lograr minimizar el error producido en la respuesta: lograr minimizar la *función de costo*, la cual penaliza de alguna manera los errores que comete la red al clasificar. Luego de varias iteraciones de ejemplos, se espera que la red neuronal sea efectiva para resolver la tarea para la que fue entrenada. En otras palabras, una red aprende resolviendo un problema de optimización, donde se buscan los parámetros de la red (pesos y sesgos, θ) que minimizan la función de costo o error J para cada ejemplo x_i en el conjunto de entrenamiento X .

A continuación describiremos en mayor detalle el ciclo iterativo de procesos del entrenamiento de una red neuronal[49].

1. Al comienzo del algoritmo, por única vez, se inicializan las matrices de pesos W y los vectores de bias b de toda la red con valores aleatorios. Estos son los *parámetros* de nuestro modelo de red neuronal.
2. Se aplica *forward propagation* (o propagación hacia adelante) para cada una de las muestras del set de entrenamiento y se obtienen los puntajes de salida y . Forward propagation es la manera en que una ANN computa los valores de entrada y los clasifica. Matemáticamente, durante este proceso la red neuronal recibe un vector de entrada x y devuelve un vector de salida y ²⁶ aplicando en cada una de las capas de la red la Fórmula 12 (que alimentará a la capa f^{i+1} , y que generalmente involucra sumatoria, multiplicación y uso de funciones de activación. De esta manera la señal se mueve desde la capa de entrada, pasando por las capas ocultas intermedias, hasta llegar a la capa de salida con un valor.
3. Los vectores de salida y obtenidos / predichos de todas las muestras de entrenamiento se comparan con los valores de sus vectores de salida y

²⁶Cada elemento del vector de salida y se corresponde a la salida de una k neurona de la capa de salida.

correspondientes que tiene las etiquetas reales, computándose de esta manera el *error* (o *loss*) de clasificación (entre el valor predicho y el real) mediante el cálculo de la *función de costo / pérdida* (*loss function*).

La función de pérdida $J(\theta)$ será diferente para cada modelo. Más adelante obtendremos los cálculos de las $J(\theta)$ que tendremos en consideración para nuestra implementación, las cuales son REFERENCIAR FORMULA ‘Función de costo Skip Gram con Softmax’; y REFERENCIAR FORMULA ‘Función de costo Skip Gram con Sigmoide’, siendo esta última la que finalmente utilizaremos en nuestro modelo y buscaremos minimizar. Ambas funciones de costo son la sumatoria sobre los costos individuales en los que incurre la red al clasificar cada elemento del conjunto de entrenamiento, y se calculan 1 vez por epoch²⁷.

4. Se aplica el proceso conocido como *backpropagation* (o propagación hacia atrás). No se explicará en detalle este procedimiento aquí porque no entra dentro de los objetivos de este trabajo. Lo que hay que tener en cuenta es que backpropagation utiliza la función de optimización²⁸ *descenso del gradiente* para poder ajustar / modificar los pesos y el bias de cada neurona de nuestra red *con el objetivo de minimizar la función de costo $J(\theta)$ obtenida en el paso 3*, y con ello minimizar el error de clasificación. En backpropagation calculamos el gradiente de $J(\theta)$ (o dicho de otra forma derivamos el error) en función de cada uno de los parámetros de nuestra red. Para esto se opera recursivamente capa tras capa propagando el error desde la salida hacia la entrada de la red.

De esta manera, considerando que nuestra función de pérdida es $J(\theta)$, donde nuestros parámetros θ son W (matriz de pesos) y b (vector de bias), el objetivo de nuestro modelo será encontrar los valores óptimos de W y b ($\theta^{(new)}$) para minimizar $J(\theta)$. De esta manera, en la ecuación 15 podemos observar la ecuación general de actualización de nuestros parámetros *theta* para la red neuronal.

$$\theta^{(new)} = \theta^{(old)} - n \cdot \nabla J(\theta) \quad (\text{Ver [49]}) \quad (15)$$

Donde n es la tasa de aprendizaje (o learning rate), del cual hablaremos con mayor detalle en la Sección *Hiper-parámetro n* , y $\nabla J(\theta)$ es el gradiente (también llamado pendiente o derivada) de la función $J(\theta)$ para nuestros parámetros $\theta^{(old)}$. Al tener un “-” por delante obtenemos el gradiente negativo, el cual apunta en la dirección de máximo decrecimiento de la función. Este gradiente negativo permite “guiar” al algoritmo descenso del gradiente para acercarse, de manera progresiva debido a n , al mínimo ideal de la función.

5. Volver al paso 2 hasta completar la cantidad de epochs seteados como hiper-parámetro de nuestro modelo de red neuronal.

Cabe destacar que este entrenamiento descrito tiene en cuenta el uso del *algoritmo de gradiente descendiente “básico”* y no el *algoritmo de gradiente descendiente*

²⁷Epoch es un hiper-parámetro de nuestra red neuronal, donde cada epoch representa una iteración sobre nuestros datos de entrenamiento completo. Esta iteración incluye el cálculo de los forward propagation y back propagation correspondientes.

²⁸Una función de optimización en redes neuronales tiene como objetivo encontrar los pesos W que minimicen los errores de clasificación / la función de coste.

estocástico (stochastic gradient descent, *SDG*), algo que explicaremos en mayor detalle en 4.4.4.8-Entrenamiento y función de costo con Softmax.

3.5.5.1 Hiper-parámetro n .

Como detallamos anteriormente, n es la tasa de aprendizaje de nuestra red neuronal e interviene en la actualización de nuestros parámetros θ al aplicar backpropagation utilizando el algoritmo descenso del gradiente. n define y permite regular la velocidad con la que nos desplazamos por el espacio de parámetros y por lo tanto cuán rápido nos acercamos al mínimo de la función $J(\theta)$. n definirá la cantidad de iteraciones requerida para que el algoritmo descenso del gradiente encuentre el mínimo de la función.

n es un parámetro que a diferencia de θ no es ajustado por el algoritmo de optimización del descenso del gradiente, sino que debe ser configurado por separado. Como mencionamos anteriormente, a este tipo de parámetros se los denomina hiper-parámetros, y requieren de cierto cuidado en su elección para garantizar el buen rendimiento de la red neuronal.

Mientras más rápido nos acerquemos al mínimo de la función, más rápido la red estará aprendiendo a realizar su tarea. Sin embargo, como ya mencionamos antes, valores muy altos del hiper-parámetro n pueden tener un efecto contrario, llevando a que eventualmente el algoritmo pase de largo de manera reiterada esta mínimo, impidiendo el aprendizaje. A veces aprender requiere ir más lento y ser más cuidadosos. Lo mismo ocurre con la red, y en ese caso es conveniente ajustar n para conseguir que la red no pase por alto los detalles. En la Figura 3.31 podemos observar el efecto de diferentes tasas de aprendizaje durante el entrenamiento.

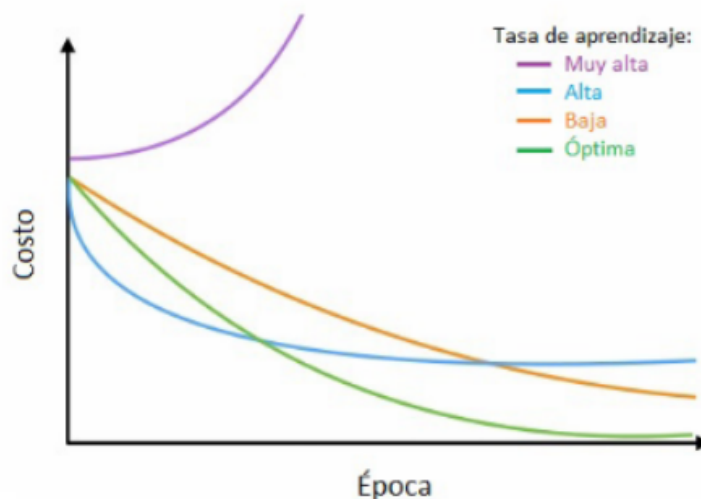


Figura 3.31: Efecto de diferentes tasas de aprendizaje n durante el entrenamiento.[49].

Una tasa de aprendizaje demasiado alta puede ocasionar que el algoritmo de entrenamiento no pueda disminuir el costo de entrenamiento más allá de cierto punto, o en el peor de los casos puede ocasionar que el mismo aumente. Una tasa de aprendizaje demasiado baja puede hacer que el costo disminuya muy lentamente, ralentizando el

aprendizaje. La tasa de aprendizaje η ideal es un compromiso entre lograr una disminución significativa del costo y mantener una buena velocidad de aprendizaje[49].

4 Natural Language Processing.

FALTA

4.1 Introducción.

FALTA

4.2 Preprocesamiento de textos.

FALTA

4.3 Similitud entre textos.

FALTA

4.4 Técnicas para medir Similitud entre textos.

FALTA

4.4.1 Cosine Similarity.

FALTA

4.4.2 Word Mover's Distance (WMD).

FALTA

4.5 Técnicas de vectorización.

FALTA

Previamente a utilizar Cosine Similarity y WMD para (—completar—) se debe emplear alguna técnica de vectorización que permita representar las palabras de nuestros textos a un espacio vectorial. De esta forma Cosine Similarity y WMD podrán interpretarlos de la mejor manera. Como técnicas de vectorización se utilizarán TF-IDF y Word Embeddings.

4.5.1 TF-IDF.

FALTA

El algoritmo TF-IDF asigna valores numéricos a las palabras en función de la frecuencia con que aparecen en los textos para medir la frecuencia de ocurrencia de un término en la colección de documentos, expresando cuán relevante es una palabra para un documento en una colección.

4.5.2 Word Embeddings.

FALTA

Los Word Embeddings son necesarios para utilizar WMD. Los Word Embeddings son una de las variantes más populares para representar textos. Son vectores previamente entrenados y generados mediante un modelo de red neuronal secuencial; de esta manera son capaces de capturar los contextos de una palabra en el documento llegando a poder contener información semántica y sintáctica.

4.5.2.1 ¿Cómo entrenar Word Embeddings?

FALTA

5 Implementación.

La implementación de este Sistema se trabajó en dos grandes partes:

1. Obtención del modelo de clasificación.

En esta primera parte se obtuvieron y preprocesaron datasets de Curriculum Vitae de distintos candidatos y descripciones de puestos de trabajo de IT publicados por distintas empresas, para luego ser comparados y obtener similitudes entre los textos utilizando las técnicas para medir distancias y obtener dichas similitudes (WMD y Cosine Similarity) y las técnicas de vectorización (TF-IDF y Word Embeddings).

Una vez obtenidas estas mediciones de similitud entre los Curriculum Vitae de los candidatos y las descripciones de los puestos laborales de IT, estos valores se utilizaron para alimentar un algoritmo de clustering K-means que a su vez, con sus datos de salida (4 clusters), alimentan a un modelo de clasificación KNN. Finalmente, con este modelo KNN logramos, en base a los valores de similitud de nuevos candidatos, clasificar qué tan similares son dichos candidatos con respecto a la descripción de un puesto de IT: similitud escasa, similitud media, similitud alta, similitud muy alta.

Estos análisis se realizaron en documentos de Jupyter Notebook utilizando Python; y sirvieron para evaluar el comportamiento del modelo de clasificación y los distintos algoritmos de medición de similitudes para luego ser utilizados en la siguiente etapa.

2. Integración al Sistema Web.

Etapas posteriores a la primera parte. Una vez observado que los resultados fueron los esperables, lo que se hizo fue reutilizar las funciones que contenían la lógica de los distintos algoritmos utilizados junto con el modelo de clasificación KNN obtenidos previamente en la parte 1, para integrar todo esto en el sistema Web. Este sistema web está realizado en Django ²⁹, y cuenta con una base de datos relacional que contiene la información de los candidatos y reclutadores junto con los Curriculum Vitae y puestos que hayan cargado.

De esta manera, nuestro sistema cuenta con una interfaz gráfica permitiendo interactuar entre candidatos y reclutadores y, principalmente, permitiendo que el reclutador sea capaz de obtener un listado con los N candidatos más similares a un puesto determinado, y ordenados de mayor a menor de acuerdo a esta *similitud*. Dicha *similitud* representa el resultado obtenido de la clasificación por nuestro modelo KNN.

²⁹Framework de desarrollo web de código abierto, escrito en Python, que respeta el patrón de diseño conocido como modelo-vista-controlador.

5.1 Obtención del modelo de clasificación.

5.1.1 Introducción.

Como inicio definamos qué es un modelo. En nuestro caso, un modelo representa Este modelo se construyó en base a ... La implementación de este Sistema se realizó en Python...

5.1.2 Esquema.

Como podemos ver la figura 5.1 representa el procedimiento utilizado para la obtención de nuestro modelo de clasificación.

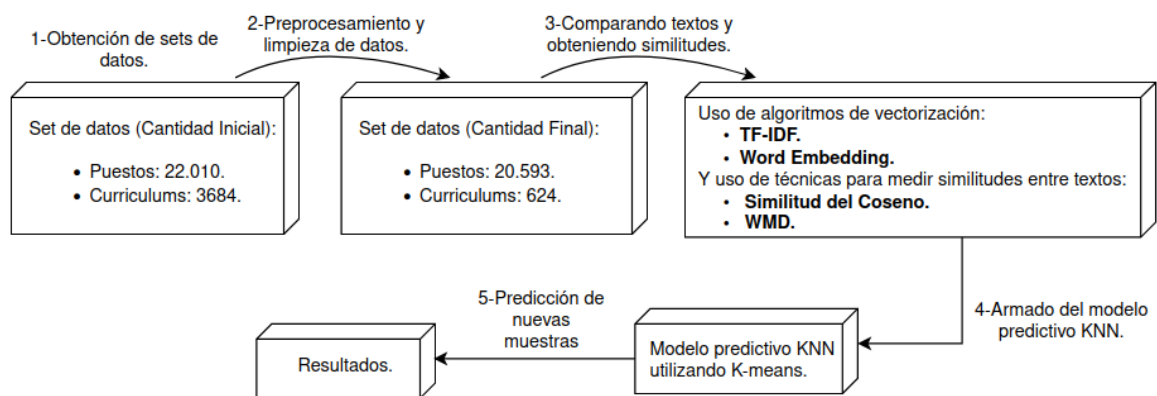


Figura 5.1: Pipeline Flow para la obtención del modelo de clasificación KNN

5.1.3 Obtención de sets de datos.

En primer lugar debemos definir qué es un set o conjunto de datos. Un set o conjunto de datos es una tabla de una base de datos o, matemáticamente, una matriz estadística de datos. Cada columna de la tabla representa una variable del set de datos; y cada fila representa a un miembro determinado del mismo.

Para este Proyecto utilizamos dos grandes sets de datos que se obtuvieron mediante la recolección de distintos archivos alojados en la Web, los cuales estan descriptos a continuación.

5.1.3.1 Curriculum Vitae.

Los set de datos de Curriculum Vitae de los candidatos se obtuvieron de las siguientes fuentes:

1. 228 Curriculums en formato docx y posteriormente convertidos a pdf, obtenidos del sitio Kaggle³⁰. Estos pdfs son candidatos de la India con experiencia en el rubro de IT.
2. 2484 Curriculums en formato CSV, obtenidos del sitio Kaggle³¹. Este CSV cuenta con curriculums vitae obtenidos del sitio web de postulación de trabajos 'livecareer.com'.
3. 962 Curriculumns en formato CSV, obtenidos del sitio Kaggle³². Este CSV cuenta con curriculums vitae repartidos en distintas categorías de IT.
4. 10 Curriculums en formato PDF, los cuales los cuales fueron obtenidos como ejemplos mediante una recolección propia de distintos sitios web.

5.1.3.2 Descripciones Puestos Laborales.

Los set de datos de descripciones de puestos laborales se obtuvieron de las siguientes fuentes:

1. 22.000 descripciones en formato CSV; obtenido del sitio Kaggle³³. El CSV cuenta con descripciones de puestos obtenidos del sitio web de USA de postulación de trabajos del rubro de IT 'Dice.com'.
2. 10 descripciones en formato CSV; obtenidas como ejemplos mediante una recolección propia del sitio Indeed³⁴ para puestos de trabajo de IT.

³⁰<https://www.kaggle.com/palaksood97/resume-dataset>

³¹<https://www.kaggle.com/snehaanbhawal/resume-dataset>

³²<https://www.kaggle.com/gauravduttakiit/resume-dataset>

³³<https://www.kaggle.com/PromptCloudHQ/us-technology-jobs-on-dicecom>

³⁴<https://www.indeed.com/q-USA-jobs.html>

5.1.4 Preprocesamiento de textos.

Previamente a utilizar las técnicas para medir distancias y obtener similitudes entre textos (WMD y Cosine Similarity) y los algoritmos de aprendizaje (KNN y K-Means) necesitamos que los datos que comparemos e introduzcamos en los algoritmos estén lo más limpios posible; ya que de lo contrario las mismos podrían clasificar o predecir de forma errónea. Este análisis previo sobre los datos debe ser minucioso ya que puede haber valores incoherentes o absurdos.

El procedimiento para la Limpieza de los Curriculum Vitae y las descripciones de los puestos laborales fue el siguiente:

1. Convertimos todo a minúscula.
2. Eliminamos datos no relevantes para nuestros análisis (mails y páginas web).
3. Eliminamos signos de puntuación y caracteres especiales (incluyendo números).
4. Eliminamos stop words.
5. Eliminamos common words no relevantes para nuestros análisis.
6. Aplicamos Lematización y Tokenización.
7. Eliminamos repetidos.
8. Obtenemos y usamos bi-gramas.

Luego de aplicar preprocesamiento y limpieza de datos nos quedarán los siguientes tamaños de nuestros datasets:

- 624 curriculums vitae de candidatos (en formato pdf y csv).
- 20593 descripciones de puestos de IT (en formato csv).

5.1.5 Cantidad final del set de datos y su uso en las distintas etapas.

El total de 624 curriculums vitae de candidatos y 20593 descripciones de puestos de IT que mencionamos previamente, serán utilizados para el entrenamiento y obtención de vectores mediante TF-IDF (para el posterior cálculo de Cosine Similarity) y para el entrenamiento de Word2Vec y obtención de los Word Embeddings (para el posterior cálculo de WMD).

Por otro lado, para calcular Cosine Similarity y WMD, para utilizarlos en K-means y para entrenar a nuestro algoritmo KNN, utilizaremos únicamente una porción de nuestros datasets:

1-Para el cálculo de Cosine Similarity y WMD:

- 301 curriculums vitae de candidatos.
- 201 descripciones de puestos de IT.

Nota: No obstante, al realizar los cálculos de distancias compararemos cada curriculum vitae con cada Job Description, obteniendo un dataframe total de 3131 filas con sus respectivos valores de WMD y Cosine Sim.

2-Para el uso de K-means y entrenamiento con KNN (eliminamos un curriculum vitae y una descripción de puesto IT que los utilizamos en '3-'):

- 300 curriculums vitae de candidatos.
- 200 descripciones de puestos de IT.

Nota: como se comentó previamente, nos quedarán 3000 filas / puntos para usar en K-means y entrenar KNN; llegando a representar estos 3000 puntos en un plano de 2 dimensiones.

3-Para la clasificación de nuevas muestras mediante KNN:

- 1 curriculum vitae de candidatos.
- 1 descripción de puesto de IT.

Nota: como se comentó previamente, nos quedarán 131 filas para clasificar.

¿Por qué utilizamos solo una porción de nuestros datasets?: Esto es debido a los drawbacks de WMD y KNN.

- WMD: posee una alta complejidad en el cálculo de la distancia, teniendo un tiempo de ejecución muy elevado. Como ejemplo, al correrlo localmente, el cálculo de WMD para 3131 filas tardó 7 horas; frente a los 3 segundos que tardó el cálculo de Cosine Similarity para la misma cantidad de filas.
- KNN: KNN es una gran opción para datasets pequeños con pocas variables de entrada; pero tiene problemas cuando la cantidad de entradas es muy grande. En

grandes dimensiones, los puntos que pueden ser similares pueden tener distancias muy grandes. Además, cada vez que se va a hacer una predicción con KNN, busca al vecino más cercano en el conjunto de entrenamiento completo. Por esto, se debe utilizar un dataset pequeño para que el clasificador KNN complete su ejecución rápidamente.

En conclusión, al utilizar solo una porción de nuestros datasets para obtener los distintos cálculos de distancias y entrenar KNN, el cálculo de WMD se podrá realizar en un tiempo finito, y nuestro clasificador KNN funcionará rápida y eficientemente al realizar predicciones.

5.2 Comparando textos y obteniendo similitudes.

FALTA

Previamente a utilizar Cosine Similarity y WMD para obtener las medidas de similitud entre los textos, se debe emplear alguna técnica de vectorización que permita representar las palabras de nuestros textos a un espacio vectorial. De esta forma Cosine Similarity y WMD podrán interpretarlos de la mejor manera. Como mencionamos previamente, como técnicas de vectorización se utilizarán TF-IDF y Word Embeddings.

5.2.1 TF-IDF & Cosine Similarity.

adgadaha

5.2.2 Word embeddings (Word2vec) & WMD.

dagagad

5.2.2.1 Elección de hiper-parámetros Word2vec.

adgadg

5.2.2.2 Word Mover's Distance (WMD.

dagadga

5.3 Armado del modelo de clasificación KNN.

FALTA

Una vez obtenidas estas mediciones de similitud entre los Curriculum Vitae de los candidatos y las descripciones de los puestos laborales de IT, estos valores se utilizarán para alimentar un algoritmo de clustering K-means que a su vez, con sus datos de salida (4 clusters), alimentarán a un modelo de clasificación KNN. Finalmente, con este modelo KNN lograremos, en base a los valores de similitud de nuevos candidatos, clasificar qué tan similares son dichos candidatos con respecto a la descripción de un puesto de IT: similitud escasa, similitud media, similitud alta, similitud muy alta.

5.4 Clasificación de nuevas muestras y resultados obtenidos.

FALTA

5.5 Integración al Sistema Web.

FALTA

Anteriormente lo que se hizo fue un análisis mediante documentos en Jupyter Notebooks para evaluar el comportamiento del modelo de clasificación y los distintos algoritmos de medición de similitudes.

Al observar que los resultados fueron los esperables, lo que se hizo en esta última etapa fue reutilizar las funciones que contenían la lógica de los distintos algoritmos utilizados junto con el modelo de clasificación KNN obtenidos en la fase previa, para integrar todo esto en el sistema Web.

Como mencionamos previamente, este sistema web está realizado en Django, y cuenta con una base de datos relacional que contiene la información de los candidatos y reclutadores junto con los Curriculum Vitae y puestos que hayan cargado.

De esta manera, nuestro sistema cuenta con una interfaz gráfica permitiendo interactuar entre candidatos y reclutadores y, principalmente, permitiendo que el reclutador sea capaz de obtener un listado de los N candidatos más similares a un puesto determinado, y ordenados de mayor a menor de acuerdo a esta *similitud*. Dicha *similitud* representa el resultado obtenido de la clasificación por nuestro modelo KNN.

El sistema web contará con 2 tipos de usuario:

- Candidato: quienes cargarán en el sistema sus Curriculum Vitae y aplicarán a los distintos puestos disponibles.
- Reclutador: quienes cargarán en el sistema los puestos de trabajo que tengan disponibles y podrán consultar, entre otras cosas, un listado con los N candidatos más similares a un puesto determinado, y ordenados de mayor a menor de acuerdo a esta *similitud*.

5.5.1 Base de datos.

Nuestros datos los almacenaremos en una base de datos **FALTA definir cual**.

Para modelar y gestionar nuestros datos utilizamos el modelo relacional ³⁵.

Sacar la mayoría del documento modelo-entidad-relacion-case-method-richar-barker.pdf

Para comprender los datos que se almacenan en dicha base de datos, los representaremos utilizando un diagrama entidad relación ³⁶. Previamente a esto explicaremos los elementos del diagrama de entidad relación:

FALTA PONER IMAGEN CON LOS ELEMENTOS: Entidad rectángulo, Unión entre entidades s
las líneas que puede ser obligatoria u opcional, cardinalidad son los 1:M / 1:1 / M:M

- Entidad: objeto concreto o abstracto que figura en nuestra base de datos. Por ejemplo una entidad puede ser un alumno, un cliente, una empresa, etc. Dentro de las entidades estan los atributos, atributos principales o clave primaria (PK) y atributos foraneos o clave secundaria (FK). Las entidades que necesitamos para crear nuestra BD son: Candidato, Puesto, Reclutador y Candidato_Puesto -entidad intermedia entre Candidato y Puesto-.
- Unión entre entidades: pueden ser obligatorias u opcionales. En nuestro diagrama nuestras uniones son todas opcionales, ya que el reclutador puede o no CARGAR un puesto, el candidato puede o no APLICAR a un puesto y a su vez el puesto puede o no ser aplicado por un candidato.
- Cardinalidad: Relación entre entidades o mapeo. La cardinalidad es el tipo de relación entre entidades. Observando la figura 5.2 y considerando que los rectángulos azules son una entidad y los naranjas son otra entidad observamos que pueden haber 4 tipos de cardinalidades posibles:
 1. Uno a uno: a cada entidad azul le corresponde solo una entidad naranja.
 2. Uno a muchos: a cada entidad azul le corresponde una o varias entidades naranjas.
 3. Muchos a uno: a cada entidad naranja le corresponde una o varias entidades azules.
 4. Muchos a muchos: las entidades azules pueden tener varias entidades naranjas y las entidades naranjas también pueden tener varias entidades azules.

³⁵Una base de datos relacional es un conjunto de una o más tablas estructuradas en registros (líneas) y campos (columnas), que se vinculan entre sí por un campo en común.

³⁶Un modelo entidad-relación es una herramienta para el modelo de datos, la cual facilita la representación de entidades de una base de datos.



Figura 5.2: Tipos de cardinalidad.

La cardinalidad entre nuestras entidades son:

- Entre Candidato y Puesto existe una relación muchos a muchos (M:M), ya que un candidato puede aplicar a M puestos y un puesto puede ser aplicado por M candidatos. Es por esto que se creó la tabla intermedia Candidato_Puesto conllevando dos relaciones uno a muchos (1:M) con Puesto y Candidato.
- Entre Reclutador y Puesto existe una relación de uno a muchos (1:M), ya que un reclutador puede cargar M puestos, y un puesto pertenece a un solo reclutador.

En cuanto a las claves pueden haber dos tipos. Por un lado está la clave primaria o atributo principal (PK) es única y toda entidad debe tener la suya. Pueden haber múltiples PKs; estas se llaman PKs compuestas. Y por el otro está la clave secundaria o atributo foráneo (FK). Estas claves identifican a una entidad externa en otra, utilizándose para generar relaciones entre nuestras entidades. Si tenemos una clave FK en una entidad, significa que dicha clave FK es clave PK en otra entidad.

El diagrama de relación que utilizamos para nuestro trabajo lo observamos en la figura 5.3.

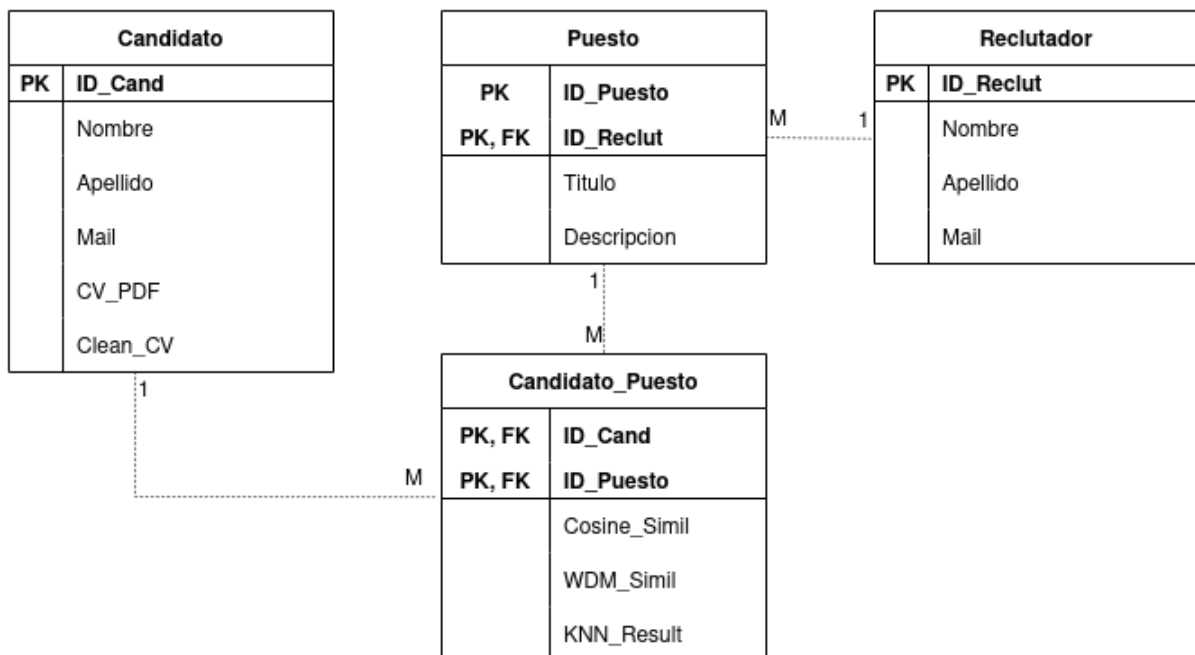
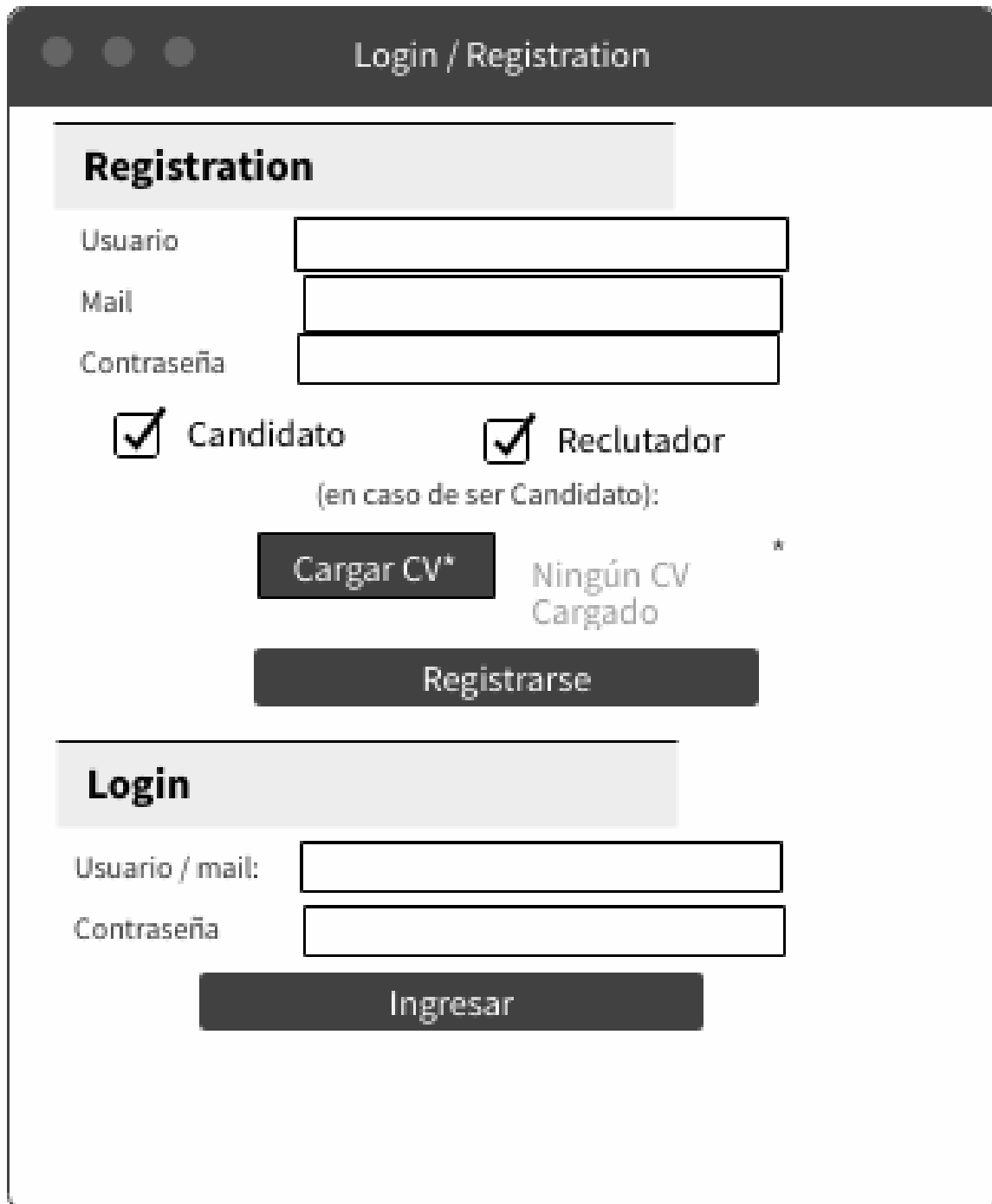


Figura 5.3: Diagrama de relación utilizado.

5.5.2 Secciones del sistema

Para registrarse o loguearse al sistema, se implementará la interfaz provista en la figura 5.4.



The image shows a sketch of a web interface for 'Login / Registration'. It is divided into two main sections: 'Registration' and 'Login'. The 'Registration' section includes input fields for 'Usuario', 'Mail', and 'Contraseña'. Below these are two checkboxes, 'Candidato' and 'Reclutador', both of which are checked. A note '(en caso de ser Candidato):' is present. There is a button 'Cargar CV*' and a status indicator 'Ningún CV Cargado *'. A large 'Registrarse' button is at the bottom of this section. The 'Login' section has input fields for 'Usuario / mail:' and 'Contraseña', followed by an 'Ingresar' button. The entire interface is enclosed in a window-like frame with a title bar.

Login / Registration

Registration

Usuario

Mail

Contraseña

☒ Candidato ☒ Reclutador

(en caso de ser Candidato):

Ningún CV Cargado *

Login

Usuario / mail:

Contraseña

Figura 5.4: Logueo y Registración.

ES UN BOCETO, FALTA PONER LA IMAGEN REAL

El Candidato tendrá acceso al menú indicado en la figura 5.5.

Candidato

Mi Perfil

Nombre y apellido

DNI

Fecha nacimiento

Teléfono

Email

Domicilio

Actualizar datos

Cargar y analizar nuevo CV

Puestos disponibles

ID Puesto	Puesto	Descripción Pu...	Ubicación
1	Programador F...	Descripción larga	Buenos Aires
2	DB Engineer	Descripción larga	Buenos Aires
3	Data Scientist	Descripción larga	Buenos Aires

(se podrá filtrar/ordenar en cada columna)

Postularse

Postulaciones

ID Puesto	Puesto	Descripción...	Ubicación	Fecha Pos...
1	Programa...	Descripción...	Buenos Ai...	15/06/2021

(se podrá filtrar/ordenar en cada columna)

Figura 5.5: Vista del Candidato.

ES UN BOCETO, FALTA PONER LA IMAGEN REAL

Por su parte, el Reclutador tendrá acceso al menú indicado en la figura 5.6.

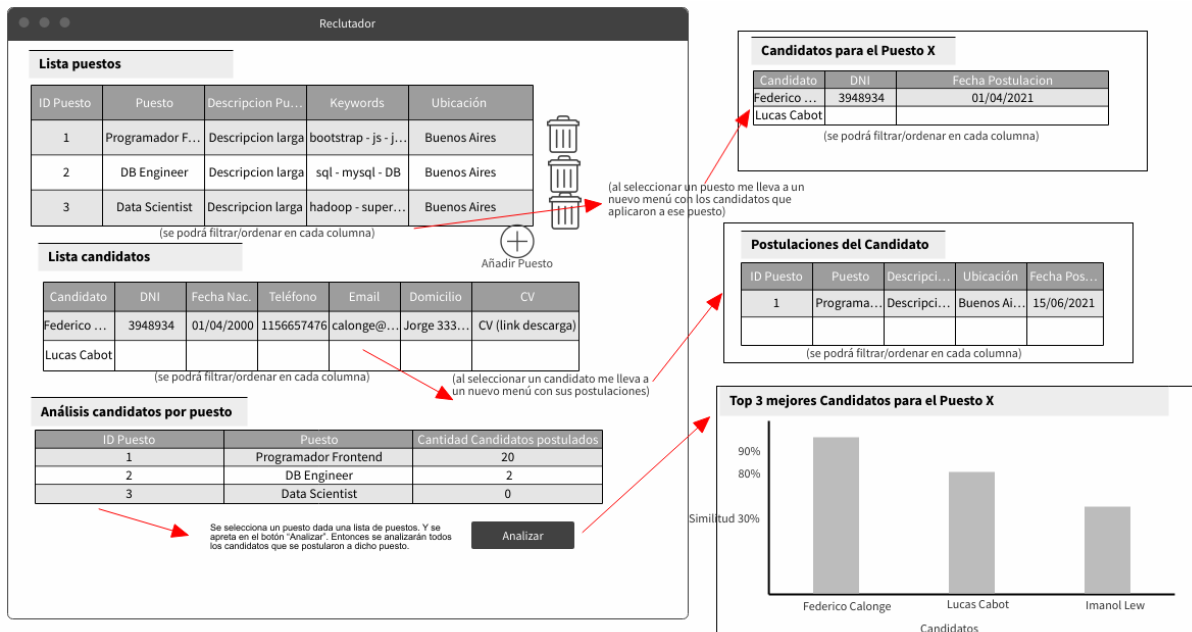


Figura 5.6: Vista del Reclutador.

ES UN BOCETO, FALTA PONER LA IMAGEN REAL

5.5.3 Manejo de los datos.

FALTA

5.5.3.1 Modelado.

FALTA

5.5.3.2 Filtrado.

FALTA

5.5.3.3 Visualización.

FALTA

5.6 Pipeline Flow final del Sistema.

FALTA

Una vez que el reclutador dentro de la sección observada en la figura 5.6 haga click en ".Analizar", el sistema reflejará el pipeline indicado en la figura 5.7 para obtener como resultado un listado con los N candidatos más similares a un puesto determinado, y ordenados de mayor a menor de acuerdo a esta *similitud*. Dicha *similitud* representa el resultado obtenido de la clasificación por nuestro modelo KNN.

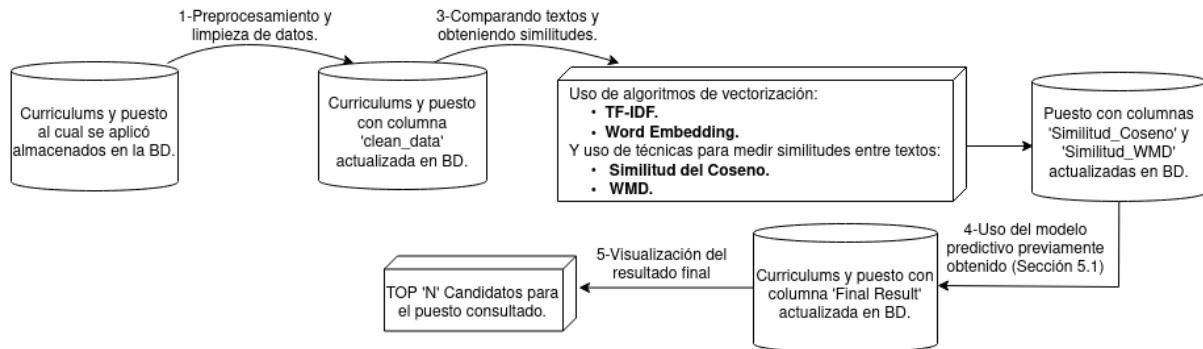


Figura 5.7: Pipeline Flow final del Sistema.

5.7 Conclusiones.

FALTA

5.8 Caso de Uso.

FALTA

5.9 Limitaciones del sistema.

Las limitaciones del sistema son las siguientes:

- Solo acepta curriculums en formato PDF.
- Curriculums y Puestos de trabajo en idioma inglés.
- Curriculums y Puestos de trabajo de IT.

6 Próximos pasos.

FALTA - acá poner mejoras

7 Anexos.

7.1 Funciones de activación.

En la figura 7.1 podemos observar una lista de las funciones de activación más comunes utilizadas en Redes Neuronales. Mientras que en la figura 7.2 se detallan las expresiones matemáticas y los posibles rangos de valores de salida para dichas funciones de activación.

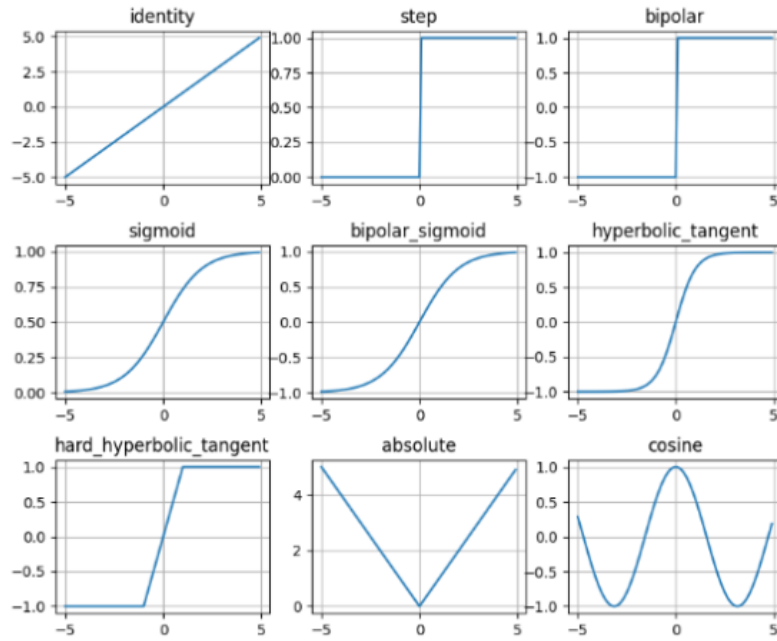


Figura 7.1: Funciones de activación más comunes[46].

Name	Expression	Range
Identity	$id(a) = a$	$(-\infty, +\infty)$
Step (Heavyside)	$Th_{\geq 0}(a) = \begin{cases} 0 & \text{if } a < 0 \\ 1 & \text{otherwise} \end{cases}$	$\{0, 1\}$
Bipolar	$B(a) = \begin{cases} -1 & \text{if } a < 0 \\ +1 & \text{otherwise} \end{cases}$	$\{-1, 1\}$
Sigmoid	$\sigma(a) = \frac{1}{1+e^{-a}}$	$(0, 1)$
Bipolar sigmoid	$\sigma_B(a) = \frac{1-e^{-a}}{1+e^{-a}}$	$(-1, 1)$
Hyperbolic tangent	$\tanh(a)$	$(-1, 1)$
Hard hyperbolic tangent	$\tanh_H(a) = \max(-1, \min(1, a))$	$[-1, 1]$
Absolute value	$abs(a) = a $	$[0, +\infty)$
Cosine	$\cos(a)$	$[-1, 1]$

Figura 7.2: Fórmulas y rangos de las funciones de activación más comunes³⁷[46].

³⁷ a indica el resultado luego de haber aplicado la función de entrada dentro de la neurona artificial. Como mencionamos anteriormente, la función de entrada más común es la sumatoria de las entradas ponderadas.

7.2 Ejemplo de obtención de Word Embeddings mediante skipgram y softmax.

Tomaremos un ejemplo del libro Oreally Online y explicaremos resumidamente los pasos que debemos realizar para obtener nuestros Word Embeddings utilizando el modelo Skip-gram con función de salida softmax y utilizando el algoritmo de gradiente descendiente básico.

El *corpus de texto* con el que nuestro algoritmo entrenará será la oración “The quick brown fox jumps over the lazy dog”. Previamente a que ingrese al modelo, este corpus debe ser tokenizado, obteniendo el siguiente *corpus tokenizado*: [“the”, “quick”, “brown”, “fox”, “jumps”, “over”, “the”, “lazy”, “dog”]. Nuestro *vocabulario*³⁸ será el siguiente: [“the”, “quick”, “brown”, “fox”, “jumps”, “over”, “lazy”, “dog”]. Observamos que el tamaño del *corpus tokenizado* es 9, mientras que el tamaño del *vocabulario* es 8: uno menos debido a la repetición de la palabra “the”.

Cabe mencionar que en este ejemplo no se preprocesaron los textos previamente a armar nuestro corpus tokenizado. Si se preprocesaran los mismos -como se realiza en nuestra implementación-, nuestro corpus tokenizado que será utilizado como entradas del modelo quedaría como [“quick”, “brown”, “fox”, “jump”, “lazy”, “dog”], y nuestro vocabulario, como no existe repetición de palabras, estarían formados por estas mismas 6 palabras / tokens.

Para el entrenamiento de nuestro modelo consideraremos los siguientes hiper-parámetros:

- Tamaño de ventana de contexto de 2. ($C = 2$).
- Tamaño de embedding de 3. ($N = 3$).

Lo primero que tenemos que hacer es armar nuestro dataset, el cual será utilizado como entrenamiento. Para mayor entendimiento, en la Figura 7.3 representaremos las palabras utilizadas en cada iteración para entrenar a nuestro modelo Skipgram. En este caso, como nuestro tamaño de ventana de contexto es 2, tomaremos para cada paso 2 palabras de cada lado de la palabra central como nuestras palabras de contexto: en el primer paso nuestra palabra central es “the” y nuestras palabras de contexto son “quick” y “brown”. En este caso no podemos tomar palabras a la izquierda, por esto únicamente tenemos 2 palabras de contexto; distinto es el caso del paso 3, donde tomamos como palabra central “brown” y sí podemos tomar las 2 palabras de contexto de ambos lados: “the”, “quick”, “fox”, “jumps”.

³⁷El corpus tokenizado son todas las palabras de nuestro corpus que serán utilizadas para entrenar y evaluar nuestro modelo.

³⁸El vocabulario es el set de palabras únicas -sin repetir- obtenidas de nuestro corpus tokenizado

Palabras utilizadas para entrenar Skipgram									N° iteraciones necesarias
the	quick	brown	fox	jumps	over	the	lazy	dog	2
the	quick	brown	fox	jumps	over	the	lazy	dog	3
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	4
the	quick	brown	fox	jumps	over	the	lazy	dog	3
the	quick	brown	fox	jumps	over	the	lazy	dog	2

Figura 7.3: Armando dataset para Skipgram

En cada iteración el modelo entrenará con un *par* de palabras (central - de contexto). En la Figura 7.4 podemos observar el dataset que obtuvimos armándolo con las palabras de entrada (centrales) y palabras de salida (de contexto; o palabras “target”) que se utilizarán para entrenar a nuestro modelo Skipgram, junto a las iteraciones necesarias. Por ejemplo, para nuestro primer paso de la Figura 7.3 (primer fila), observamos que nuestra palabra central es “the” y nuestras palabras de contexto son “quick” y “brown”. Vemos que se forman 2 pares de palabras, por esto en la primer iteración entrenaremos con el par (“the”, “quick”) y en la segunda iteración usaremos el par (“the”, “brown”). Aquí lo que estamos haciendo esencialmente es capturar los pares de palabras de todo nuestro corpus para capturar el *contexto* y usarlo como información para entrenar nuestros word embeddings.

Entrada (palabra central)	Salida (palabras de contexto)	Iteración
the	quick	1
the	brown	2
quick	the	3
quick	brown	4
quick	fox	5
brown	the	6
brown	quick	7
brown	fox	8
brown	jumps	9
fox	quick	10
fox	brown	11
fox	jumps	12
fox	over	13
(...)	(...)	(...)
dog	the	29
dog	lazy	30

Figura 7.4: Dataset para Skipgram

Como mencionamos previamente, las palabras que ingresarán como entrada en nuestro modelo, presentarán un vector en formato one-hot encoding: Ver Figura 7.5. Recordemos que mediante one hot encoding se asigna un único código para cada única palabra.

Este vector tiene las características descritas a continuación.

- Su valor es 1 para la posición que tiene la palabra en nuestro vocabulario y 0 para las demás posiciones.
- Su tamaño es igual al número de palabras de nuestro vocabulario (en nuestro caso $V = 8$).

Entrada (palabra central)	One-hot encoding	Índice en el vocabulario
the	[1,0,0,0,0,0,0,0]	1
quick	[0,1,0,0,0,0,0,0]	2
brown	[0,0,1,0,0,0,0,0]	3
fox	[0,0,0,1,0,0,0,0]	4
jumps	[0,0,0,0,1,0,0,0]	5
over	[0,0,0,0,0,1,0,0]	6
the	[1,0,0,0,0,0,0,0]	1
lazy	[0,0,0,0,0,0,1,0]	7
dog	[0,0,0,0,0,0,0,1]	8

Figura 7.5: Representación en one-hot encoding para cada una de las palabras de entrada

Teniendo en cuenta lo mencionado previamente, pasaremos a explicar gráficamente las iteraciones necesarias para que nuestro modelo Skipgram entrene, el cual podemos visualizar en la Figura 7.6.

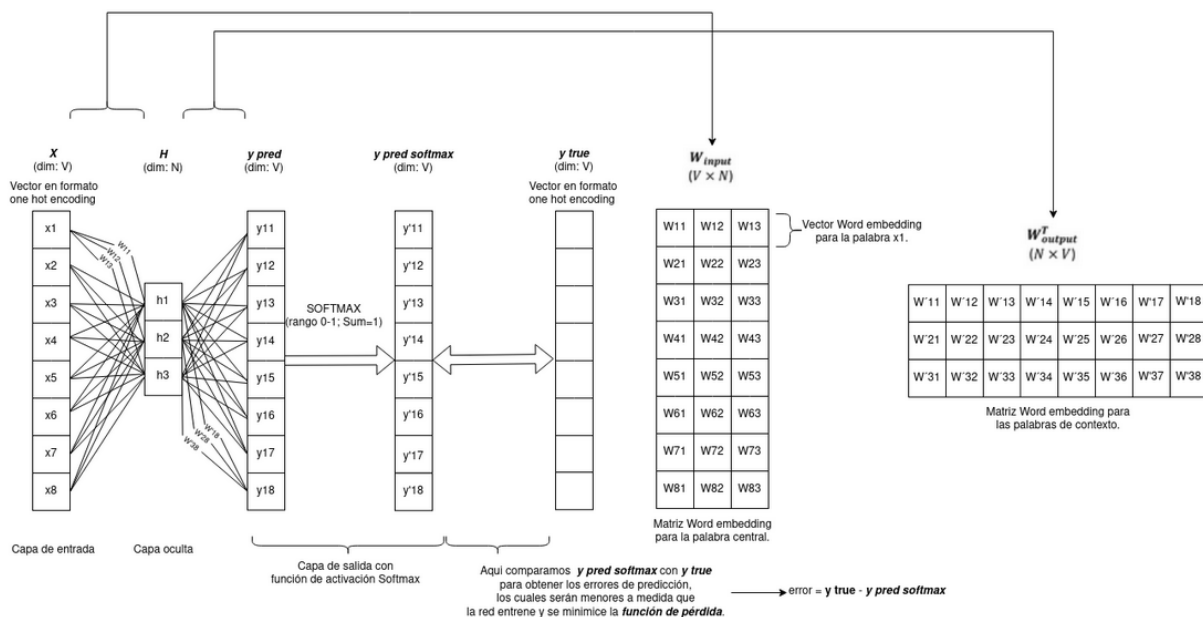


Figura 7.6: Arquitectura Skipgram con $V=8$, $N=3$.

A cotinuación detallaremos las iteraciones de entrenamiento necesarias teniendo en cuenta la Figura 7.4.

1. El primer paso consiste en la inicialización aleatoria por única vez de las matrices de pesos, las cuales son los parámetros (θ) de nuestro modelo de red neuronal. En otras palabras, esto quiere decir que se llenan con valores numéricos las matrices W_{input} (o W) y W_{output} (o W'): los valores W_{11} a W_{83} y W'_{11} a W'_{38} respectivamente.
 2. Este paso consiste en la aplicación de forward propagation para obtener nuestra salida $-y_{pred softmax}-$. Este paso se divide en 2 sub-pasos: *Paso 2.1-Forward propagation (capa de entrada a capa oculta)* y *Paso 2.2-Forward propagation (capa oculta a capa de salida)*. Algo importante a tener en cuenta es que, debido a que estamos utilizando el algoritmo de gradiente descendiente básico y no SGD, en este paso N°2 tenemos que obtener los valores de $y_{pred softmax}$ de todos nuestros pares de palabras (central, contexto) para luego, teniendo estos resultados, pasar al paso N° 3. Es por esto que, *este paso N°2 (y en consecuencia los sub-pasos 2.1 y 2.2) se realizan por cada una de las 30 iteraciones que vimos en la Figura 7.4* hasta obtener 30 vectores $y_{pred softmax}$ (los cuales luego se compararán con los 30 vectores y_{true} correspondientes a la palabra de entrada de cada iteración).
- Iteración 1: Ingresa “the” como palabra central en la capa de entrada y se intenta predecir la palabra de contexto “quick” en la capa de salida. Ambas palabras son representadas como vectores one hot encoding (vectores X e y_{true} respectivamente). Esto lo observamos en la Figura 7.7. En la Figura 7.8 podemos observar un mayor detalle.

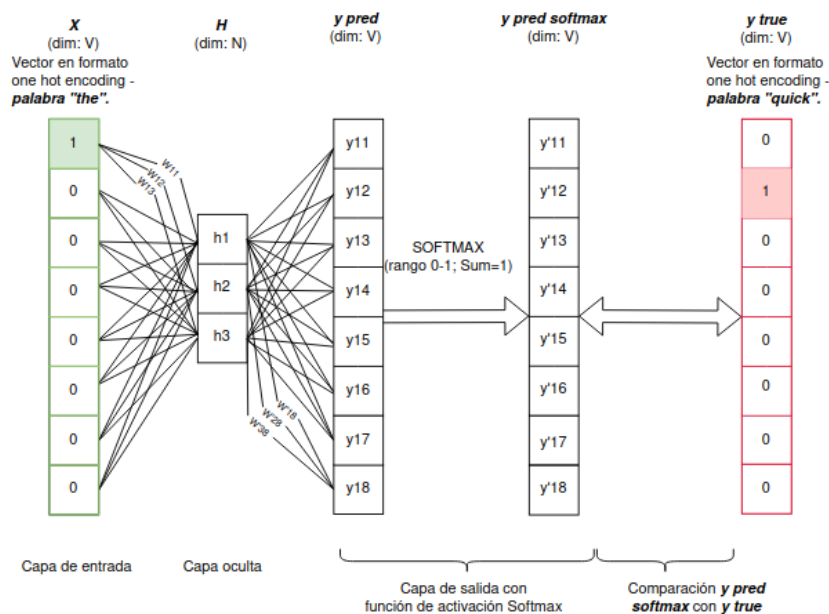


Figura 7.7: Primera iteración Skipgram del paso 2.

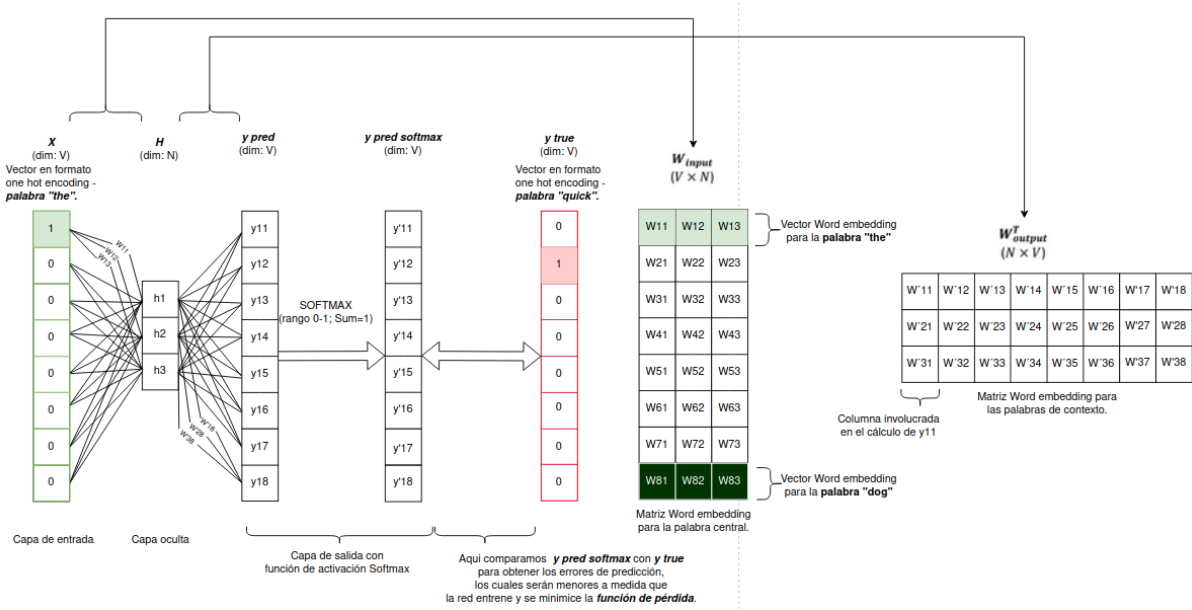


Figura 7.8: Primera iteración Skipgram del paso 2 en mayor detalle.

- *Paso 2.1 para la Iteración 1:* Forward propagation (capa de entrada a capa oculta): En este paso se calcula el valor de las neuronas h_1, h_2 y h_3 de la capa oculta H . Para esto se utiliza la palabra central de entrada "the" en conjunto con su fila correspondiente de la matriz de embedding W : la cual observando la Figura 7.8 podemos ver que corresponde a los valores W_{11}, W_{12} y W_{13} . Debido a que solo una posición del vector de entrada "the" es 1 (x_1) y los demás son 0 (x_2 a x_8), entonces los valores W_{11}, W_{12} y W_{13} son los valores que tomarán h_1, h_2 y h_3 respectivamente). En 16 observamos el cálculo necesario para la obtención de h_1 .

$$\begin{aligned}
 h_1 &= x_1.W_{11} + x_2.W_{21} + x_3.W_{31} + x_4.W_{41} + x_5.W_{51} \\
 h_1 &= x_1.W_{11} + 0 \\
 h_1 &= 1.W_{11} \\
 h_1 &= W_{11}
 \end{aligned} \tag{16}$$

Teniendo en cuenta este cálculo, h_2 y h_3 se calculan de la misma manera que h_1 , siendo $h_2 = W_{12}$ y $h_3 = W_{13}$. Entonces obtenemos $H = (W_{11}, W_{12}, W_{13})$.

- *Paso 2.2 para la Iteración 1:* Forward propagation (capa oculta a capa de salida): En este paso se calcula el valor de cada neurona de la capa de salida luego de pasar por la función de salida softmax (vector $y_{pred softmax}$ de la Figura 7.8). Debido a que la capa oculta no tiene función de activación, entonces los valores calculados en el *paso 2.1* pasan directamente a la capa de salida. De esta manera, los valores de las neuronas de la capa de salida se calculan multiplicando el vector H que nos dió anteriormente con la otra matriz W' -de tamaño $N \times V$ -. Esto nos dará un vector y_{pred} de tamaño

V. En 17 observamos el cálculo necesario para la obtención de y_{11} e y_{12} .

$$\begin{aligned} y_{11} &= h1.W'_{11} + h2.W'_{21} + h3.W'_{31} \\ y_{12} &= h1.W'_{12} + h2.W'_{22} + h3.W'_{32} \end{aligned} \quad (17)$$

Los cálculos de y_{13} a y_{18} siguen la misma lógica descrita en el cálculo 17. De esta manera, tendremos los valores para nuestro vector y_{pred} .

Por último, este vector alimentará a la función de salida softmax con el objetivo de obtener una distribución de probabilidades sobre el espacio de nuestro vocabulario - $y_{pred softmax}$ -, el cual se calculará, utilizando la Fórmula 14). Por ejemplo, en 18 observamos el cálculo necesario para la obtención de y'_{11} e y'_{12} .

$$\begin{aligned} y'_{11} &= \sigma(y_{11}) = \frac{\exp(y_{11})}{1 + \exp(y_{11})} \\ y'_{12} &= \sigma(y_{12}) = \frac{\exp(y_{12})}{1 + \exp(y_{12})} \end{aligned} \quad (18)$$

Los cálculos de y'_{13} a y'_{18} siguen la misma lógica descrita en el cálculo 18. De esta manera, tendremos los valores para nuestro vector $y_{pred softmax}$.

- Iteración N°2: Ingresa nuevamente “the” como palabra central en la capa de entrada. Pero ahora se intenta predecir la palabra de contexto “brown” en la capa de salida. Nuevamente ambas palabras son representadas como vectores one hot encoding (vectores X e y_{true} respectivamente): ver Figura 7.9.

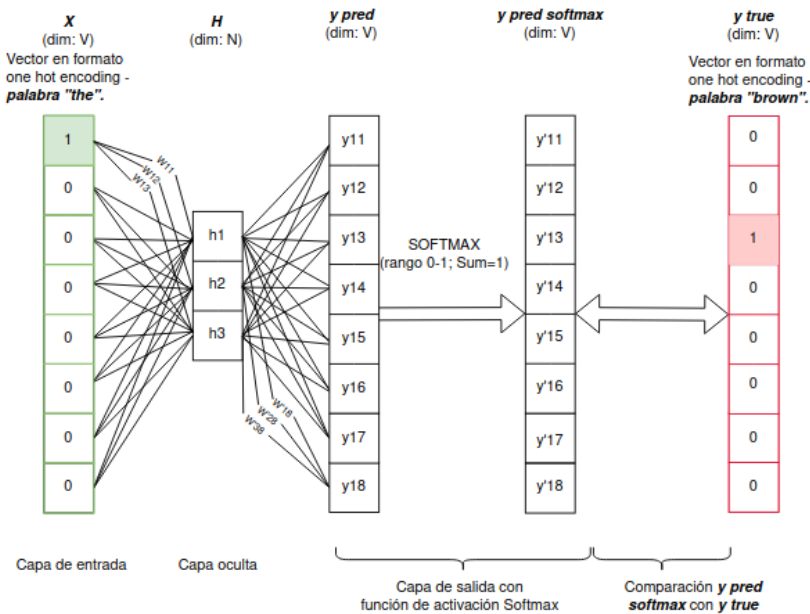


Figura 7.9: Segunda iteración Skipgram del paso 2 .

Para esta segunda iteración se realizan los *pasos 2.1 y 2.2* descritos anteriormente para la iteración 1, obteniendo así el vector $y_{pred softmax}$ correspondiente.

- Se realizarán 28 iteraciones más hasta llegar a la iteración N°30, en la cual se ingresará como entrada la palabra central “dog” y como vector y true tendremos la representación en formato one hot encoding de la palabra “lazy”. De esta manera, al finalizar el paso N°2 tendremos 30 vectores y *pred softmax* y 30 vectores y *true* con los que comparar.
3. Luego de obtener las distribuciones $-y$ *pred softmax*- (valor predicho) para cada combinación de palabras (central, contexto), estas son comparadas con los vectores de salida $-y$ *true*- correspondientes (que tienen las etiquetas reales y al igual que los vectores de entrada están codificados en formato one hot encoding), computándose de esta manera el *error (o loss)* de clasificación (entre el valor predicho y el real) mediante el cálculo de la *función de costo* $J(\theta)$ (Ver Fórmula R2) .

En la Figura 7.10 podemos observar un ejemplo numérico de cómo podría quedar conformado nuestro vector y *pred softmax* (donde la suma de los valores dá 1) cuando el vector de entrada es, por ejemplo “the”, y cómo se realizaría la comparación contra el vector de salida y *true* cuando la palabra es “quick” para obtener los errores de clasificación. Vemos que el primer resultado de nuestro entrenamiento fue medianamente bueno, ya que en nuestro vector y *pred softmax*, en la posición y ‘12, obtuvimos el mayor valor. Y esto es lo que queremos lograr, ya que si comparamos este valor con el vector y *true* en la misma posición, tiene un 1. La idea es que a medida que la red entrene y minimice la función de costo J , los valores de y *pred softmax* se acerquen cada vez más a los valores de y *true*: en nuestro caso que el valor 0,28 sea cada vez más cercanos al 1, y los otros 7 valores que sean cada vez más cercanos al 0.

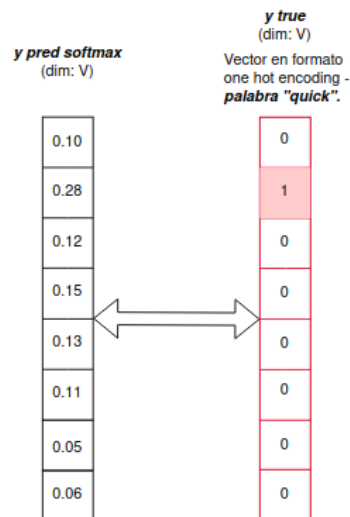


Figura 7.10: Comparando y *pred softmax* con y *true*.

4. Este paso consiste en aplicar el proceso backpropagation utilizando el algoritmo de optimización *descenso del gradiente* para poder ajustar / modificar las matrices W_{input} y W_{output} con el objetivo de minimizar la función de costo $J(\theta)$ obtenida en el paso 3, y con ello minimizar el error de clasificación.
5. Se debe volver al paso 2 hasta completar la cantidad de *epochs* seteados como hiper-parámetro de nuestro modelo.

Finalmente, luego de haber iterado $epoch$ veces sobre nuestro conjunto de entrenamiento y habiendo minimizado (idealmente) la función de costo J en cada iteración, lo que haremos es *obtener la matriz W* (de dimensión $V \times N$), la cual será *nuestra matriz de embeddings*: Figura 7.11 Observamos que, para cada fila de nuestra matriz W , tendremos el valor del word embedding para la palabra que tenga el mismo índice en nuestro vocabulario: por ejemplo para “the” que tiene índice 1, la 1er fila de W será su representación en word embedding.

W^{input}
($V \times N$)

W11	W12	W13	} Vector Word embedding para la palabra "the"
W21	W22	W23	
W31	W32	W33	
W41	W42	W43	
W51	W52	W53	
W61	W62	W63	
W71	W72	W73	
W81	W82	W83	} Vector Word embedding para la palabra "dog"

Matriz Word embedding
para la palabra central.

Figura 7.11: Matriz W' de embeddings objetivo.

De esta manera, entrenamos nuestra red neuronal no con el objetivo de predecir la palabra de contexto a partir de la palabra central, sino con el objetivo de tomar de esa red neuronal su matriz de pesos W' ya entrenada y utilizarla como nuestra matriz de embeddings.

7.3 Palabras repetidas en nuestro Corpus y palabras polisémicas.

A continuación responderemos a dos preguntas muy interesantes: *¿qué sucede en el entrenamiento con las palabras que aparecen más de una vez en nuestro Corpus?* y *¿qué sucede con las palabras polisémicas*³⁹?

Word2vec nos permite, en caso que la palabra aparezca N veces en nuestro corpus de texto, que el modelo aprenda los N contextos de esa palabra; o dicho de otra forma *permite aprender cuando se usa una misma palabra en diferentes contextos*. Además, si esa palabra tiene M significados (palabras polisémicas), lo que hará Word2vec es tener en cuenta estos M significados para armar un word embedding que “promedie” dichos significados.

En el ejemplo descrito en la sección del Anexo *Ejemplo de obtención de Word Embeddings mediante skipgram y softmax* esto ocurre con la palabra “the”, la cual ingresa al modelo 2 veces (siendo usada en 2 contextos distintos). En ese ejemplo, “the” es usada en contexto muy similares, por lo que su significado en este corpus también es similar, ya que ambas veces preceden a un adjetivo (“quick” y “lazy”).

Tomaremos un mejor ejemplo para explicar esto: consideremos que entrenamos nuestro algoritmo con el corpus de texto “She will *park* the car on the street so we can walk in the central *park*”. Luego de aplicar un pre-procesamiento y tokenización, nuestro *corpus tokenizado* quedará como: [“*park*”, “car”, “street”, “walk”, “central”, “*park*”]. Mientras que nuestro *vocabulario* será: [“*park*”, “car”, “street”, “walk”, “central”].

Al entrenar nuestro modelo con dicho *corpus tokenizado*, la palabra “park” ingresará al mismo 2 veces -con la misma representación en formato one-hot encoding: [1, 0, 0, 0, 0]-, ya que es usada en dos contextos diferentes. La diferencia con el anterior ejemplo, es que “park” en este corpus de texto tiene 2 significados muy distintos: el primero hace referencia a “estacionar” y el segundo a “parque”.

Al entrenar nuestro Word2vec con este *corpus tokenizado*, y considerando un tamaño de ventana de 1 ($C = 1$), obtendremos lo siguiente:

- En las iteraciones 1 y 2 *el modelo aprenderá que “park” aparece cerca de “car”* (ya que en la iteración N°1 la palabra central / de entrada es “park” y la palabra de contexto / salida es “car”: y para la iteración N°2 la palabra central / de entrada es “car” y las palabras de contexto / salida son “park” y “street”).
- Mientras que en las iteraciones 5 y 6 *el modelo aprenderá que “park” además aparece cerca de “central”* (ya que en la iteración N°5 la palabra central / de entrada es “central” y las palabras de contexto / salida son “walk” y “park”; y para la iteración N°6 la palabra central / de entrada es “park” y la palabra de contexto/salida es “central”).

De esta manera, luego de realizar el entrenamiento, *implícitamente* nuestro modelo Word2vec aprenderá los 2 significados de “park” (debido al entrenamiento en distintos contextos en las iteraciones 1,2,5,6 previamente mencionadas). En conclusión,

³⁹Footnote: Palabras que tienen más de un significado

obtendremos el vector word embedding para “park” que tendrá en cuenta los N contextos en lo que se usó dicha palabra y sus M distintos significados (en este caso N y $M = 2$). Este vector permitirá representar razonablemente el *significado promedio* de “park” teniendo en cuenta sus 2 significados dentro del corpus (estacionar y parque); tendiendo a moverse en las coordenadas entre estos 2 significados.

Referencias

- [1] Pinky Sitikhu, Kritish Pahi, Pujan Thapa, & Subarna Shakya. (2019, Octubre). *A Comparison of Semantic Similarity Methods for Maximum Human Interpretability*. IEEE International Conference on Artificial Intelligence for Transforming Business and Society. (pp. 1-4).
- [2] World Economic Forum. (2020, Octubre). *The Future of Jobs Report*. (pp. 29-31).
- [3] Matt Kusner, Yu Sun, Nicholas Kolkin, & Kilian Weinberger. (2015, Julio). *From word embeddings to document distances*. International Conference on Machine Learning. (pp. 957-966).
- [4] Jiapeng Wang, & Yihong Dong. (2020, Agosto). *Measurement of Text Similarity: A Survey*. Information 2020. 11(9). (pp. 1-8, 13,14).
- [5] Baoli Li, & Liping Han. (2013, Octubre). *Distance Weighted Cosine Similarity Measure for Text Classification*. IDEAL2013 Conference. (pp. 1,2).
- [6] Chunjie Luo, Jianfeng Zhan, Lei Wang, & Qiang Yang. (2017, Octubre). *Cosine Normalization: Using Cosine Similarity Instead of Dot Product in Neural Networks*. (pp. 1,2).
- [7] Dani Gunawan, C A Sembiring, & Mohammad Andri Budiman. (2018, Marzo). *The Implementation of Cosine Similarity to Calculate Text Relevance between Two Documents*. 2nd International Conference on Computing and Applied Informatics 2017. Journal of Physics Conference Series. 978(1). (pp. 1-2).
- [8] Derek S. Chapman, & Jane Webster. (2003, Junio-Septiembre). *The Use of Technologies in the Recruiting, Screening, and Selection Processes for Job Candidates*. International journal of selection and assessment. 11(2/3). (pp. 113-114, 117-119).
- [9] Pshdar Abdalla Hamza, Baban Jabbar Othman, Bayar Gardi, Sarhang Sorguli, Hassan Mahmood Aziz, Shahla Ali Ahmed, Bawan Yassin Sabir, Nechirwan Burhan Ismael, Bayad Jamal Ali, & Govand Anwar. (2021, Mayo-Junio). *Recruitment and Selection: The Relationship between Recruitment and Selection with Organizational Performance*. International journal of Engineering, Business and Management (IJEEM). 5(3). (pp. 1-6).
- [10] Luis Argerich, Natalia Golmar, Damián Martinelli, Martín Ramos Mejía, & Juan Andrés Laura. (2019, Enero). *75.06, 95.58 Organización de Datos*. Apunte del Curso Organización de Datos, Universidad de Buenos Aires, Facultad de Ingenieria. (pp. 4-8,351,352,377-379, 387-389,429-431,470-477).
- [11] Ladders Company. (2018). *Eye-Tracking Study*. (pp. 2,6).
- [12] Riza Tanaz Fareed, Sharadadevi Kaganurmuth, & Rajath V. (2021, Agosto). *Resume Classification and Ranking using KNN and Cosine Similarity*. International Journal of Engineering Research & Technology (IJERT). 10(8). (pp. 192-194).

- [13] Senthil Kumaran V, & Annamalai Sankar. (2013, Mayo). *Towards an automated system for intelligent screening of candidates for recruitment using ontology mapping (EXPERT)*. International Journal of Metadata, Semantics and Ontologies. 8(1). (pp. 56-64).
- [14] Nuno Silva, & Joao Rocha. (2003). *Ontology Mapping for Interoperability in Semantic Web*, IADIS International Conference WWW/Internet (ICWI), Portugal. (pp. 1).
- [15] Wahiba Ben Abdesslem Karaa, & Nouha Mhimdi. (2011) *Using ontology for resume annotation*. International Journal of Metadata, Semantics and Ontologies. 6(3). (pp. 166-174).
- [16] Duygu Çelik, Askýn Karakas, Gülsen Bal, Cem Gültunca, Atilla Elçi, Basak Buluz, & Murat Can Alevli. (2013, Septiembre). *Towards an Information Extraction System Based on Ontology to Match Resumes and Jobs*. Computer Software and Applications Conference Workshops (COMPSACW), IEEE 37th. (pp. 333-338).
- [17] Frank Färber, Tim Weitzel, & Tobias Keim. (2003, Agosto). *An automated recommendation approach to selection in personnel recruitment*. 9th Americas Conference on Information Systems (AMCIS). (pp. 1-11).
- [18] Chirag Daryania, Gurneet Singh Chhabrab, Harsh Patel, Indrajeet Kaur Chhabrad, & Ruchi Patel. (2020). *An Automated Resume Screening System using Natural Language Processing and Similarity*. Topics In Intelligent Computing And Industry Design. 2(2). (pp. 99-103).
- [19] Juneja Afzal Ayub Zubeda, Momin Adnan Ayyas Shaheen, Gunduka Rakesh Narsayya Godavari, & Sayed ZainulAbideen Mohd Sadiq Naseem. (2016, Mayo). *Resume Ranking using NLP and Machine Learning*. Proyecto de Tesis para carrera de grado *Bachiller en Ingeniería*. School of Engineering and Technology Anjuman-I-Islam's Kalsekar Technical Campus. (pp. 1-3).
- [20] Jai Janyani, Kartik Agarwal, & Abhishek Sharma. (2018). *Automated Resume Screening System*. Proyecto de Tesis para carrera de grado *Bachiller en Tecnología*. Rajasthan Technical University. (pp. 5, 9-15).
- [21] V. V. Dixit, Trisha Patel, Nidhi Deshpande, & Kamini Sonawane. (2019, Abril). *Resume Sorting using Artificial Intelligence*. International Journal of Research in Engineering, Science and Management. 2(4). (pp. 423-425).
- [22] Dr. K.Satheesh, A.Jahnavi, L Aishwarya, K.Ayesha, G Bhanu Shekhar, & K.Hanisha. (2020). *Resume Ranking based on Job Description using SpaCy NER model*. International Research Journal of Engineering and Technology (IRJET). 7(5). (pp. 74-77).
- [23] Paolo Montuschi, Valentina Gatteschi, Fabrizio Lamberti, Andrea Sanna, & Claudio Demartini. (2014, Septiembre-Octubre). *Job recruitment and job seeking processes: how technology can help*. IT Professional. 16(5). (pp. 41-49).
- [24] Leila Yahiaoui, Zizette Boufaïda, & Yannick Prié. (2006). *Semantic Annotation of Documents Applied to E-Recruitment*. SWAP 2006, the 3rd Italian Semantic Web Workshop. (pp. 1-6).

- [25] Rémy Kessler, Nicolas Béchet, Mathieu Roche, Juan Manuel Torres-Moreno, & Marc El-Bèze. (2012). *A hybrid approach to managing job offers and candidates*. Information Processing & Management. 48(6). (pp. 1124-1135).
- [26] Pradeep Kumar Roy, Sarabjeet Singh Chowdhary, & Rocky Bhatia. (2020). *A Machine Learning approach for automation of Resume Recommendation system*. International Conference on Computational Intelligence and Data Science (ICCIDIS). Procedia Computer Science. 167. (pp. 2318-2327).
- [27] Ioannis Paparrizos, B. Barla Cambazoglu, & Aristides Gionis. (2011). *Machine learned job recommendation*. 5th ACM Conference on Recommender Systems, ACM. (pp. 325-328).
- [28] Paul Resnick, & Hal R. Varian. (1997). *Recommender Systems*. Communications of the ACM40. (pp. 56-59).
- [29] M. Emre Celebi, Michael W. Berry, Azlinah Mohamed, & Bee Wah Yap. (2020). Libro *Supervised and Unsupervised Learning for Data Science*. (pp. 3,4,14,15).
- [30] Karthik Tangirala. (2011). *Semi-supervised and transductive learning algorithms for predicting alternative splicing events in genes*. Proyecto de Tesis para carrera de grado *Master of Science*. Kansas State University. (pp. 1-4).
- [31] Daniel Berra. (2018). *Cross-Validation*. Libro *Reference Module in Life Sciences*. Data Science Laboratory, Tokyo Institute of Technology. (pp. 1-5).
- [32] S. B. Kotsiantis, D. Kanellopoulos, & P. E. Pintelas. (2006). *Data Preprocessing for Supervised Learning*. International Journal of Computer Science. 1(1). (pp. 111,112,116).
- [33] Stephen Bradshaw, & Colm O’Riordan. (2018). *Evaluating Better Document Representation in Clustering with Varying Complexity*. In Proceedings of the 10th International Joint Conference on Knowledge Discovery, Knowledge Engineering and Knowledge Management 2018. 1. (pp. 199-201).
- [34] Julio-Omar Palacio-Niño, & Fernando Berzal. (2019, Mayo). *Evaluation Metrics for Unsupervised Learning Algorithms*. (pp. 1-6).
- [35] Nathalie Japkowicz. (2006, Mayo). *Why Question Machine Learning Evaluation Methods (An Illustrative Review of the Shortcomings of Current Methods)*. (pp. 1-5)
- [36] Alexei Botchkarev. (2018, Septiembre). *Performance Metrics (Error Measures) in Machine Learning Regression, Forecasting and Prognostics: Properties and Typology*. Interdisciplinary Journal of Information, Knowledge, and Management, 2019. 14. (pp. 1-8).
- [37] Haider Khalaf Jabbar, & Rafiqul Zaman Khan. (2014). *Methods to avoid over-fitting and under-fitting in supervised machine learning (comparative study)*. (pp. 163-165)
- [38] Zhou Yong, Li Youwen, & Xia Shixiong. (2009, Marzo). *An Improved KNN Text Classification Algorithm Based on Clustering*. Journal of Computers. 4(3). (pp. 230, 233).

- [39] Haneen Arafat Abu Alfeilat, Ahmad B.A. Hassanat, Omar Lasassmeh, Ahmad S. Tarawneh, Mahmoud Bashir Alhasanat, Hamzeh S. Eyal Salman, & V.B. Surya Prasath. (2019, Diciembre). *Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review*. Libro *Big Data*. 7(4). (pp. 1-9).
- [40] Boyang Li. (2018, Febrero). *An Experiment of K-Means Initialization Strategies on Handwritten Digits Dataset*. Intelligent Information Management. 10. (pp. 43-46).
- [41] Mengyao Cui. (2020). *Introduction to the K-Means Clustering Algorithm Based on the Elbow Method*. Accounting, Auditing and Finance. 1. (pp. 5-8).
- [42] Shraddha Shukla, & Naganna S. (2014). *A Review On K-means Data Clustering Approach*. International Journal of Information & Computation Technology. 4(17). (pp. 1847-1849, 1852, 1853).
- [43] Kodinariya, T.M. (2014). *Survey on Exiting Method for Selecting Initial Centroids in K-Means Clustering*. International Journal of Engineering Development and Research. 4(2). (pp. 2865-2868).
- [44] David Arthur, & Sergei Vassilvitskii. (2007, Enero). *K-Means++: The Advantages of Careful Seeding*. SODA '07: Proceedings of the 18th annual ACM-SIAM symposium on Discrete algorithms. (pp. 1027-1035).
- [45] César Menacho Ch. (2013,Diciembre). *Modelos de regresión lineal con redes neuronales (Lineal regression models with neural networks)*. Anales Científicos. 75(2). (2014). (pp. 253-259).
- [46] Andrea Apicella, Francesco Donnarumma, Francesco Isgrò, & Roberto Prevete. (2021,Febrero). *A survey on modern trainable activation functions*. (pp. 1-25).
- [47] Damián Jorge Matich. (2001,Marzo). *Redes Neuronales: Conceptos Básicos y Aplicaciones*. Trabajo de Investigación. Cátedra Informática Aplicada a la Ingeniería de Procesos – Orientación I. Universidad Tecnológica Nacional, Facultad Regional Rosario. (pp. 4-28).
- [48] Juan José Montaña Moreno. (2002). *Redes Neuronales Artificiales aplicadas al Análisis de Datos*. Proyecto de Tesis Doctoral. Universitat de les illes balears, Mallorca. Facultad de Psicología. (pp. 17-47).
- [49] Ariel E. Repetur. (2019, Abril). *Redes Neuronales Artificiales*. Trabajo final para carrera Licenciatura en Ciencias Matemáticas. Universidad Nacional del Centro de la Provincia de Buenos Aires, Facultad de Ciencias Exactas. (pp. 5-19,41-43).