Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

Trabajo de Fin de Grado

Grado en Ingeniería Informática

Procesamiento de Lenguaje Natural en información postal

Natural Language Processing for address information

Jorge Cabrera Rodríguez

San Cristóbal de La Laguna, 7 de julio de 2023

Dña. **Isabel Sánchez Berriel**, con N.I.F. 42.885.838-S profesora contratada doctora tipo 1 adscrita al Departamento de Ingeniería informática y de sistemas de la Universidad de La Laguna, como tutora

Dña. **Luz Marina Moreno de Antonio**, con N.I.F. 45.457.492-Q profesora contratada doctora tipo 1 adscrita al Departamento de Ingeniería informática y de sistemas de la Universidad de La Laguna, como cotutora

C E R T I F I C A N

Que la presente memoria titulada:

"Procesamiento de Lenguaje Natural en información postal"

ha sido realizada bajo su dirección por D. **Jorge Cabrera Rodríguez**,

con N.I.F. 43.488.362-P.

Y para que así conste, en cumplimiento de la legislación vigente y a los efectos oportunos firman la presente en La Laguna a 7 de julio de 2023

Agradecimientos

A mis padres, por haberme dado la educación, el amor y los valores para llegar hasta donde estoy hoy.

A Isabel y Luz Marina, por haberme guiado en este proyecto educativo de la mejor forma posible.

A los pibes de *Destiny Crew,* por haberme aguantado y apoyado en mis momentos más bajos.

A mi tía Yaya, por haber sido como una segunda madre todos estos años.

Licencia

Resumen

El objetivo de este trabajo ha sido diseñar e implementar un modelo de coincidencia de direcciones que permita identificar direcciones equivalentes (direcciones escritas de diferente forma que dirigen al mismo lugar geográfico). Para ello, se han investigado técnicas de Procesamiento de Lenguaje Natural (PLN) en conjunto con algoritmos y modelos de aprendizaje automático.

Se han implementado tanto técnicas de clasificación como técnicas de agrupamiento para alcanzar un modelo funcional. Se destacan los algoritmos de clasificación "Random Forest", "Naive Bayes Gaussiano" y "XGBoost". En el lado del agrupamiento, se utilizó el algoritmo "K-medias".

El paradigma de trabajo ha sido realizar coincidencias binarias tales que, para una dirección dada, se determinen para otras direcciones los siguientes valores: MATCH si la dirección a comparar es equivalente y NO MATCH si la dirección a comparar no es equivalente.

Tras múltiples operaciones de revisión y tras múltiples técnicas de corrección de datos, generación de direcciones variadas y sobre-muestreo de direcciones, se ha obtenido un modelo práctico funcional, que demuestra un excelente rendimiento al realizar coincidencia de direcciones.

Palabras clave: PLN, Aprendizaje automático, dirección, clasificación, agrupamiento, *XGBoost*, *Random Forest*, *Naive Bayes*, Gaussiano.

Abstract

The objective of this work has been to design and implement an address matching model that can identify equivalent addresses (addresses written in different formats that refer to the same geographic location). To achieve this, techniques from Natural Language Processing (NLP) have been investigated in conjunction with machine learning algorithms and models.

Both classification techniques and clustering techniques have been implemented to create a functional model. Notable classification algorithms used include "Random Forest," "Gaussian Naive Bayes," and "XGBoost." On the clustering side, the "K-means" algorithm was utilized.

The working paradigm has been to perform binary matches, where given a reference address, other addresses are determined to have either a MATCH or NO MATCH value. A MATCH indicates equivalence with the reference address, while a NO MATCH indicates non-equivalence.

Through multiple rounds of revision, data correction techniques, generation of diverse addresses, and oversampling of addresses, a practical and functional model has been obtained. It demonstrates excellent performance in address matching tasks.

Keywords: *NLP*, *Machine Learning*, *address*, *classification*, *clustering*, *XGBoost*, *Random Forest*, *Naive Bayes*, *Gaussian*.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Introducción

Este proyecto busca implementar un sistema de concordancia de direcciones *(address matching)* mediante algoritmos de aprendizaje automático *(machine learning)* capaces de identificar la coincidencia de una nueva dirección entre una de las contenidas en el conjunto de datos *(dataset)* de entrenamiento con un cierto grado de certeza. Para ello, se cuenta con una base de datos de información georreferenciada de la Comunidad Autónoma de Canarias, un sistema de georreferenciación para fines estadísticos creado por el Instituto de Estadística de Canarias (ISTAC) [1]. Esta base de datos contiene información preprocesada, normalizada y lista para ser utilizada referente a vías, portales, edificios, municipios y demás valores relevantes a nivel geográfico del archipiélago.

Con el potencial que esta información brinda a nivel de procesamiento, se busca desarrollar un sistema que, dada una dirección cualquiera (esté dentro del conjunto de datos o no), indique con un cierto grado de certeza la coincidencia con alguna dirección contenida dentro de la base de datos, si hubiera alguna.

Este proceso es conocido como concordancia de direcciones y se divide en tres fases diferentes [2]:

- Fase de análisis *(parsing)*: se identifican los elementos clave en una cadena de texto de una dirección, como pueda ser el tipo de vía en la misma, la vía a la que hace referencia, el municipio si lo hubiera, etc.
- Fase de normalizado *(normalization)*: se toman los elementos clave separados de la fase de análisis y se modifican para que sigan unas normas de concordancia, para que sean todos compatibles a la hora de ser utilizados.
- Fase de correspondencia *(matching)*: se aplican diferentes técnicas entre las direcciones normalizadas para obtener direcciones parejas.

En el caso del conjunto de datos de partida, las operaciones de análisis y normalizado ya han sido realizadas, por lo que se requiere aplicar técnicas de identificación de direcciones. Este problema ya ha sido planteado y resuelto por el Instituto Canario de Estadística (ISTAC) mediante la aplicación de técnicas basadas en distancias entre direcciones, como puedan ser:

- Distancia euclídea
- Similitud por coseno
- Distancia de *Levenshtein*

Lo que se busca en este proyecto es resolver este problema mediante técnicas de aprendizaje automático, siguiendo como ejemplo directo los artículos [2] y [3], donde se propone resolver el problema de concordancia de direcciones utilizando modelos de aprendizaje automático.

Capítulo 2. Estado del arte

2.1 Algoritmos de concordancia usados por el ISTAC

Este proyecto toma como base de datos de estudio aquella la analizada y preprocesada por el ISTAC presentada en su estudio "Sistema de georreferenciación para fines estadísticos" [1]. En este artículo se explica el trabajo que ha desarrollado para generar una base de datos de direcciones georreferenciadas de todo el archipiélago canario.

Para ello ha aplicado diferentes procesos de normalización de datos a la información en bruto para producir campos que sirvan en el proceso de georreferenciación. Acto seguido se han implementado diferentes algoritmos de georreferenciación, desde algoritmos determinísticos basados en similitud hasta algoritmos probabilísticos basados en APIs comerciales pasando por algoritmos aleatorios en los que se asignan direcciones por "cercanía" a zonas con un mayor tamaño (véase tamaño como un valor calculado según el número de personas, la población activa, etc.).

2.2 Algoritmos de concordancia mediante aprendizaje automático

Existen artículos sobre concordancia de direcciones que siguen los mismos pasos de normalización llevados a cabo en el artículo del ISTAC, pero que difieren en las técnicas utilizadas a la hora de establecer las correspondencias de las direcciones. Un ejemplo de ello es *"Postmatch A Framework for Efficient Address Matching"* [2], un artículo sobre concordancia de direcciones donde se usan técnicas de aprendizaje automático utilizando como conjunto de pruebas una base de datos de direcciones australianas que prescinde de los algoritmos por similitud entre sentencias anteriormente mencionados.

En su lugar, *Postmatch* aplica diferentes técnicas de preprocesado de PLN a las direcciones del conjunto de datos, así como la distancia de edición de *Jaro-Winkler* [4] para obtener sus correspondientes representaciones numéricas. Con estas representaciones se entrena un modelo de clasificación con el algoritmo *XGBoost* para determinar la concordancia entre dos listas de direcciones. Además, indican mediante métricas como la exactitud *(accuracy)*, la precisión *(precisión)* o la sensibilidad *(recall)* [5] la eficiencia del modelo respecto a un conjunto de datos de prueba *(testing)*.

Además del anterior artículo, en *"Machine Learning Innovations in Address Matching: A practical comparison of word2vec and CRFs"* [3] se utiliza el algoritmo CRF para obtener la segmentación de la dirección. Para generar la representación numérica de las direcciones se usa la distancia de *Jaro-Winkler* entre los campos de la dirección. Finalmente se resuelve un problema de clasificación MATCH/NO-MATCH solo entre direcciones que comparten el mismo código postal

para reducir el coste computacional. Se usa un ensamblaje de clasificadores supervisados binarios (MATCH, NO-MATCH) *XGBoost* y *Random Forest*. En este trabajo también se introduce una representación numérica basada en vectores de palabras generados con *word2vec*, con los que se resuelve posteriormente el problema de clasificación MATCH/NO-MATCH.

Basándose en los anteriores trabajos, el trabajo de Final de Máster *"Machine Learning and NLP approaches in address matching"* [6] plantea alternativas a los modelos probabilísticos y de similitud implementados por el ISTAC: propone un procesamiento de los datos mediante técnicas de PLN similares a las de *Postmatch*. En [6] se analizan diferentes algoritmos para generar los vectores de representación de palabras, como *word2vec* [7], *doc2vec* [8], *FastText* [9] o *BERT* [10] pero con un subconjunto reducido de direcciones del conjunto de datos original. El modelo resultante obtiene valores de precisión de 99% y sensibilidad de 93%.

2.3 Técnicas de aprendizaje automático

Las técnicas de aprendizaje automático se dividen en dos tipos [11]:

- **Aprendizaje supervisado:** en este tipo de técnicas, el conjunto de datos con el que se alimenta al modelo está etiquetado, por lo que para cada entrada existe una serie de valores y etiquetas. Los modelos de aprendizaje supervisado deben ser capaces de aprender de la relación entrada-etiqueta para predecir las etiquetas correctas para nuevas entradas no vistas.
- **Aprendizaje no supervisado:** en este tipo de técnicas, el conjunto de datos con el que se alimenta al modelo carece de etiqueta o resultado alguno. El modelo, en este caso, se encarga de extraer patrones de similitud entre diferentes entradas del conjunto de datos. Esto permite al modelo reconocer los patrones de similitud de una nueva entrada con las entradas existentes.

En este proyecto se aplicarán tanto técnicas de aprendizaje supervisado (modelos de clasificación) como técnicas de aprendizaje no supervisado (modelos de agrupamiento).

2.3.1 Clasificación

Los algoritmos de clasificación son un tipo de algoritmo de aprendizaje automático que se encargan de recibir datos etiquetados y entrenarse con ellos, de tal forma que cuando el modelo reciba una nueva entrada sea capaz de predecir los valores de estas etiquetas de forma correcta.

Los modelos de clasificación requieren de valores numéricos (valores discretos, vectores, matrices, etc.) para poder operar. Existen ciertos algoritmos de clasificación utilizados en aprendizaje automático que sirven para clasificar vectores numéricos de forma eficiente, como puedan ser:

- Clasificador *Random Forest*
- Clasificador *Naive Bayes* Gaussiano
- Clasificador *XGBoost*

Random Forest

El algoritmo *Random Forest* [12] es un algoritmo de clasificación que se entrena a base de generar múltiples árboles de decisión en diferentes muestras del conjunto de datos en cuestión. De estos árboles resultantes se escogen aquellos cuyas métricas de rendimiento sean más altas,

para luego obtener árboles promedio entre estos.

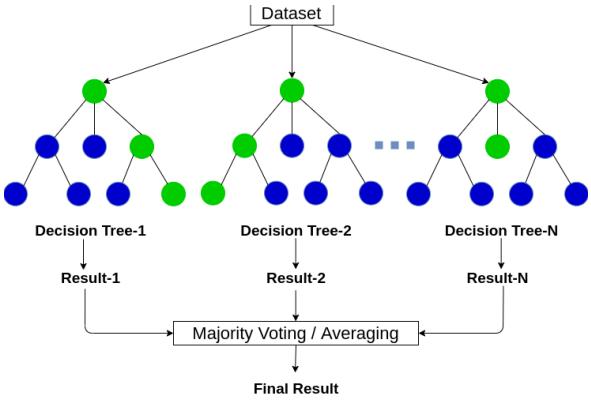

Ilustración 1. Representación de árboles de decisión de algoritmo "Random Forest" [13].

Naive Bayes Gaussiano

El algoritmo *Naive Bayes* Gaussiano *(Gaussian Naive Bayes)* [14] se basa en la suposición de que la distribución de entradas de una clase cualquiera sigue una distribución normal respecto a todo el resto de las clases del conjunto de datos. De esta forma, para un nuevo dato a clasificar, se puede determinar su clase correspondiente como la clase en el espacio vectorial en la que el elemento tenga un valor de decisión mayor.

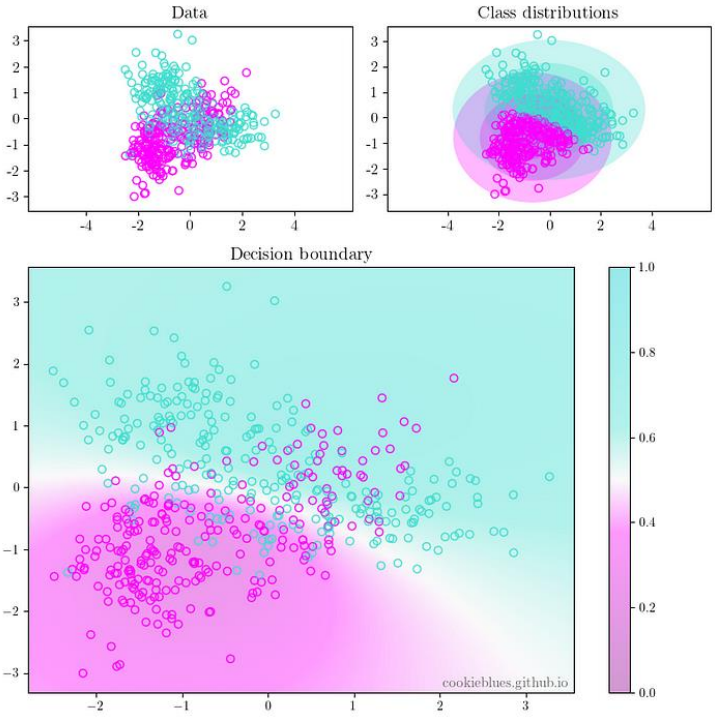

Ilustración 2. Distribución de clases en Gaussian NB [14].

XGBoost

El algoritmo *XGBoost* [15] es un algoritmo similar al *Random Forest* en cuanto a la mejora progresiva de los clasificadores (árboles) en la fase de entrenamiento. La diferencia frente al *Random Forest* es que este algoritmo aplica múltiples operaciones adicionales para optimizar la mejoría del clasificador, como puedan ser:

- Cálculos mediante algoritmo de *newton-raphson*.
- Disminución de nodos hoja de árboles de decisión.
- Penalización de árboles con peores resultados.
- Posibilidad de implementación mediante sistemas distribuidos.

El algoritmo *XGBoost* requiere que todas las clases de un problema de clasificación se encuentren numeradas de forma secuencial, comenzando por el valor 0. De esta forma, si se cuenta con N clases, estas deben estar etiquetadas según la secuencia $[0, N)$.

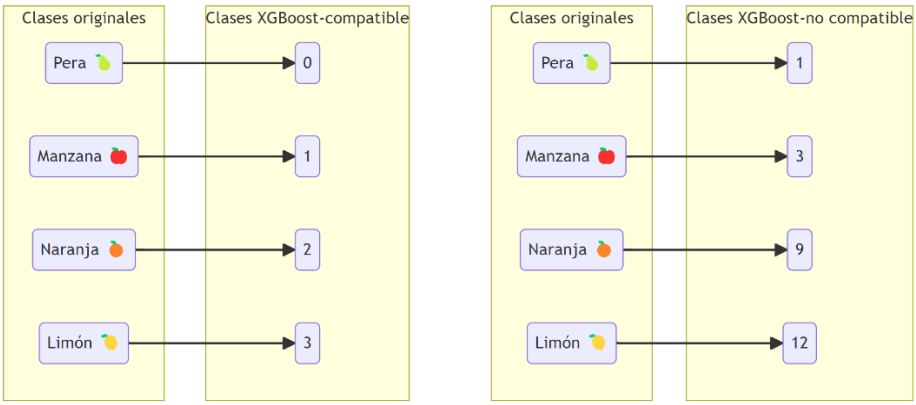

Ilustración 3. Requerimiento de clases de algoritmo XGBoost.

2.3.2 Agrupamiento

Los modelos de agrupamiento son técnicas de aprendizaje no supervisado y se encargan de agrupar elementos de un conjunto de datos en diferentes grupos *(clusters)*, donde los elementos dentro de cada grupo son más "similares" entre sí que con los elementos de otros grupos.

Se tratan de modelos no supervisados porque, a diferencia de los modelos de clasificación, estos no tienen clases o etiquetas especificados por un usuario. En su lugar, los modelos de agrupamiento generan sus propias etiquetas acorde al número de grupos generado.

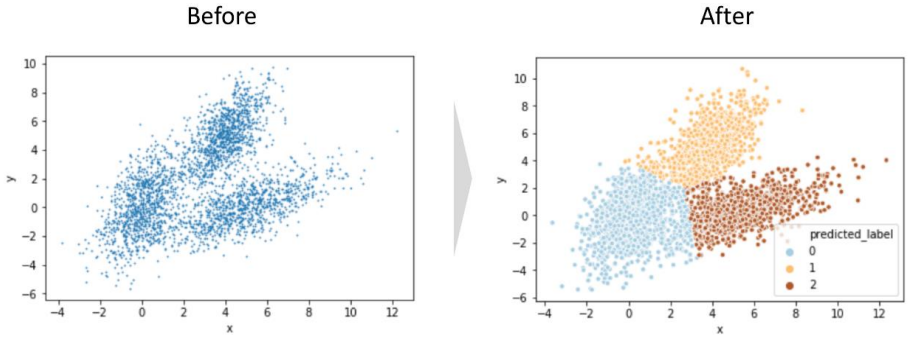

Ilustración 4. Funcionamiento de modelo de agrupamiento [16].

Dependiendo de la distribución de los datos a utilizar, puede ser más conveniente emplear un determinado algoritmo de agrupamiento.

Algunos ejemplos de algoritmos de agrupamiento son:

- Algoritmo k-medias

- DBSCAN

Algoritmo K-medias

El algoritmo K-medias *(K-means)* [17] es un algoritmo de agrupamiento que se basa en encontrar k centroides que representan los centroides de los grupos. El centroide de un grupo es un punto equidistante a los restantes puntos del grupo. El objetivo del algoritmo es crear los grupos de forma que la distancia entre los elementos del grupo sea la menor posible entre ellos y máxima con respecto a los miembros de otros grupos. Los datos se asignan al grupo cuyo centroide esté más cerca en términos de distancia euclídea. En cada iteración se actualiza el centroide. El algoritmo itera hasta converger y optimizar la asignación de los puntos a los grupos.

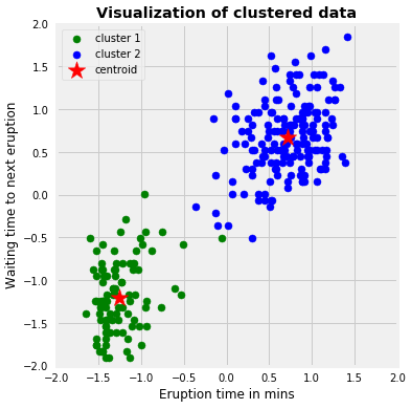

Ilustración 5. Visualización del funcionamiento de K-medias [17].

El algoritmo K-medias es frecuentemente utilizado en problemas con conjuntos de datos homogéneos, que no denoten ninguna forma geométrica en ningún plano, pues suele dar mejores resultados [18].

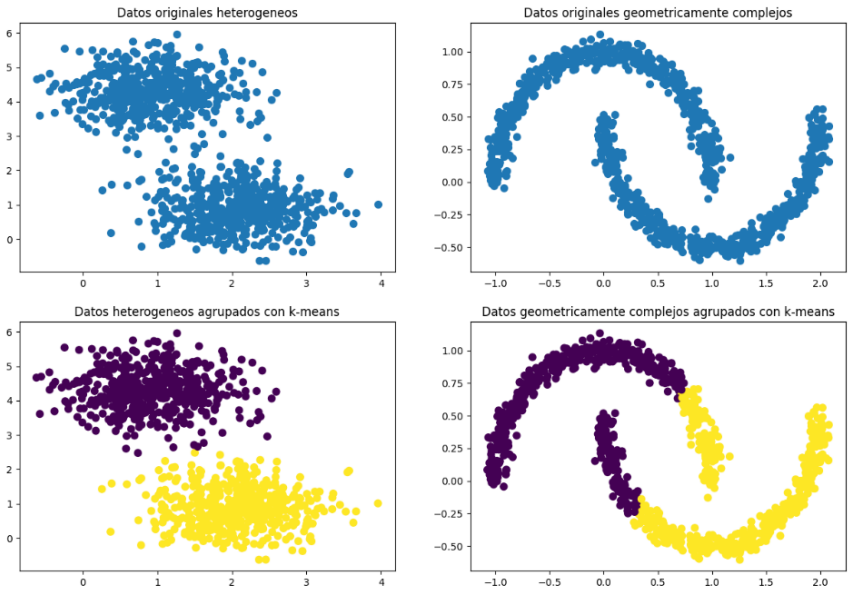

Ilustración 6. Ejemplo de K-means con diferentes distribuciones de datos.

Algoritmo DBSCAN

El algoritmo DBSCAN *(Density-based spatial clustering of applications with noise)* [19] es un algoritmo de agrupamiento cuyo funcionamiento se basa en el axioma de que un determinado

punto de un espacio dimensional pertenece a un grupo si es suficientemente cercano a muchos de los puntos de dicho grupo. Difiere del K-medias en que este algoritmo, para cada punto sin agrupar, realiza los cálculos con el punto más cercano al mismo dentro de una cierta distancia máxima, en vez de con el centroide más cercano como el algoritmo anterior.

El algoritmo permite ajustar la distancia mínima a considerar para que un punto pertenezca a un grupo, así como el número mínimo de puntos próximos para considerarse un grupo.

Este algoritmo es frecuentemente utilizado en problemas con conjuntos de datos heterogéneos, que denoten formas o patrones geométricos.

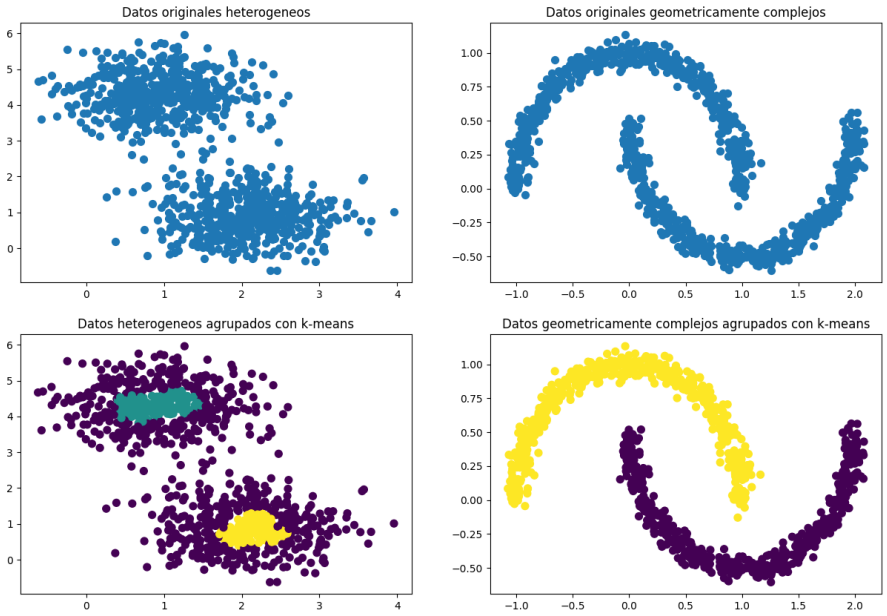

Ilustración 7. Ejemplo de DBSCAN con diferentes distribuciones de datos.

2.3.3 Métricas de rendimiento

A la hora de determinar cómo de eficiente o capaz es un algoritmo de clasificación, se suelen seguir múltiples métricas de rendimiento que permiten detectar diferentes puntos fuertes o débiles de un algoritmo. Las métricas más relevantes a la hora de analizar un algoritmo de clasificación son las siguientes:

- Exactitud
- Precisión
- Sensibilidad
- Matriz de confusión

Exactitud

La exactitud *(accuracy)* [5] es una métrica que indica el número de aciertos que ha tenido un clasificador respecto al número de elementos del conjunto de datos de prueba.

$$Accuracy = \frac{TrueNegatives + TruePositives}{TruePositives + FalsePositives + TrueNegatives + FalseNegatives}$$

Frecuentemente se piensa que una exactitud alta indica de forma inequívoca que un

modelo funciona correctamente. Sin embargo, puede que una clase en cuestión se clasifique mejor que el resto, caso que la exactitud no puede contemplar. Por ello son necesarias más métricas de rendimiento.

Precisión

La precisión *(precision)* [5] es una métrica que indica para una determinada clase (habitualmente la que se denomine como "clase correcta" o "clase objetivo") cuántos elementos del conjunto de datos que fueron clasificados como dicha clase pertenecen de verdad a la clase.

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives}$$

Un valor bajo de precisión sirve para determinar que una clase está englobando más muestras de las que le corresponden.

Sensibilidad

La sensibilidad *(recall)* [5] es una métrica que indica cuántas instancias de una determinada clase fueron clasificadas correctamente como dicha clase en el proceso de prueba.

$$Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

Un valor bajo de sensibilidad sirve como contraposición a la precisión, pues indica que una determinada clase no es suficientemente reconocible como para ser clasificada, por lo que está siendo englobada dentro de otras clases.

Matriz de confusión

La matriz de confusión *(confusion matrix)* [20] es una métrica que indica, para un modelo de clasificación dado, las clases disponibles frente a sus predicciones. En una matriz de confusión cada fila representa la instancia de una determinada clase, mientras que cada columna representa las predicciones de una determinada clase.

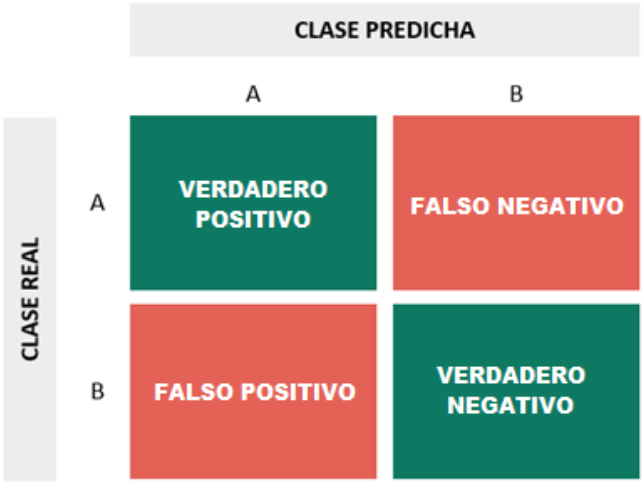

Ilustración 8. Representación gráfica de matriz de confusión.

La matriz de confusión recoge toda la información necesaria para el cálculo de las métricas anteriores. Además, de sus análisis se puede determinar si existe desbalanceo, así como datos anómalos que tiendan a clasificarse dentro de cierta clase.

2.3.4 Problemas en aprendizaje automático

Existen ciertos problemas a la hora de entrenar un modelo de aprendizaje automático que pueden comprometer su funcionamiento resultando en métricas de rendimiento bajas, anómalas, etc. Los problemas más notorios son:

- Sobreajuste
- Desbalance de datos

Sobreajuste

El sobreajuste se define como un excesivo aprendizaje respecto a un subconjunto de datos de entrenamiento, en el que el modelo no es capaz de generalizar correctamente las características de los datos y solo es capaz de reconocer las entradas utilizadas en el entrenamiento. Los modelos con sobreajuste son capaces de ofrecer excelentes resultados en el conjunto de datos de entrenamiento frente a resultados pobres cuando se emplean sobre datos que nunca han sido utilizados en el entrenamiento.

El sobreajuste se detecta al comparar métricas de rendimiento y obtener valores elevados sobre el subconjunto de entrenamiento, pero pésimos sobre el subconjunto de prueba (siempre que no compartan valores ambos subconjuntos).

Técnicas de detección de sobreajuste: validación cruzada

La validación cruzada es un proceso de entrenamiento y prueba en la que el entrenamiento y la prueba del modelo se realiza en múltiples iteraciones, y en cada iteración los conjuntos de datos de entrenamiento y prueba varían con el objetivo de demostrar que el modelo, independientemente del subconjunto de datos que haya sido seleccionado, es capaz de clasificar correctamente. El número de iteraciones que se realizan en un proceso de validación cruzada se denominan capas o pliegues *(folds)*.

Este tipo de validación permite:

- Evaluar el modelo de forma precisa, pues al obtener medias promedio entre múltiples iteraciones se puede ver de forma más clara si una métrica es variable o constante.
- Observar un posible sobreajuste del modelo, ya que, al evaluar el rendimiento del modelo en múltiples subconjuntos de datos, se puede detectar si el modelo se desempeña bien en todos los posibles casos.

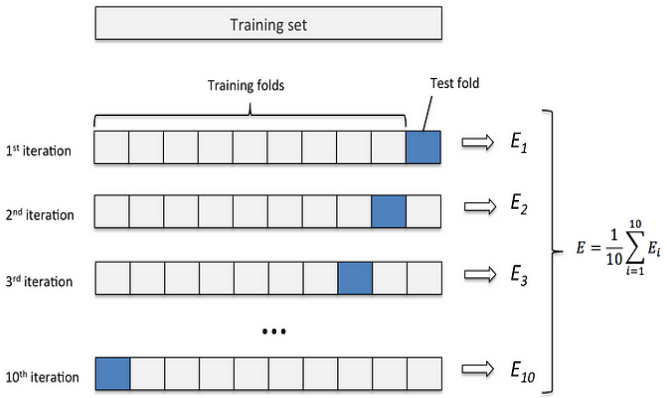

Ilustración 9. Representación gráfica de validación cruzada [21].

Desbalance de datos

El desbalance de datos en el aprendizaje automático se refiere a una situación en la que la distribución de las clases objetivo está significativamente sesgada, lo que dificulta el entrenamiento efectivo del modelo y puede llevar a predicciones sesgadas o inexactas.

Una posible solución al desbalance de datos es la aplicación de técnicas de balanceo, que permiten aumentar o disminuir el número de datos de cada clase con el objetivo de que se distribuyan de forma equilibrada.

Técnicas de resolución de desbalanceo: SMOTE

En el contexto de los modelos de aprendizaje automático se suele buscar conjuntos de datos cuyas clases se encuentren balanceadas entre sí, pues un conjunto de datos desbalanceado puede derivar en un modelo mal entrenado que no sea capaz de reconocer las características de una determinada clase, y por ende funcionar incorrectamente.

Aquellas clases cuyo número de elementos sea muy inferior al resto de muestras son conocidas como **clases minoritarias**, mientras que las clases cuyo número de elementos sobrepasa con creces al resto de clases son denominadas **clases mayoritarias**.

Para solventar el problema del desbalanceo existen dos tipos de algoritmos frecuentemente utilizados:

- **Algoritmos de infra-muestreo:** reducen el número de elementos de las clases mayoritarias para que se asemejen en número a las minoritarias.
- **Algoritmos de sobre-muestreo:** aumentan de forma artificial el número de elementos de las clases minoritarias para que se asemejen en número a las mayoritarias.

Uno de los algoritmos más conocidos de sobre-muestreo es el algoritmo SMOTE [22] *(Synthetic Minority Oversampling Technique)*, que mediante interpolaciones entre los vectores de las clases minoritarias genera nuevos vectores "sintéticos" sin repetición, por lo que no existirán vectores repetidos en los conjuntos de vectores utilizados.

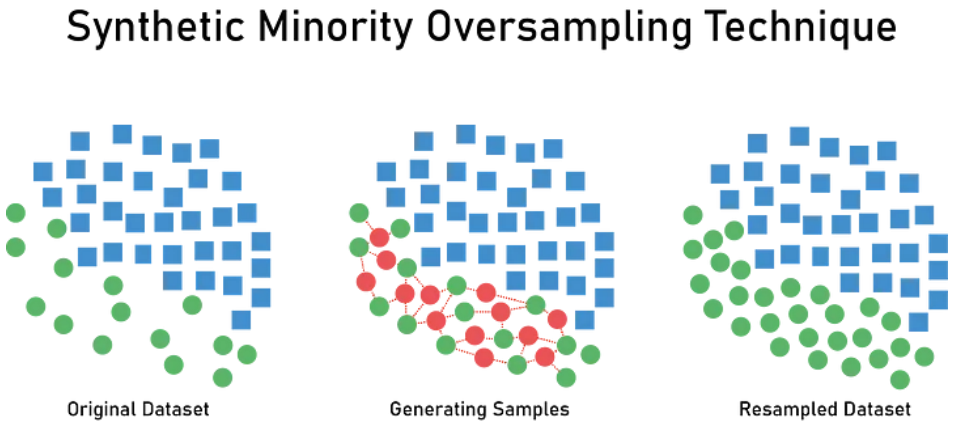

Ilustración 10. Ejemplo gráfico de SMOTE [23] en funcionamiento.

2.4 Técnicas de Procesamiento de Lenguaje Natural (PLN)

Para poder integrar las técnicas de aprendizaje automático sobre cadenas de texto (de naturaleza variable, flexible y no numérica) hace falta realizar diferentes tareas del área del Procesamiento de Lenguaje Natural. Esto se emplea con el objetivo de convertir las cadenas de texto a utilizar (direcciones en el caso de este proyecto) en valores compatibles para los modelos de clasificación. Las tareas por realizar son las siguientes:

- Tokenizado de direcciones
- Representación de textos

2.4.1 Tokenizado

La primera tarea de procesamiento de lenguaje que se realiza en los problemas de aprendizaje automático centrados en datos textuales, como [2] o [6], es el tokenizado de sentencias. Los modelos de clasificación no reciben cadenas de texto en bruto para ser clasificadas, sino que reciben en su lugar vectores numéricos denominados "vectores de características" que representan a estas cadenas de texto. Para poder generar estas direcciones, muchos algoritmos requieren que las sentencias estén divididas en palabras individuales: este proceso de división de sentencias en palabras es conocido como tokenizado [24].

2.4.2 Representación de textos

Una vez tokenizadas las sentencias, los vectores de palabras deben pasar por algún proceso que los convierta en vectores de características válidos. Esta conversión la realizan los denominados "algoritmos de representación de texto", un tipo de algoritmos que suelen extraer la representación vectorial de una palabra respecto a un corpus utilizando criterios como su frecuencia dentro del mismo corpus o su frecuencia dentro de un determinado documento respecto al resto de documentos, entre otros [24]. Algunos de los algoritmos de representación de texto más utilizados son:

- *Bag of N-Grams*
- *TF-IDF*
- *Doc2vec*
- *FastText*
- *BERT*

Bag of N-grams

El algoritmo de bolsa de N-gramas o *"Bag of n-grams"* parte de la idea de que una sentencia de N palabras se puede descomponer en secuencias de palabras de longitud $[1, N]$. De esta forma, dada la sentencia de ejemplo "CALLE AVE VERDE 18", se pueden obtener los siguientes fragmentos:

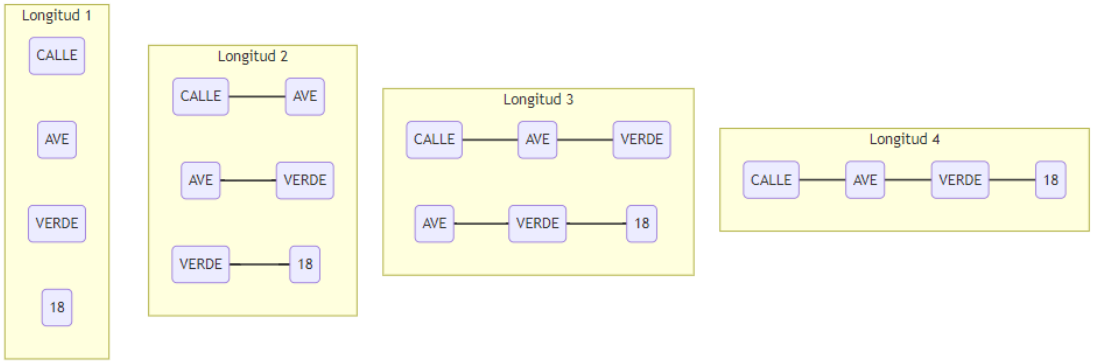

Ilustración 11. Representación gráfica de N-gramas.

Los fragmentos resultantes de dividir una sentencia se llaman n-gramas, y la *N* en su nombre define el número de palabras que contiene cada fragmento en específico. Asimismo, en el ejemplo anterior se observa de izquierda a derecha: unigramas, bigramas, trigramas y tetragramas.

El algoritmo de Bolsa de N-gramas se basa en calcular para cada sentencia la frecuencia de sus n-gramas dentro de la misma [25]. De esta forma se obtiene una representación única para cada sentencia. Cabe recalcar que cuando los n-gramas tienen un tamaño mayor a 2 (bigramas, trigramas, etc.), los n-gramas de dos sentencias con las mismas palabras en diferente orden resultarán en vectores diferentes. Este comportamiento se denomina "sensibilidad al orden".

El algoritmo de Bolsa de N-gramas permite limitar su funcionamiento a palabras con un mínimo de frecuencia en el corpus, para evitar vectores de representación de dimensionalidad extremadamente elevada (mayor a 500 dimensiones).

TF-IDF

La representación mediante *TF-IDF* se basa en calcular la frecuencia de aparición de una palabra en un determinado documento respecto a su frecuencia en el resto de los documentos [26]. Se tienen en cuenta los siguientes axiomas:

- Si una palabra aparece en un documento, cuantas más veces aparezca en él más relevante será.
- Si una misma palabra aparece en múltiples documentos, significa que no es relevante pues se repite y no aporta información que discrimine. Por tanto, cuanto más aparezca en diferentes documentos, menos importante será.

De esta forma, el valor *TF-IDF* de una palabra dada se calcula de la siguiente forma:

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

Donde:

- *t* es la palabra en cuestión
- *d* es el documento en el que se encuentra la palabra
- *D* es el conjunto de documentos (corpus)

En esta fórmula *tf(t, d)* es la frecuencia de aparición de la palabra *t* en el documento *d*, e *idf(t, D)* es el inverso de la frecuencia de aparición de la palabra *t* en el conjunto de documentos *D*.

Cabe resaltar que, si bien la definición original del algoritmo contempla únicamente el conteo de palabras, este algoritmo puede ser aplicado también a n-gramas, llevando el recuento de un determinado n-grama dentro de un documento para un determinado corpus, permitiendo así dar sensibilidad al orden entre palabras.

El algoritmo *TF-IDF* permite limitar su funcionamiento a palabras con un mínimo de frecuencia en el corpus, para evitar vectores de representación de dimensionalidad extremadamente elevada, pudiendo alcanzar el número de palabras diferentes en el corpus.

Doc2vec

El algoritmo *doc2vec* permite obtener representaciones vectoriales de dimensionalidad moderada (escogida por el usuario) y semejanza semántica alta. Se trata de una derivación de la red neuronal *word2vec* [7], que en resumidas cuentas predice el valor semántico de una palabra con relación al contexto de palabras que la rodean [8].

Los algoritmos antes mencionados cumplen su cometido de asignar a cada palabra (o sentencia) un vector numérico único, de tal forma que dos sentencias cualesquiera diferentes no lleguen a compartir un mismo vector. El problema de estos algoritmos, sin embargo, es que no son capaces de reflejar la semejanza semántica entre sentencias de forma medible.

Supongamos que se busca representar vectorialmente las palabras “CARRETERA”, “CALLE”, “PLAZA”, y “AVELLANA”. Se entiende que “CARRETERA” y “CALLE” son palabras similares (ambas representan un tipo de vía), por lo que independientemente de qué algoritmo se siga para asignarles un vector, ambos deberían tener una distancia entre ellos menor que la que resultaría entre “CARRETERA” y “PLAZA” o “CALLE” y “AVELLANA”.

Esta cercanía entre vectores puede beneficiar a la hora de clasificar, pues sentencias semánticamente similares pueden ser clasificadas como la misma por los clasificadores al estar próximas entre ellas.

FastText

El algoritmo *FastText* se basa en una red neuronal de predicción semántica acorde al contexto de una palabra, de forma similar a *doc2vec* y *word2vec*. La diferencia con los algoritmos anteriores es el concepto de token de cada algoritmo: *word2vec* y *doc2vec* entrenan sus redes neuronales internas con palabras y sentencias respectivamente [9]. *FastText*, por otro lado, entrena sus redes neuronales mediante n-gramas de caracteres.

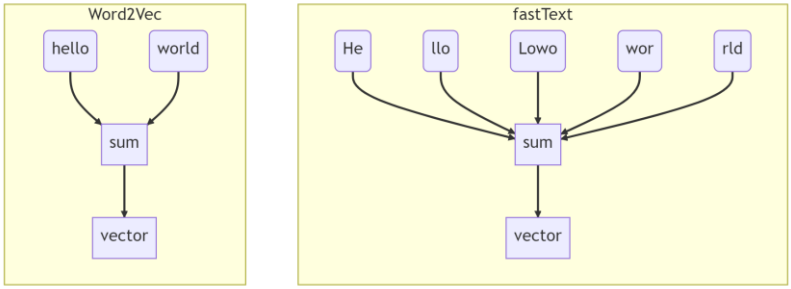

Ilustración 12. Funcionamiento interno de FastText.

Este entrenamiento mediante caracteres permite a las redes neuronales de tipo *FastText*

predecir dada una palabra de inicio sus sinónimos.

BERT

El modelo *BERT* [10] *(Bidirectional Encoder Representation from Transformers)* es un tipo de red neuronal desarrollada por *Google* que se caracteriza por el uso de unos componentes denominados Transformers.

Los Transformers son un tipo de componente del área del aprendizaje automático que permiten:

- Convertir un elemento (Ej. Una sentencia) en un vector numérico mediante sus codificadores *(encoders).*
- Convertir un vector numérico de vuelta a su formato original mediante decodificadores *(decoders).*

El modelo *BERT* apila múltiples codificadores de forma secuencial y luego los pre-entrena con múltiples parejas de datos. Un beneficio de *BERT* frente al resto de métodos de representación de texto es que puede ser ajustado *(fine tuning)* a diferentes ámbitos manteniendo el rendimiento. Esto permitiría, por ejemplo, que un modelo entrenado con sentencias en español sea capaz de reconocer sentencias inglesas o francesas tras ser ajustado.

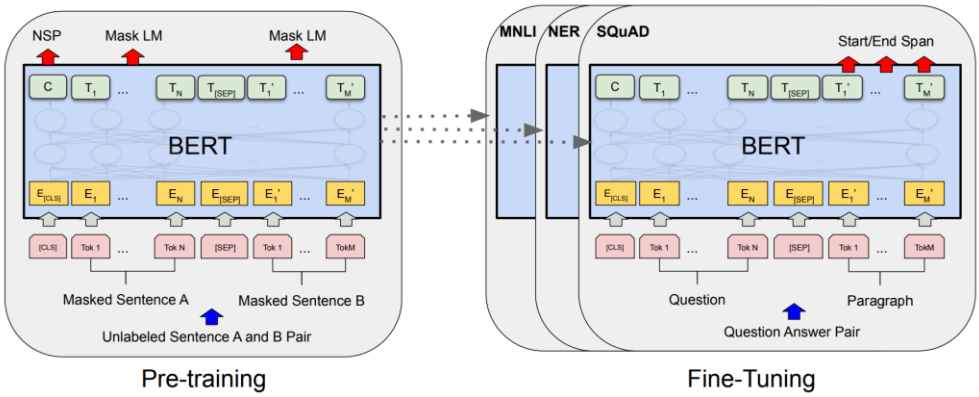

Ilustración 13. Diagrama de componentes de modelo BERT [27].

Capítulo 3. Fases del proyecto

El flujo de trabajo total del proyecto está descrito en el siguiente gráfico:

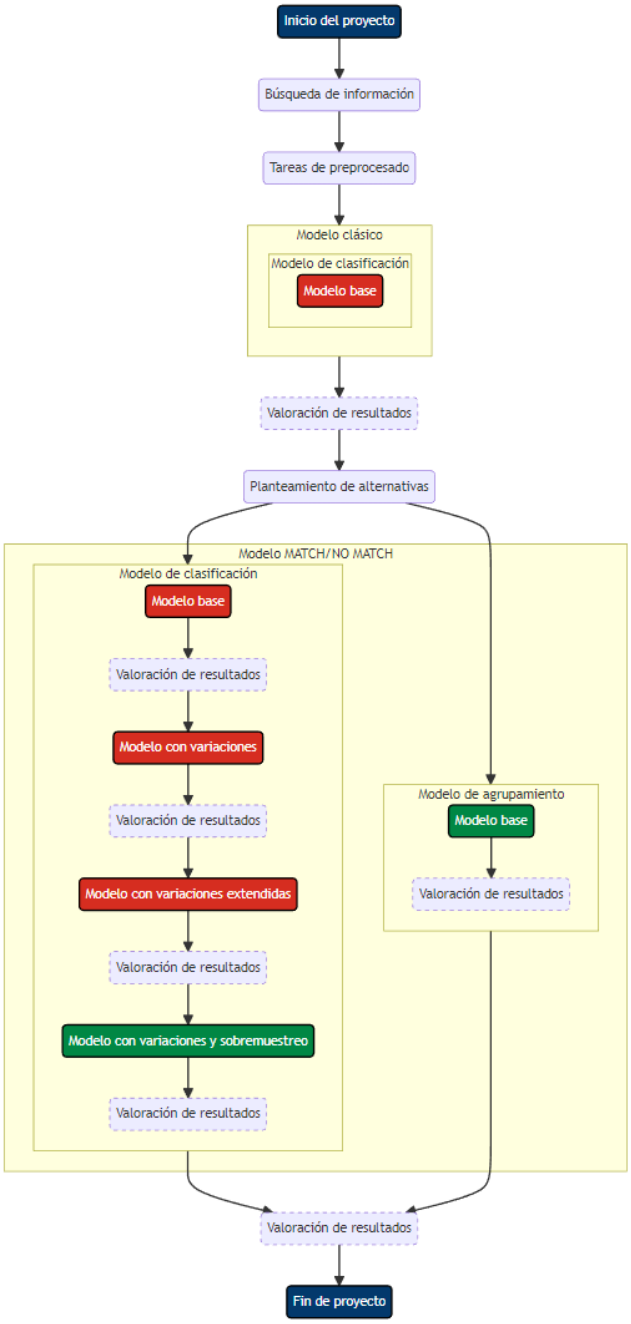

Ilustración 14. Diagrama de flujo del proyecto.

Como se puede apreciar en el gráfico anterior, se ha comenzado el desarrollo del proyecto con una búsqueda de la información necesaria para entender el problema, los componentes esenciales del PLN, los diferentes tipos de algoritmos disponibles, etc. En esta fase también se incluyen la lectura de documentación de las diferentes librerías utilizadas en el proyecto.

A continuación, se implementaron las tareas de preprocesamiento de datos en las que se corrigieron posibles errores en el conjunto de datos original, además de realizar diferentes transformaciones de formato de los datos originales.

En la siguiente fase se realizó una primera clasificación de direcciones primitiva, en la que el modelo se alimentaba con los millones de direcciones del conjunto de datos. Viendo los resultados tan negativos del experimento, se valoraron diferentes modelos alternativos que pudieran dar mejores resultados.

Los modelos alternativos, como se verá más adelante, entran dentro del paradigma "MATCH/NO-MATCH". Dentro de este paradigma se desarrollaron dos tipos de modelos de coincidencia de direcciones:

- **Modelo de clasificación:** partió de mejores resultados que los obtenidos de la fase de clasificación clásica, pero requirió de múltiples fases de mejora hasta dar resultados válidos.
- **Modelo de agrupamiento:** modelo con resultados válidos desde un inicio.

Una vez obtenidos los resultados computacionales de ambos modelos se entró en la fase de "Valoración de resultados", en la que se analizaron las métricas para establecer qué modelos se considerarán válidos.

Capítulo 4. Preprocesado de datos

4.1 Descripción de los datos

Los datos para trabajar, consiste en la base de conocimiento formada por el registro de portales donde se almacenan todas las direcciones o portales georreferenciados cedida por el ISTAC con fines de desarrollar la investigación. Este conjunto de datos contiene un total de 1.784.217 registros georreferenciados de toda Canarias, y la información de cada dirección se ve desglosada en múltiples campos: desde la API utilizada para geolocalizar una dirección en concreto hasta la vigencia de la propia API al momento de realizar el estudio, pasando por el tipo de vía de la dirección o su número de calle (en caso de tenerla). Los más relevantes a tener en cuenta en este proyecto son los recogidos en la Tabla 1:

Nombre de campo	**Descripción de campo**
uuid_idt	identificador exclusivo de una dirección
tvia	Tipo de vía
nvia	Nombre de vía
numer	Número de vía de la dirección (si lo hubiese)
codmun	Código de municipio
nommun	Nombre del municipio
direccion	Dirección completa (concatenación de los campos anteriores)

Tabla 1. Campos relevantes del conjunto de datos.

En este conjunto de datos, una misma dirección puede aparecer múltiples veces con diferentes variaciones (Tabla 2), ya que puede escribirse de múltiples formas. A todas estas "variaciones" o "direcciones variadas" les corresponde el mismo identificador, pues, aunque parezcan diferentes direcciones, el ISTAC ha determinado con sus algoritmos que hacen referencia al mismo lugar físico [1].

"CALLE SECORITA MARIA MANRIQUE LARA 2 AGAETE"
"CALLE SRTA M MANRIQUE LARA 2 AGAETE"
"CALLE MARIA MANRIQUE LARA 0 AGAETE"
"CALLE MARIA MANRIQUE LARA 2 AGAETE"
"CALLE ST M MANRIQUE LARA 2 AGAETE"
"CALLE MANRIQUE LARA 2 AGAETE"

Tabla 2. Direcciones para un "uuid_idt" dado.

No todos los ID en el conjunto de datos original tienen asociados el mismo número de entradas. La distribución de entradas por ID se describe en la siguiente tabla:

Número de direcciones asociadas	**Frecuencia**	**Porcentaje**
1	264.294	47,17%
2	114.643	20,46%
3	60.624	10,82%
4	35.483	6,33%
5	22.137	3,95%
6	14.758	2,63%
7	10.127	1,81%
8	7.390	1,32%
9	5.385	0,96%
10 o más	25.500	4,55%

Tabla 3. Distribución de entradas por ID en conjunto de datos completo.

Se puede observar que más del 50% de los IDs (67,63%) tienen asociadas a lo sumo dos direcciones, ascendiendo hasta 88,73% los IDs con 5 entradas.

Antes de poder realizar las tareas de análisis de datos, se debe preprocesar la información disponible para eliminar o corregir posibles valores erróneos, realizar transformaciones a los datos y diversos pasos previos.

4.2 Limpieza del conjunto de datos

Las tareas de limpieza de datos son las encargadas de corregir todos aquellos campos del conjunto original con valores erróneos, incompletos o directamente inexistentes. Esto servirá para

obtener representaciones vectoriales más detalladas y correctas de las direcciones en cuestión.

Pese a que el ISTAC ha realizado múltiples correcciones previas, comentadas en su informe original, hay ciertos valores irregulares que se deben corregir, así como valores nulos a rellenar para su uso en la clasificación. Algunos de estos campos a corregir son:

- Valores nulos para ser asignados o rellenados.
- Entradas duplicadas que deben ser eliminadas.

4.2.1 Detección de valores nulos

Ciertos campos relevantes contienen valores nulos que pueden dificultar la clasificación, pues son campos que no están aportando información a la dirección.

uuid_idt	0
tvia	130360
nvia	10219
numer	0
codmun	0
nommun	0
direccion	0

Tabla 4. Valores nulos originales del conjunto de datos.

Como se aprecia en la tabla anterior, existen aproximadamente 130.000 tipos de vía y 10.000 números de vía nulos. Estos valores nulos incluyen tanto zonas que el proceso del ISTAC no ha reconocido (Ej. **tvia** en “URBANIZACION CANDELARIAS E 13 0 AGAETE”) como zonas donde no se ha especificado el tipo de vía de forma reconocible o directamente no se ha reconocido (Ej. “28 AGAETE”).

Para eliminar estos valores nulos se asignarán valores por defecto a las vías nulas, pues las técnicas de representación de textos suelen dar mejores resultados con campos no nulos. En otros campos los valores nulos se han denotado con la cadena de texto “_U”, así que por mantener la coherencia este será el valor nulo por defecto del conjunto de datos. Realizando esta operación de rellenado, se han conseguido eliminar por completo los registros nulos.

uuid_idt	0
tvia	0
nvia	0
numer	0
codmun	0
nommun	0
direccion	0

Tabla 5. Valores nulos tras reducciones en conjunto de datos.

4.2.2 Reducción de valores nulos

Habiendo rellenado los valores nulos, el objetivo actual es reducir su número en la medida de lo posible pues un valor nulo, aunque valga "_U" por defecto, no aporta demasiada información a un modelo de clasificación (Ej. el tipo de vía "_U" no sirve para identificar una calle, una avenida o una urbanización).

El primer campo por corregir será **tvia**, pues es el más sencillo de procesar por ser (normalmente) el primero que aparece en una dirección.

tvia	**direccion**
CALLE	CALLE FUENTE SANTA 5 AGAETE
CALLE	CALLE CRUZ CHIQUITA 2 AGAETE
URBANIZACIÓN	URBANIZACION RESIDENCIAL PALMERAL 9 AGAETE
CALLE	CALLE SECORITA MARIA MANRIQUE LARA 2 AGAETE

Tabla 6. Campo "tvia" respecto a "direccion".

Como primera aproximación al problema, se han buscado aquellas direcciones duplicadas en el conjunto de datos que contengan un campo de tvia nulo y eliminarlas, pues no aportan ningún tipo de información relevante.

```
{"tvia": "_U",    "codmun": 38741, "nommun": "ARICO", ...} # ❌ Eliminamos esta
{"tvia": "calle", "codmun": 38741, "nommun": "ARICO", ...} # ✅ Dejamos esta
```

Ilustración 15. Ejemplo de eliminación de entradas nulas.

Los resultados de aplicar esta transformación son los siguientes:

Frecuencia de tvia desconocido	
Antes de transformación	130.360
Después de transformación	122.763

Tabla 7. Resultados de reducción de nulos por eliminación.

En la tabla anterior se observa una eliminación de 7.597 registros con valores nulos, lo que representa un 0,42% del total de entradas del conjunto de datos. Se trata de una mejora respecto al caso anterior, pero una mejora marginal.

4.2.3 Asignación de valores nulos

Buscando mejorar esta reducción ínfima, se ha optado por una estrategia alternativa: la mejora que se realizó a este proceso ha sido comprobar para cada entrada del conjunto de datos con **tvia** nulo si este campo se encontraba originalmente en el campo **direccion** pero no fue seleccionado por el proceso de limpieza del ISTAC, como es el caso de la entrada con campo **direccion** "RESIDENCIAL CANDELARIA 18 AGAETE", que se observa claramente que contiene el tipo de vía al comienzo (**tvia** = "RESIDENCIAL") pero el mencionado proceso no pudo reconocer.

"RESIDENCIAL CANDELARIA 18 AGAETE"
"HOYO 9 ALTO AGAETE"
"URBANIZACIÓN PALMERAL MANZANA H B1 AGAETE"
"CALLE OBISPO PILDAIN BL 3 AGAETE"
"CAMINO CRUZ 137 AGAETE"
"CRUZ CHIQUITA 7 AGAETE"
"CHAPIN 5 AGAETE"
"URBANIZACIÓN CANDELARIAS E25 AGAETE"

*Tabla 8. Ejemplo de entradas con **tvia** nulo.*

Para solventar esto se ha propuesto generar un conjunto con los tipos de vía más comunes y comprobar para cada dirección con **tvia** nulo si contiene alguno de estos tipos de vía recurrentes al comienzo: en caso afirmativo, a la entrada del conjunto de datos se le asignará el **tvia** detectado dentro del campo **direccion**.

Realizando esta operación de modificación de registros, se han obtenido los siguientes resultados de campos nulos:

Frecuencia de tvia desconocido	
Antes de transformación	122.763
Después de transformación	98.720

*Tabla 9. Frecuencia de entradas con **tvia** desconocido.*

Como se observa en la tabla anterior, se han convertido 24.043 registros con **tvia** originalmente nulos en registros completos, lo que implica casi un 20% (19,17%) del número de entradas con **tvia** nulo tras la operación de limpieza anterior.

4.3 Formateo de la información

4.3.1 Muestra de los datos

El conjunto de datos a utilizar es demasiado extenso como para poder ser usado para un estudio de datos ágil, pues los requerimientos computacionales de memoria y tiempo (especialmente este último) que se requieren para entrenar los modelos de aprendizaje son muy altos. Para remediar esto y poder obtener resultados representativos de forma breve, se ha decidido tomar una muestra reducida del total de datos.

Para que la muestra sea lo más representativa posible, se seleccionará respecto al campo menos restrictivo: el nombre de municipio. De esta forma no habrá ningún municipio sin estudiar. Para agilizar los tiempos sin perder volumen de información, se ha tomado como umbral 1.000 filas por municipio, pues sabemos que los municipios del conjunto de datos suelen tener más de 1.000 filas.

1.000 filas por municipio producen una muestra de 92.000 entradas, una reducción del 94,92% respecto al tamaño original del conjunto de datos. En [6] se toma como muestra únicamente las entradas del municipio de **Santa Brígida**: 16.024 direcciones.

Con esta reducción de datos se ha pasado de 1.776.620 de registros (tras la limpieza de datos) a 92.000, una reducción del 94,82%. Estos 1.776.620 se traducen en 67.795 IDs únicos.

4.3.2 Tareas de tokenizado

La primera tarea para realizar con el conjunto datos limpio es tokenizar las direcciones para procesarlas más adelante.

Para realizar esta tarea se han planteado utilizar diferentes librerías con herramientas de tokenizado, buscando la que cumpla la tarea con menores costos. Se han seleccionado 3 posibles librerías:

- *Keras* [28]: librería de aprendizaje automático derivada de *TensorFlow*, con funcionalidades de procesamiento de textos, imágenes o vídeos.
- *NLTK* [29]: librería de PLN apta para múltiples idiomas, con funcionalidades de procesado y reconocimiento de textos.
- *Preln* [30]: librería de PLN enfocada para el reconocimiento de textos en español. Posee herramientas de tokenizado con detección de errores incluida.

Pese a que todas las librerías comentadas son capaces de tokenizar textos, no todas lo

hacen de la misma manera, requiriendo diferentes tiempos de ejecución o resultando en vectores de tokens diferentes. La implementación de tokenizador de *Keras*, por ejemplo, recibe como entrada una cadena de texto y produce un vector de valores numéricos en vez de un vector de palabras, por lo que se descartó su implementación.

Como alternativas a *Keras*, se dispone de *NLTK* y *Preln*. Ambas funcionan de forma similar: Reciben en cada llamada una cadena de texto y producen un vector de palabras como deseamos. Sabiendo que tienen un funcionamiento similar y que ambos soportan tokenizado de sentencias en español, se ha decidido comparar los tiempos de ejecución de cada tokenizador para apreciar las diferencias:

Algoritmo	**Original (s)**	**Muestra (s)**
NLTK	81	3
Preln	100	3

Tabla 10. Diferencias entre tiempos de ejecución de tokenizadores.

En la tabla anterior se puede apreciar que *NLTK* consigue unos tiempos de ejecución más bajos que *Preln* en condiciones de grandes volúmenes de datos, mientras que en condiciones controladas como puedan ser las muestras de ejecución sus tiempos son similares. En base a estos resultados se ha decidido escoger *NLTK* como herramienta de tokenizado, pues sus costos computacionales son más favorables que los de *Preln*.

4.3.3 Representación gráfica de la muestra

Para facilitar la comprensión de los datos con los que se está trabajando, se ha realizado un recuento de palabras, partiendo del resultado obtenido de la tokenización. Se han contado las apariciones de cada una de las posibles palabras y se ha obtenido la siguiente nube de palabras.

Ilustración 16. Nube de palabras de la muestra tomada.

Se puede identificar que en nuestra muestra abundan las palabras de tipo “Calle”, “Avenida”, “Carretera” o “Camino”: tipos de vía muy comunes. También podemos ver nombres de lugares y pueblos como puedan ser “Palma”, “Acentejo”, “Cruz” o “Gomera”, pues son nombres y prefijos muy utilizados en las calles canarias.

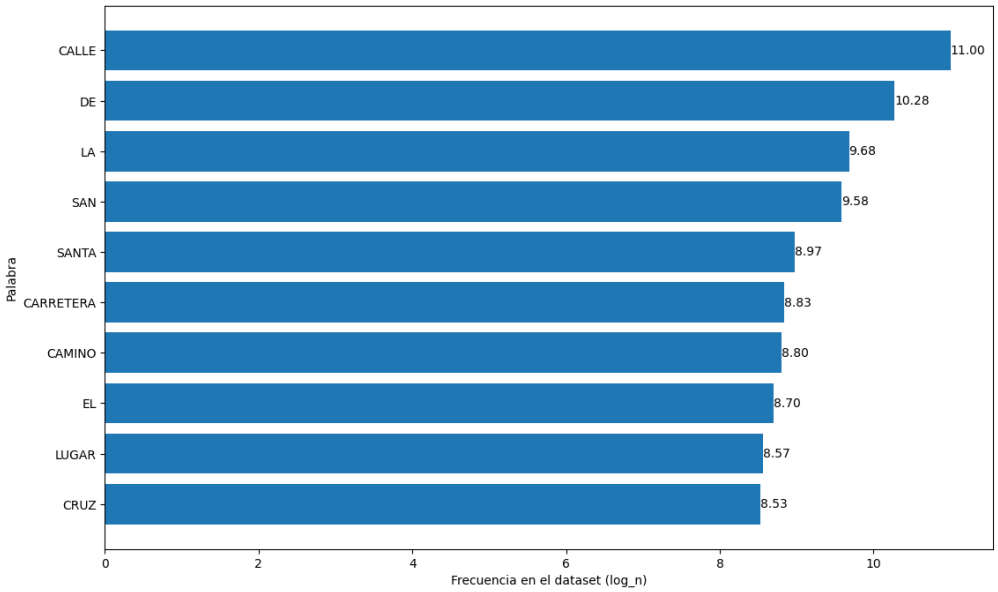

Ilustración 17. Diagrama de palabras más frecuentes de la muestra.

Al centrarse en la frecuencia de palabras en forma de gráfico de barras, se observa que la palabra más frecuente suele ser "CALLE", seguida de "DE" y de tipos de vía como "carretera".

4.3.4 Tareas de representación de texto

El siguiente paso por aplicar será tomar las sentencias tokenizadas y alimentar con ellas diferentes algoritmos de representación de textos con el objetivo de obtener las representaciones vectoriales de las sentencias. Tras esto se seleccionará para el resto del proyecto una única representación vectorial, por lo que compararemos resultados y limitaciones de cada algoritmo.

Para esta comparación se han usado los siguientes algoritmos de representación de textos:

- *TF-IDF*
- *Bag of N-grams*
- *Doc2vec*

Los resultados obtenidos para cada algoritmo con el conjunto de datos de muestra son los siguientes:

Algoritmo	**Tiempo de procesamiento (s)**	**Dimensionalidad**
TF-IDF	0,7s	5.408
Bolsa de n-gramas	0,7s	5.408
doc2vec	18,1s	100

Tabla 11. Resultados computacionales de algoritmos de representación de texto.

En la tabla anterior se puede apreciar que los algoritmos basados en N-gramas (*TF-IDF* y Bolsa de n-gramas) tienen unos tiempos de procesamiento mucho más bajos que los del algoritmo *doc2vec*, concretamente 25 veces más rápidos.

Además de esto hay que sumar el hecho de que *doc2vec* aporta vectores de sentencias cuya cercanía semántica se traducirá en cercanía geométrica, lo que puede servir en el entrenamiento de los modelos de clasificación para obtener resultados más certeros. En [6] se realizó esta misma comparativa de algoritmos de representación, reforzando el buen funcionamiento de *doc2vec* frente al resto de modelos.

Por estos motivos, se ha decidido utilizar *doc2vec* como principal algoritmo de representación de textos.

Capítulo 5. Modelos de clasificación

5.1 Definiendo el problema de clasificación

Las diferentes versiones de una misma dirección (comentadas anteriormente en el capítulo 4.1) en la base de datos definen una clase o grupo. Por tanto, todas las direcciones de una clase están registradas bajo el mismo valor de **uuid_idt** (ID). El problema por resolver consiste en un problema de clasificación multiclase, en el que la etiqueta de cada clase está asignada en el campo **uuid_idt**.

Por razones de claridad e interpretación se realiza un proceso de asignación de etiquetas más simples (valores numéricos en vez de cadenas de texto) como identificadores de la clase o grupo.

5.2 Fase de clasificación ordinaria

5.2.1 Descripción del modelo

Antes de implementar los modelos de coincidencia binarios (MATCH/NO MATCH) vistos en *Postmatch* [2] y en [6], se realizará una clasificación de direcciones "en crudo" con el objetivo de valorar el rendimiento de los modelos de aprendizaje automático con conjuntos de datos de direcciones. De esta forma, se podrá comparar más adelante los resultados del modelo de coincidencia binario frente al modelo de coincidencia actual.

Para esta "clasificación en crudo" se busca asociar a cada ID de dirección un identificador numérico procesable por un modelo de aprendizaje automático, en vez de la cadena de texto que originalmente era.

ID original	**ID numérico asignado**
"ACEA79F2-7B7A-11EB-896E-6F4967C1D9D4"	0
"6B4132F0-3251-11E8-BF79-480FCF5217B3"	1
"8D631995-79D7-11EB-A554-CB98385CFF1E"	2

Tabla 12. Ejemplo de etiquetado de direcciones.

Una vez realizada esta "creación de etiquetas", se alimentará al modelo de clasificación con el conjunto de datos etiquetado con el objetivo de valorar sus resultados. Este modelo será definido como "modelo genérico" o "modelo ordinario".

5.2.2 Creación del modelo

Para la creación de este modelo de clasificación genérico se han implementado dos algoritmos de clasificación diferentes, con el objetivo de comparar el rendimiento en múltiples aplicaciones:

- Algoritmo *Random Forest*
- Algoritmo Gaussiano *Naive Bayes*

En este modelo ordinario no se ha aplicado el algoritmo *XGBoost*, pues como se comentó en el capítulo A, el algoritmo requiere de entradas de entrenamiento cuyas clases se encuentren perfectamente definidas en un rango $[0, N)$. Sabiendo que muchos IDs solo cuentan con una dirección asociada, existirán muchas clases que aparezcan únicamente en el subconjunto de pruebas, sin ningún valor en el subconjunto de entrenamiento.

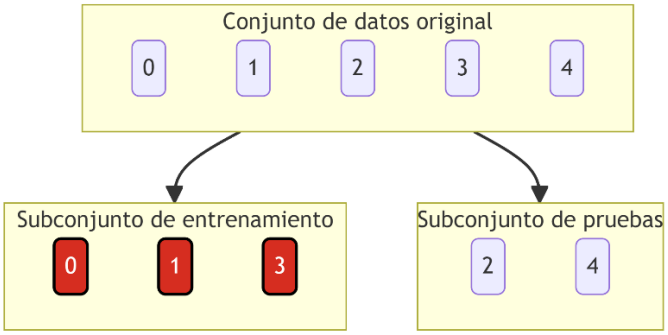

Ilustración 18. Problemática de XGBoost en modelo de clasificación ordinario.

Existiendo tantas etiquetas en el conjunto de datos, los valores de precisión y sensibilidad se calcularán como la media entre todos los valores de precisión y sensibilidad resultantes del modelo. La matriz de confusión, sin embargo, se omitirá en este modelo por exceso de etiquetas. Los resultados son los siguientes:

Fase de creación	**Tiempos de ejecución (s)**	
	Random Forest	***Naive Bayes* Gaussiano**
Creación del conjunto de datos	0,7	0,7
Entrenamiento del modelo	190,98	9,73
Predicción del modelo	185,36	45.600

Tabla 13. Tiempos de ejecución de modelo de clasificación genérico.

Algoritmo	**Exactitud (%)**	**Precisión (%)**	**Sensibilidad (%)**
Random Forest Classifier	0,024	0,047	0,071
Gaussian NB	0,015	0,056	0,069

Tabla 14. Métricas de rendimiento de modelo de clasificación genérico.

Como se puede apreciar en las tablas anteriores, los resultados tanto a nivel de tiempos

de ejecución como a nivel de métricas de rendimiento son extremadamente negativos.

Comenzando por los tiempos de ejecución se puede observar que los tiempos derivados de la creación de conjuntos no son excesivamente elevados. Respecto a las medidas del algoritmo *Random Forest* se observa que requiere de mucho tiempo de cómputo para ser entrenado, sobrepasando los 3 minutos de entrenamiento. En contraposición a este algoritmo, el *Naive Bayes* Gaussiano demuestra unos tiempos de entrenamiento record de 9,73s.

El primer gran problema de este modelo aparece al realizar las predicciones, pues el algoritmo *Random Forest* entrenado con un volumen de datos tan elevado requiere de más de 3 minutos para realizar las predicciones de prueba, mientras que el *Naive Bayes* Gaussiano supera las 12 horas para lo mismo.

Analizando las métricas de rendimiento, el modelo de clasificación genérico es incapaz de generalizar las características de ningún ID. Al ver que ninguno de los valores de exactitud alcanza el 0.5%, y que ninguno de los valores de precisión o sensibilidad alcanza siquiera un 0.05%, el modelo funciona de manera muy deficiente.

Este problema puede deberse a la falta de direcciones por ID, pues no todas las direcciones tienen múltiples entradas asociadas. Esto podría provocar que, ante tal volumen de etiquetas diferentes y habiendo tan pocas entradas por etiqueta el modelo no sea capaz de reconocer las características diferenciales de cada una de ellas, provocando los resultados pésimos vistos anteriormente.

Para analizar esto, y sabiendo que los resultados de los dos modelos aplicados son similares, se han extraído las direcciones mejor clasificadas del modelo *Random Forest*, como se observa en el siguiente gráfico.

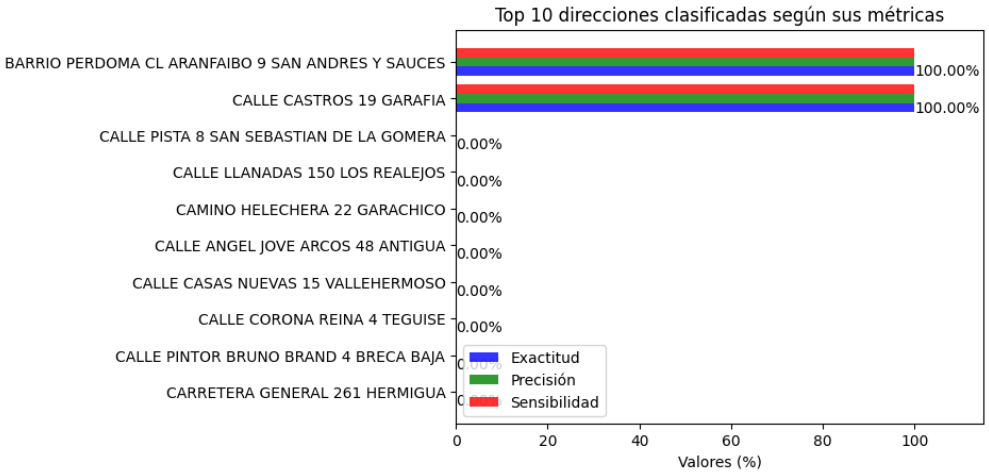

Ilustración 19. Direcciones mejor clasificadas en modelo genérico Random Forest.

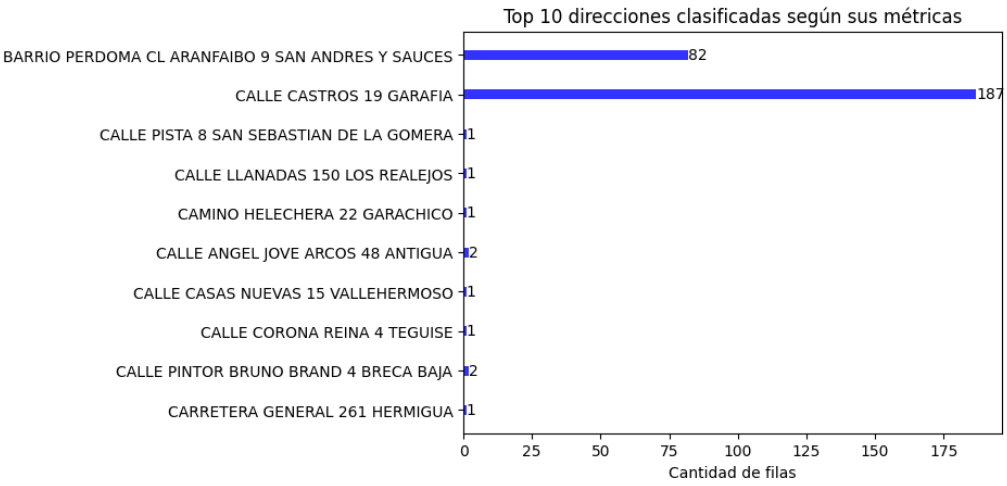

Ilustración 20. Frecuencia de direcciones en modelo genérico Random Forest.

Se observa que las únicas direcciones clasificadas correctamente fueron aquellas con más entradas asociadas en el conjunto de datos, mientras que el resto de las direcciones han sido clasificadas erróneamente por carecer de suficientes filas.

Este problema se ve reflejado en la Tabla 3, donde un importante porcentaje de los IDs del conjunto de datos (84,78% de las direcciones) cuentan únicamente con 1 a 4 direcciones asociadas, lo que puede ser insuficiente para la coincidencia de direcciones. Esto, sumado al elevado número de IDs en el conjunto de datos (67.795), provoca con total seguridad los problemas de rendimiento del modelo.

5.3 Fase de clasificación binaria

Como se observó en la fase anterior, los datos disponibles no son óptimos para una clasificación al uso en la que se alimente al modelo con todas las direcciones del conjunto de datos. En lugar de aplicar el modelo de clasificación genérico, se ha optado por diseñar un modelo de multi-clasificadores binarios inspirado en el artículo de [3]. La estrategia a seguir consiste en generar modelos binarios capaces de clasificar si una dirección pertenece a una determinada clase (MATCH) o no (NO MATCH), uno por cada etiqueta, en vez de contar con un único modelo capaz de clasificar millones de clases diferentes, una para cada ID.

Para aplicar esta técnica, se ha comenzado por dividir el conjunto de datos por municipios, para evitar clasificar las direcciones de un municipio con las de otro, realizar una clasificación selectiva y reducir el tamaño de cada problema de clasificación que se resuelve. Este paso se ha realizado dividiendo las entradas según su campo **nommun**.

El siguiente paso ha sido, para cada ID del conjunto de datos, tomar todas las direcciones de su mismo municipio. Este subconjunto de datos será dividido en entrenamiento *(train)* y prueba *(test)* para alimentar al modelo binario. Dentro de este subconjunto habrá dos tipos de direcciones:

- Direcciones con el mismo ID que se busca clasificar: MATCH o 1.
- Direcciones con diferente ID del que se busca clasificar: NO MATCH o 0.

Para evitar posibles sobreajustes a la hora de entrenar el modelo, y siempre que el subconjunto original lo permita (haya suficientes entradas), no habrá **datos cruzados** entre los

subconjuntos de entrenamiento y prueba.

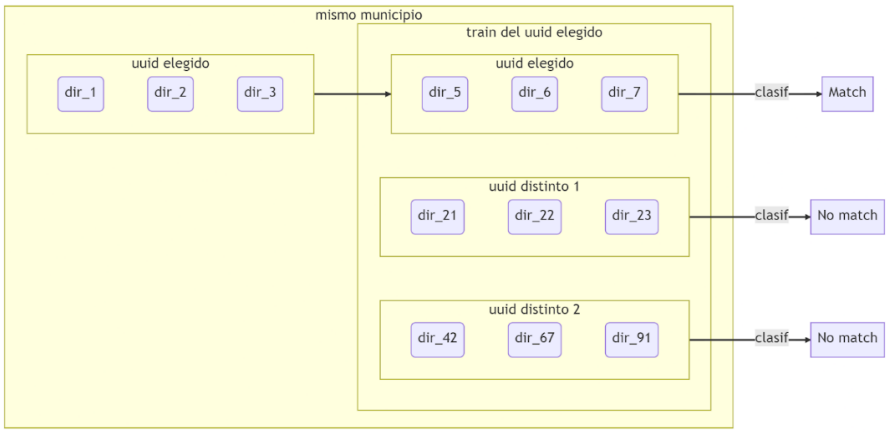

Ilustración 21. Esquema de modelo de clasificación binaria.

Estas clasificaciones se realizarán con los dos algoritmos de clasificación diferentes utilizados en la fase de clasificación ordinaria, con el objetivo de obtener múltiples resultados a comparar. Además, se ha añadido el algoritmo *XGBoost*, pues la limitación de etiquetas vista en la clasificación ordinaria se solventa en este modelo.

El modelo de multi-clasificadores ha sufrido múltiples revisiones a lo largo del proyecto, relacionadas principalmente con el balanceo de los subconjuntos de entrenamiento y testeo de cada ID, que se verá más adelante.

5.3.1 Modelo base

Deficiencias de los datos

Como se ha comentado anteriormente, la estrategia de clasificación es común para todos los modelos, variando solo la estrategia a la hora de escoger y generar los subconjuntos de entrenamiento. En esta primera versión del modelo, la estrategia de selección del subconjunto se basa en tomar para un ID en concreto todas las direcciones del conjunto original de su municipio y dividirlas en MATCH y NO MATCH. Haciendo esto, para un ID cualquiera, la distribución promedio de clases es la siguiente:

Clase	Direcciones	Porcentaje
MATCH	1,47	0,147%
NO MATCH	998,526	99,853%

Tabla 15. Proporción base de clases en subcojunto promedio.

Se puede observar que la proporción de MATCH es extremadamente inferior a la de NO MATCH, lo que indica un desbalance evidente que perjudicará las tareas de entrenamiento de modelos.

Generación del modelo

Aunque se haya observado que el conjunto de datos total cuenta con problemas de desbalance, para esta primera versión del modelo se han generado los subconjuntos individuales de cada ID sin aplicar ningún tipo de balanceo a los datos. La única medida contra el desbalance ha

sido limitar el tamaño de los subconjuntos de cada ID a 20 entradas, de tal forma que el número de entradas MATCH sea más significativo.

Debido al completo desbalanceo de clases, en esta primera versión del modelo no se dispone de suficientes entradas de clase MATCH para dividir el conjunto de datos en entrenamiento y pruebas sin que ambos subconjuntos compartan direcciones. Se deben analizar las métricas de precisión y sensibilidad con precaución pues se puede dar un sobreajuste por falta de direcciones MATCH.

Los tiempos de ejecución obtenidos son los siguientes:

Fase de creación	**Tiempos de ejecución (s)**		
	Random Forest	***Naive Bayes* Gaussiano**	***XGBoost***
Creación del conjunto de datos	518	518	518
Entrenamiento del modelo	121,14	0,71	30,88
Predicción del modelo	8,007	0,22	1,2

Tabla 16. Tiempos de ejecución de modelo de clasificación base.

Las métricas de rendimiento resultantes se han obtenido como la media ponderada de las métricas de cada modelo MATCH/NO MATCH de cada ID usando el subconjunto de datos de prueba *(test)*. Los resultados del modelo base son los siguientes:

Algoritmo	**Exactitud (%)**	**Precisión (%)**	**Sensibilidad (%)**
Random Forest Classifier	99,6650	98,6018	99,5000
Gaussian NB	99,7316	99,4240	99,7634
XGBClassifier	94,4100	6,4789	7,3000

Tabla 17. Métricas de rendimiento de modelo de clasificación base.

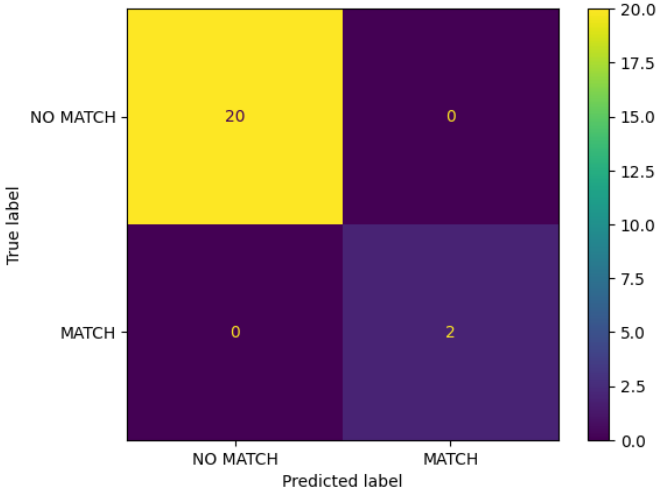

Ilustración 22. Matriz de confusión de modelo de clasificación base.

Los resultados de exactitud del modelo podrían hacer pensar que funciona de manera excelente, pues ninguno de sus 3 valores baja del 94%. Esto mismo confirma la precisión y sensibilidad, superando ambos los valores de 96% (exceptuando los valores anómalos del algoritmo *XGBoost*, asociados probablemente a trabajar con subconjuntos de datos tan reducidos).

Sin embargo, los resultados de la matriz de confusión del problema confirman las sospechas de desbalance en los datos y carencia de direcciones MATCH: en un caso promedio, los clasificadores binarios solo cuentan con 2 direcciones de tipo MATCH para realizar las pruebas. Las limitaciones de los subconjuntos de entrenamiento y prueba indicadas anteriormente revelan que al menos 1 de esas direcciones había sido contenida en el subconjunto de entrenamiento, lo que puede falsear los resultados.

Respecto a los tiempos de ejecución, se puede observar como el modelo *Naive Bayes* Gaussiano completa su entrenamiento en apenas 0,7 s frente a los 30,88 s del modelo *XGBoost* y los 121,14 s del *Random Forest*, colocándose como el modelo más veloz de entrenamiento hasta el momento. Sin embargo, ninguno de los tres modelos funciona de forma adecuada actualmente, por lo que no se podrá destacar ningún modelo de clasificación por sus tiempos de entrenamiento y prueba hasta que la fase de clasificación se realice correctamente.

5.3.2 Clasificación mediante variaciones

Aumento de direcciones

En la segunda versión del modelo de clasificación, la estrategia de generación del conjunto de datos debe remediar en la medida de lo posible la falta de direcciones MATCH vista en el apartado anterior. Una posible solución que se ha propuesto es la generación de variaciones de las direcciones MATCH base para aumentar de forma artificial su número de muestras.

Para establecer un algoritmo de variación de direcciones, se debe analizar qué posibles cambios se pueden aplicar a una dirección para considerarla válida y equivalente.

- Toda variación que altere números o nombres propios, como puedan ser “27” por “72” o “ALMUDENA” POR “CANDELARIA” no se considerará variación válida, pues son cambios que pueden generar direcciones diametralmente opuestas.
- Toda aquella operación que cambie de orden números o nombres propios entre sí no se considerará variación válida, pues al igual que el caso anterior generaría direcciones diametralmente opuestas.

Estas condiciones impiden generar variaciones mediante el nombre del municipio (**nommun**) o del número de la vía (**nummer**), quedando solo como campo variable el tipo de vía (**tvia**). Para generar variaciones sobre este campo, se ha decidido aprovechar la capacidad de los algoritmos de representación de tipo *FastText* [9] para obtener sinónimos de los tipos de vía. Esto se ha realizado utilizando un modelo *FastText* pre-entrenado [31] con millones de documentos en español con el objetivo de que pueda reconocer la mayor cantidad de **tvia** posibles.

Este tipo de variaciones se denominarán “variaciones semánticas”.

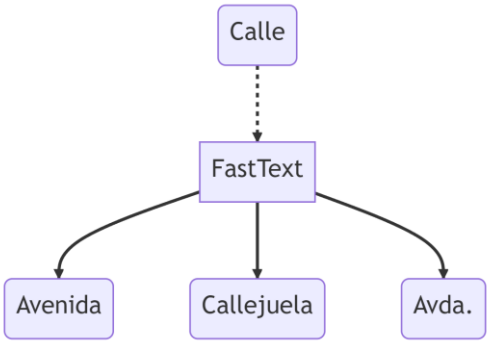

*Ilustración 23. Generación de variaciones según el tipo de vía (**tvia**).*

Para esta versión del modelo se ha escogido un número máximo de tres sinónimos por tipo de vía, pues a partir del 4º el modelo comenzaba a generar sinónimos que no mantenían demasiada relación con la palabra original, como pueda ser el caso de "Calle" con "Rotonda" o "Autopista".

Cabe recalcar que el algoritmo de variaciones de direcciones siempre devuelve, para una dirección X original, las variaciones de dicha dirección junto a la propia X.

Generación del modelo

Teniendo en cuenta el balance de los subconjuntos comentado en el apartado anterior, los modelos de clasificación utilizados en la fase anterior han sido recreados y reentrenados. Se ha mantenido el límite de 20 entradas por subconjunto de entrenamiento aplicado anteriormente para obtener subconjuntos más balanceados que en la fase anterior.

Clase	**Direcciones**	**Porcentaje**
MATCH	4,059	0,4%
NO MATCH	998,559	99,6%

Tabla 18. Proporción promedio de clases en modelo con variaciones.

En esta fase del modelo se repite el mismo problema de la fase anterior: insuficientes entradas de MATCH como para dividir los subconjuntos en entrenamiento y prueba. Como en el modelo de la fase anterior, de manera excepcional existirán direcciones MATCH que coincidan tanto en el subconjunto de entrenamiento como en el de prueba.

La diferencia frente al modelo de la fase previa ha sido un aumento de direcciones de la clase MATCH del 0,253%, alcanzando un 0,4% de direcciones MATCH en esta fase.

Fase de creación	**Tiempos de ejecución (s)**		
	Random Forest	***Naive Bayes* Gaussiano**	***XGBoost***
Creación del conjunto de datos	589	589	589
Entrenamiento del modelo	119,32	0,68	29,86
Predicción del modelo	8,37	0,25	1,23

Tabla 19. Tiempos de ejecución de modelo de clasificación con variaciones.

Las métricas de rendimiento se han obtenido mediante métricas ponderadas respecto al subconjunto de pruebas, al igual que en el apartado anterior. Los resultados del modelo con variaciones son los siguientes:

Algoritmo	**Exactitud (%)**	**Precisión (%)**	**Sensibilidad (%)**
Random Forest Classifier	85,2669	17,1562	8,0996
Gaussian NB	84,2053	5,4357	2,5214
XGBClassifier	84,2499	6,0501	2,9038

Tabla 20. Métricas de rendimiento de modelo de clasificación con variaciones.

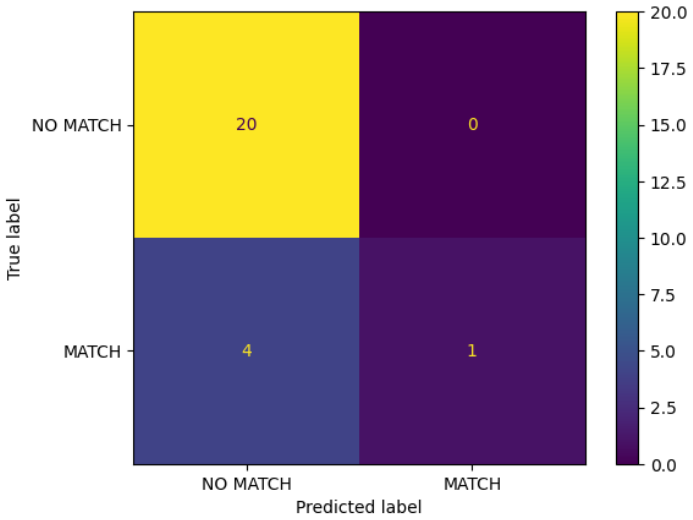

Ilustración 24. Matriz de confusión de modelo de clasificación con variaciones.

Se observa que los resultados de exactitud en todos los algoritmos disminuyen aproximadamente un 14%. Sabiendo que la única modificación realizada respecto a la fase anterior ha sido la inserción de nuevas direcciones MATCH sintéticas, y que el criterio para escoger las NO MATCH no ha sido modificado, la pérdida de exactitud demuestra que el modelo sigue sin clasificar de forma correcta las direcciones MATCH.

Esto parece confirmarse con las pésimas métricas de precisión y sensibilidad, ya que ninguna supera en ninguno de los algoritmos probados el 18%. Esto significa que:

- El modelo clasifica muchas de las escasas direcciones MATCH como no MATCH.
- El modelo no es capaz de reconocer las características de la clase MATCH como para extrapolarlas a otras direcciones de la misma clase.

Por último, la matriz de confusión termina de confirmar que el modelo no está sabiendo clasificar correctamente, pues de 5 direcciones de tipo MATCH, solo ha clasificado correctamente 1, clasificando las otras 4 como NO MATCH. Esta única dirección clasificada correctamente es altamente probable que sea la dirección MATCH que se encontraba también en el subconjunto de entrenamiento, como se comentó previamente.

Esta última métrica demuestra que el algoritmo de generación de variaciones ha sido

insuficiente para balancear los datos, pues la proporción MATCH/NO MATCH resultante ha sido de 1/4, insuficiente para que el modelo trabaje de forma óptima.

Respecto a los tiempos, se obtienen resultados similares a la fase anterior: el algoritmo Gaussiano supera en varias magnitudes a los algoritmos *XGBoost* y *Random Forest* en tiempos de entrenamiento y prueba. Las métricas de rendimiento no son suficientemente elevadas como para considerar a ningún modelo funcional, por lo que esta comparativa de tiempos no es relevante todavía.

5.3.3 Clasificación mediante variaciones extendidas

Mejora del algoritmo de variaciones

En la tercera versión del modelo, la estrategia de generación de subconjuntos de entrenamiento y prueba se basa en mejorar el algoritmo de variación de direcciones para que sea capaz de generar más direcciones derivadas y, por ende, evitar el problema de desbalance acarreado desde la primera fase.

Para este proceso el algoritmo de variación implementará las siguientes variaciones nuevas:

- Variaciones ortográficas
- Variaciones posicionales

Las variaciones ortográficas se encargan de tomar una dirección original y añadir errores ortográficos de forma intencionada, como puedan ser letras intercambiadas o incorrectas, palabras en orden equivocado, etc.

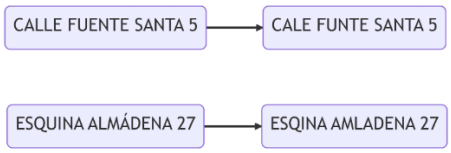

Ilustración 25. Modelo de variaciones ortográficas.

Este mecanismo de variación cuenta con un beneficio frente a las variaciones semánticas implementadas en la versión anterior del modelo, pues el número de errores insertados en una dirección es variable, por lo que se pueden producir muchas más variaciones por ortografía que por semántica. En este proyecto se ha limitado a 5 las variaciones ortográficas por dirección original.

Las variaciones posicionales, por otro lado, se encargan de tomar una dirección original y cambiar ciertos campos de sitio, sabiendo que la dirección resultante conservará su significado original. El campo por desplazar es el nombre del municipio (**nommun**), que se desplaza del inicio de la dirección (donde se suele escribir) al final de esta, pues se entiende que el significado de la dirección se mantiene.

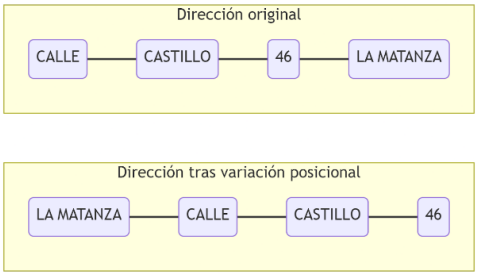

Ilustración 26. Modelo de variaciones posicionales.

Como se observa en el gráfico anterior, desplazando el **nommun** se obtiene una dirección nueva semejante a la original e igual de válida. Aplicando esto, se obtienen 2 direcciones válidas a partir de una dirección original.

Apilando todos los mecanismos de variación en una misma secuencia (*pipeline*) se obtiene un aumento considerable de direcciones MATCH en los subconjuntos de datos, aproximadamente 30 veces el número original de entradas MATCH.

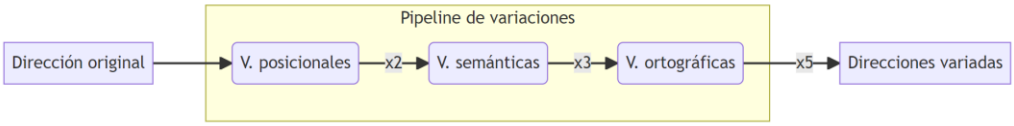

Ilustración 27. Secuencia de variaciones de direcciones.

Generación del modelo

Con la secuencia de variaciones mejorada, se ha realizado el mismo proceso de entrenamiento y testeo de las versiones anteriores del modelo. El balance de clases ha sido el siguiente:

Clase	Direcciones	Porcentaje
MATCH	37,984	3,66%
NO MATCH	998,559	96,34%

Tabla 21. Proporción promedio de clases en modelo con variaciones extendidas.

En la tabla anterior se puede observar que el resultado de la secuencia ha sido el balanceo de datos más notorio realizado hasta esta fase, pues se han alcanzado un 3,66% de direcciones MATCH frente al 0,4% de la fase anterior. Este aumento de direcciones permite la creación de los subconjuntos de entrenamiento y pruebas sin que existan direcciones MATCH repetidas entre ambos conjuntos, como ocurría en las fases anteriores del proyecto de forma excepcional. Sin embargo, esta mejora sigue sin ser suficiente como para que el modelo funcione adecuadamente, como se observa en las siguientes tablas:

Fase de creación	Tiempos de ejecución (s)		
	Random Forest	***Naive Bayes* Gaussiano**	***XGBoost***
Creación del conjunto de datos	658	658	658
Entrenamiento del modelo	121,15	0,72	30,59
Predicción del modelo	13,71	1,67	3,68

Tabla 22. Tiempos de ejecución de modelo de clasificación con variaciones extendidas.

Las métricas de rendimiento se han obtenido mediante métricas ponderadas respecto al subconjunto de pruebas, al igual que en el apartado anterior. Los resultados del modelo con variaciones son los siguientes:

Algoritmo	Exactitud (%)	Precisión (%)	Sensibilidad (%)
Random Forest Classifier	87,9883	16,5249	100,00
Gaussian NB	92,7650	37,4000	96,83
XGBClassifier	79,5990	6,4377	99,99

Tabla 23. Métricas de rendimiento de modelo de clasif. con variaciones extendidas.

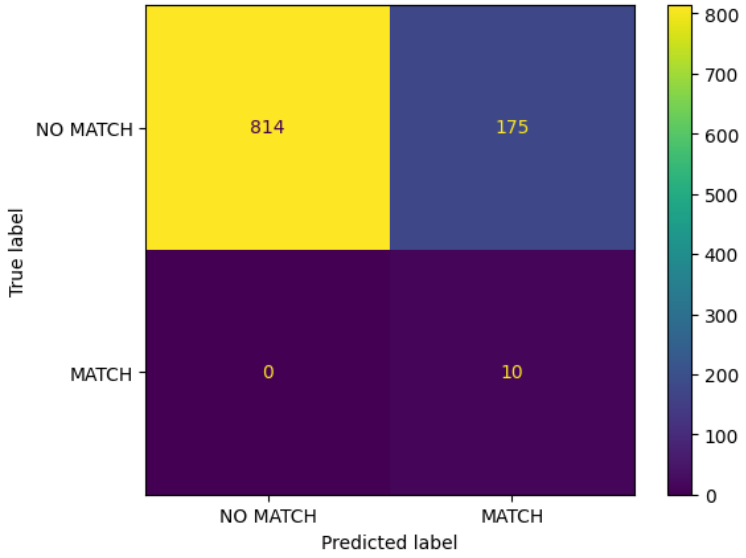

Ilustración 28. Matriz de conf. de modelo de clasif. con variaciones extendidas.

Como se observa en la Tabla 23, los valores de esta versión se mantienen similares a la fase anterior, un promedio de 86,7% de exactitud.

Valorando la sensibilidad de la clase MATCH se aprecia que más del 95% de las entradas MATCH en cualquiera de los 3 algoritmos implementados se clasifican correctamente. Sin embargo, la precisión MATCH se mantiene baja, por lo que existen muchas entradas de la clase NO MATCH que están siendo erróneamente clasificadas como MATCH. Al observar la representación

la matriz de confusión se confirman las sospechas sobre el modelo: un promedio de 120 entradas por modelo han sido clasificadas erróneamente como MATCH, frente a las 10 entradas MATCH clasificadas correctamente.

Una posible explicación a esto puede ser que el número de entradas MATCH utilizadas para entrenar el modelo sea insuficiente para que identifique sus características; Aunque se extraigan una variedad de direcciones MATCH derivadas de las originales, el modelo está limitado a 10 MATCH y 10 NO MATCH a la hora de entrenar, lo que puede resultar insuficiente para que el modelo funcione correctamente.

Existe la posibilidad de que, aunque se aumenten el número de entradas por subconjunto, el número de entradas MATCH siga siendo insuficiente, incluso habiendo aplicado la secuencia de variaciones completo. Para demostrar esto, se dispone de un supuesto práctico ideal, en el que se den todas las condiciones para que los subconjuntos estén balanceados. En este supuesto se busca algún municipio que contenga pocas direcciones, de tal forma que la desigualdad entre direcciones MATCH y NO MATCH sea la mínima posible: el municipio de **Betancuria**.

Betancuria tiene registradas en el conjunto de datos original (previo a la muestra) un total de 1.170 direcciones, siendo el municipio con menos direcciones registradas, y el tercero con menos IDs registrados (556 después de Fuencaliente de la Palma con 484 y Agulo con 338). El número de direcciones por ID se distribuye según la siguiente tabla:

Uuid_idt (ID)	**Frecuencia**
"6D2EDBEC-3251-11E8-A9A5-480FCF5217B3"	71
"C707DBA9-3EDA-11EB-845E-5FC45ADE36E3"	25
"C0E74708-0AD1-11EA-8401-7756D0C25261"	20
"ACEA79F2-7B7A-11EB-896E-6F4967C1D9D4"	14
"6B4132F0-3251-11E8-BF79-480FCF5217B3"	13
"8D631995-79D7-11EB-A554-CB98385CFF1E"	11
"8D636897-79D7-11EB-A554-CB98385CFF1E"	11
"C0C126AD-0AD1-11EA-8401-7756D0C25261"	10
"C07709C5-0AD1-11EA-8401-7756D0C25261"	10
"C08531EB-0AD1-11EA-8401-7756D0C25261"	10
"C0D37E60-0AD1-11EA-8401-7756D0C25261"	9

Tabla 24. Ejemplo de muestra de IDs del municipio "Betancuria".

Hay una cierta cantidad de IDs con más de 10 direcciones registradas, pero sabiendo que en el municipio hay alrededor de 500 IDs, no dejan de ser una minoría. Buscando un caso óptimo

para el supuesto, se tomará como referente el ID "C0C126AD-0AD1-11EA-8401-7756D0C25261", que tiene 10 direcciones registradas:

- Este ID cuenta con 10 direcciones registradas, que se multiplicarán por 30 en el proceso de variación para generar 300 direcciones de MATCH.
- Con 300 direcciones de MATCH, entonces se dispone de 1170 - 10 = 1160 direcciones de NO MATCH.

El desbalance en este caso es bastante claro, y es que se cuenta con casi 1 dirección de MATCH por cada 4 direcciones de NO MATCH.

Hasta esta fase se ha sabido que iba a haber menos direcciones MATCH que NO MATCH, por lo cual hasta ahora se han limitado el número de direcciones MATCH de entrenamiento a un 70% de las totales, dejando el 30% restante para probar el modelo. Además, se han limitado el número de direcciones NO MATCH del subconjunto de entrenamiento a un tamaño igual al de las MATCH. Con esto se busca:

- Balancear el subconjunto de entrenamiento.
- Permitir una cantidad de direcciones MATCH que permita al modelo generalizar sus características.

Respecto a los tiempos de ejecución, se observan resultados similares a las fases anteriores: el algoritmo *Naive Bayes* Gaussiano supera tanto en tiempos de entrenamiento como de prueba a los algoritmos *Random Forest* y *XGBoost*.

5.3.4 Clasificación mediante variaciones y balanceo

Aumento de direcciones mediante sobre-muestreo

Como se ha visto en las fases anteriores, el balanceo de MATCH realizado hasta ahora ha resultado insuficiente: Un 3,6% de entradas MATCH no son suficientes como para considerar ambas clases balanceadas.

En la cuarta y última versión del modelo de clasificación se ha decidido retomar el balanceo de direcciones mediante la secuencia de variaciones y añadirle un componente más para realizar el balanceo de la forma más completa posible: aplicando SMOTE [22] para realizar un sobre-muestreo a la clase minoritaria MATCH.

Generación del modelo

Para el balanceo del modelo se ha aplicado SMOTE al subconjunto de datos formado por todas las direcciones NO MATCH del municipio a analizar junto con las direcciones MATCH resultantes de la secuencia previamente desarrollada. Este proceso de sobre-muestreo ha volcado los siguientes resultados:

Clase	Direcciones	Porcentaje
MATCH	1017,5	50%
NO MATCH	1017,5	50%

Tabla 25. Proporción promedio de clases en modelo con sobre-muestreo.

Se puede observar un balanceo ideal en el promedio de las muestras, resultado de haber

aplicado el algoritmo SMOTE sobre las ya aumentadas muestras MATCH. Una vez balanceadas las clases, y para asegurar que el conjunto de datos de cada ID sea suficiente y no pueda dar lugar a errores, se ha decidido aplicar una validación cruzada de 5 pliegos. Los tiempos de ejecución son los siguientes:

Fase de creación	Tiempos de ejecución (s)		
	Random Forest	***Naive Bayes* Gaussiano**	***XGBoost***
Creación del conjunto de datos	337	337	337
Entrenamiento del modelo	2455,72	38,31	335,45
Predicción del modelo			

Tabla 26. Tiempos de ejecución de modelo de clasificación con sobre-muestreo.

Las métricas de rendimiento se han obtenido mediante métricas ponderadas respecto al subconjunto de pruebas, al igual que en el apartado anterior. Los resultados del modelo con variaciones son los siguientes:

Algoritmo	Exactitud (%)	Precisión (%)	Sensibilidad (%)
Random Forest Classifier	99,4135	99,3808	99,4410
Gaussian NB	93,4233	94,6958	92,4043
XGBClassifier	99,0162	98,6560	99,4047

Tabla 27. Métricas de rendimiento de modelo de clasificación con sobre-muestreo.

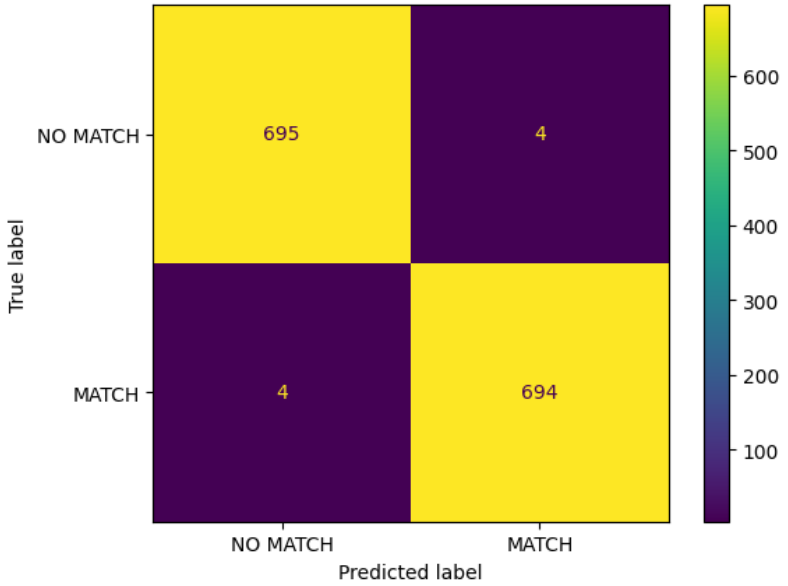

Ilustración 29. Matriz de conf. de modelo de clasif. con sobre-muestreo.

Como se observa en las métricas resultantes, el modelo final funciona de forma óptima tras el balanceo de los subconjuntos de entrenamiento y prueba.

La exactitud promedio de los tres algoritmos de clasificación supera el 97%, siendo el valor más bajo el asociado al *Naive Bayes* Gaussiano (93,42%). Analizando la precisión y sensibilidad del modelo, se observan valores excelentes, donde ninguna métrica baja del 92%,

siendo la más baja la sensibilidad de *Naive Bayes* Gaussiano.

Tanto por la exactitud y la sensibilidad se determina que todos los algoritmos clasifican de forma excelente las clases contenidas, siendo la más reveladora el valor de la precisión: en la fase anterior era muy baja, indicando que clasificaba como MATCH muchas más direcciones de las que debía (generalizaba demasiado). En esta última fase, sin embargo, se aprecia que la precisión se mantiene muy alta, indicando que ambas clases están siendo clasificadas de manera correcta.

En esta última fase de clasificación los tres modelos de clasificación demuestran resultados correctos, lo que permite realizar una comparativa de tiempos de ejecución. Se observa que el algoritmo *Naive Bayes* Gaussiano supera por varias magnitudes a los algoritmos *Random Forest* (64 veces más rápido) y *XGBoost* (casi 9 veces más rápido).

Capítulo 6. Modelos de agrupamiento

Alternativamente a los modelos de clasificación, y aprovechando el paradigma MATCH/NO MATCH utilizado anteriormente para dividir el conjunto de datos original, se ha propuesto utilizar modelos de agrupamiento para obtener resultados similares a los obtenidos en las fases anteriores del proyecto.

6.1 Planteamiento del modelo de agrupamiento

El modelo de agrupamiento MATCH/NO MATCH aprovecha el funcionamiento básico de un modelo de agrupamiento binario: al forzar a un modelo a agrupar un conjunto de datos cualesquiera en 2 grupos (agrupamiento binario), producirá unos resultados similares a los siguientes.

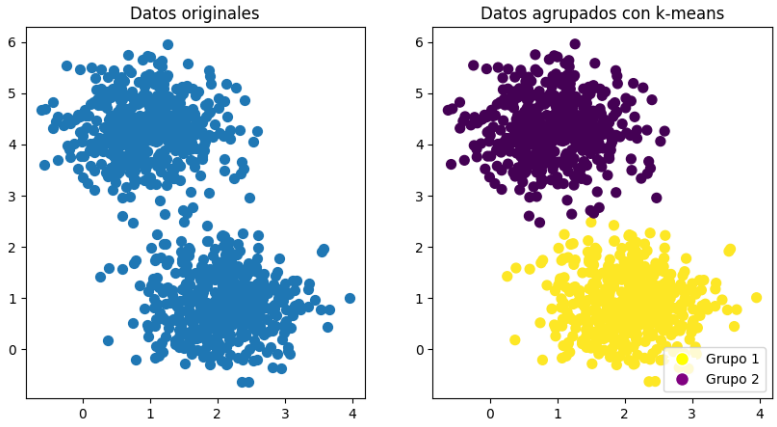

Ilustración 30. Ejemplo de funcionamiento de modelo de agrupamiento binario.

Como se observa en el gráfico anterior, el funcionamiento de un modelo de agrupamiento binario es equivalente al de un modelo de clasificación binario: ambos reciben datos como entrada y les asignan una de las dos etiquetas posibles, con la diferencia de que en la clasificación a la etiqueta se denomina “clase”, mientras que en el agrupamiento se denomina “grupo”.

Conociendo su semejanza, se pueden utilizar con las mismas métricas y procedimientos que los algoritmos de clasificación, siendo igualmente válidos a la hora de dividir las muestras en 2 grupos.

Sin embargo, estos grupos no siempre se asignarán en el orden esperado por el usuario: existe la posibilidad de que asocie los grupos MATCH a las direcciones de la clase NO MATCH, y viceversa. Un ejemplo de este comportamiento es el siguiente gráfico:

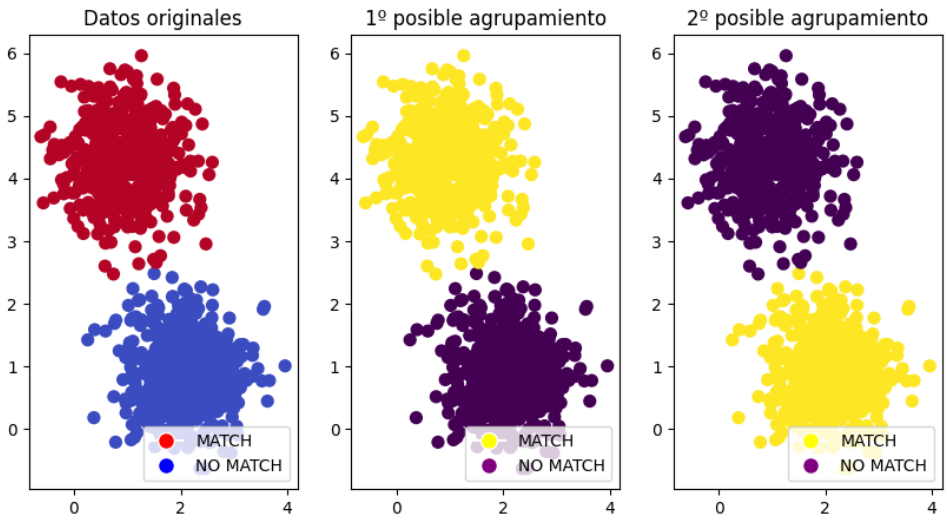

Ilustración 31. Posibles comportamientos de modelo de agrupamiento binario.

Como se observa en el ejemplo anterior, ambos agrupamientos son virtualmente iguales, pues las entradas han sido divididas en los mismos grupos en ambas ejecuciones del modelo. Sin embargo, en el primer agrupamiento los grupos asignados coinciden con las clases originales del problema, mientras que en el segundo son los opuestos.

Para solucionar este problema se ha propuesto obtener los grupos asignados por el agrupamiento como primera solución posible, para luego generar otros grupos asignados que sean los opuestos a los del primer grupo, exactamente igual al gráfico anterior donde los grupos del segundo agrupamiento resultan ser los opuestos del primero. De esta forma se obtiene el mismo agrupamiento con dos representaciones diferentes, y se selecciona como resultado final aquella que haya acertado más entradas: la solución con mayor valor de exactitud.

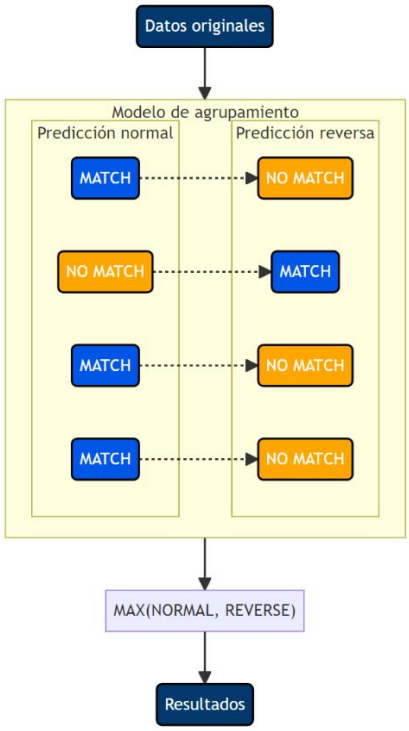

Ilustración 32. Inversión de resultados de agrupamiento.

6.2 Generación del modelo

Para poder determinar de forma directa el número de grupos que debe de utilizar el algoritmo de agrupamiento, se ha decidido implementar el algoritmo K-medias. Se han especificado 2 grupos y 10 ejecuciones con cada centroide. Los resultados computacionales obtenidos son los siguientes:

Fase del modelo	**Tiempo de ejecución (s)**
Creación del conjunto de datos	0,7
Entrenamiento del modelo	0,1
Valoración del modelo	0,1

Tabla 28. Tiempos de ejecución de modelo de agrupamiento con k-medias.

Algoritmo	**Exactitud (%)**	**Precisión (%)**	**Sensibilidad (%)**
Random Forest Classifier	72,1855	72,5544	74,2720

Tabla 29. Métricas de rendimiento de modelo de agrupamiento con k-medias.

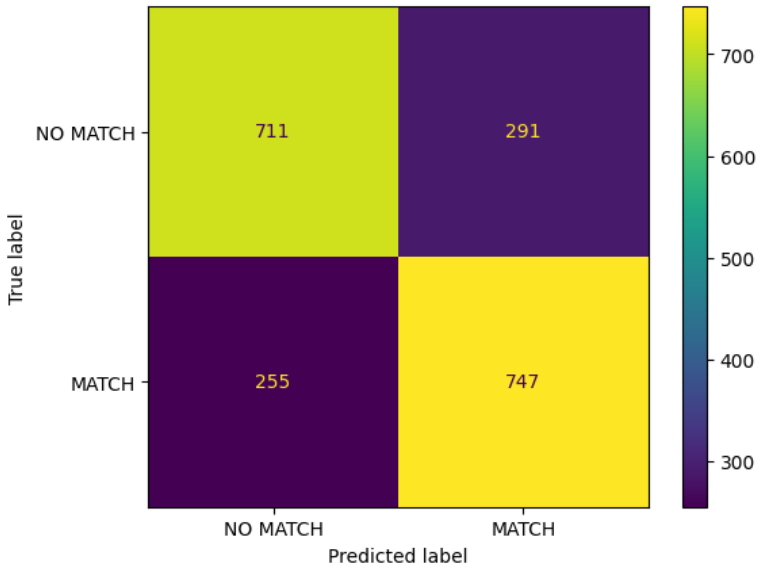

Ilustración 33. Matriz de confusión de modelo de agrupamiento con k-medias.

Se puede observar unos resultados correctos de exactitud, en los que aproximadamente ¾ de las direcciones han sido agrupadas correctamente. Al contrario que en las primeras fases del modelo de clasificación, los valores de precisión y sensibilidad general se mantienen similares a la exactitud, alrededor del 72%.

La matriz de confusión demuestra que los resultados de precisión y sensibilidad se asocian a datos bien balanceados, pues no existen valores anómalos extremadamente bajos como se podían observar en las primeras fases del modelo de clasificación.

Capítulo 7. Conclusiones y líneas futuras

Los resultados de concordancia de direcciones aplicando las técnicas vistas en los estudios mencionados al inicio [2] [6] han sido excelentes, obteniendo exactitudes superiores al 93% en los tres algoritmos implementados, además de métricas de precisión y sensibilidad con resultados similares.

En la aplicación mediante algoritmos de agrupamiento también se han obtenido resultados correctos, observando un promedio de métricas superior al 72%, incapaz de igualar la clasificación, pero igualmente válidas.

Estos resultados permiten determinar que el objetivo inicial, construir un modelo capaz de detectar coincidencia de direcciones aplicando técnicas de aprendizaje automático, ha sido completado con éxito. No solo se ha alcanzado el objetivo mediante algoritmos de clasificación, como en los estudios citados, sino que se han alcanzado resultados prometedores con algoritmos de agrupamiento, aprovechando la naturaleza binaria del paradigma MATCH/NO MATCH a favor del proyecto.

También se ha logrado desarrollar una metodología efectiva para resolver problemas de coincidencia de direcciones de forma correcta y eficiente, sea mediante clasificación binaria o mediante agrupamiento binario.

Como posibles mejoras y añadidos al proyecto, se podrían buscar métodos para optimizar los tiempos de generación de los subconjuntos de datos de entrenamiento y prueba, así como los tiempos de entrenamiento de los modelos. Una posible opción para esto sería el uso de librerías o herramientas de concurrencia, para paralelizar los cálculos requeridos y reducir los tiempos. Al comienzo de este proyecto se propuso el uso de *Apache Spark* [32] como herramienta de optimizado de los cálculos, pero por falta de tiempo se descartó su implementación.

También se propone para futuras revisiones del proyecto el uso de diferentes configuraciones de redes neuronales, con el objetivo de observar los resultados obtenidos aplicando técnicas de aprendizaje profundo *(deep learning)*.

Como último posible añadido, se podrían aplicar las técnicas aplicadas en este estudio a conjuntos de datos de diferentes regiones para ver el rendimiento en otras localizaciones.

Capítulo 8. Summary and Conclusions

The address matching results obtained by applying the techniques mentioned in the studies [2] [6] has been excellent, with accuracies exceeding 93% in all three implemented algorithms. Furthermore, precision and recall metrics have shown similar outcomes.

In the implementation using clustering algorithms, correct results have also been obtained, with an average precision and recall score above 72%. While unable to match the classification excellent performance, these results are still considered valid.

These results allow us to determine that the initial objective of building a model capable of performing address matching using machine learning techniques has been successfully achieved. Not only has the objective been accomplished through classification algorithms, as in the cited studies, but promising results have also been obtained with clustering algorithms, leveraging the binary nature of the MATCH/NO MATCH paradigm in favor of the project.

As possible improvements and additions to the project, one could explore methods to optimize the generation times of training and testing data subsets, as well as the training times of the models. One possible option for this would be to use concurrency libraries or tools to parallelize the required computations and reduce the overall time. At the beginning of this project, the use of Apache Spark [32] was proposed as a tool for optimizing computations, but due to time constraints, its implementation was discarded.

Furthermore, it is proposed for future revisions of the project to explore the use of different configurations of neural networks, with the objective of observing the results obtained by applying deep learning techniques.

As a final potential addition, the techniques applied in this study could be applied to datasets from different regions to assess the performance in other locations.

Capítulo 9. Presupuesto del proyecto

9.1 Personal

Científico de datos *junior* [33]	45 €/h brutos

9.2 Recursos computacionales

Servicio de cómputo por GPU en la nube [34]	1,82 €/h

9.3 Recursos digitales

Conjunto de datos de direcciones	0 €
Modelo de representación de texto pre-entrenado	0 €

9.4 Desglose de tiempos

Fase	**Tiempo de trabajo (h)**	**Tiempo de cómputo (h)**
Estudio inicial de artículos de coincidencia de direcciones	40	0
Análisis inicial del conjunto de datos	32	4
Limpieza de datos	8	4
Preprocesado de datos	8	4
Estudio de algoritmos de representación de textos	32	6
Aplicación de algoritmos de representación de textos	8	6
Diseño de modelo genérico	8	0
Implementación de modelo genérico	8	8
Diseño de modelo base	16	0
Implementación de modelo base	8	4

Diseño de modelo con variaciones	16	0
Implementación de modelo con variaciones	8	4
Diseño de modelo con variaciones extendidas	16	0
Implementación de modelo con variaciones extendidas	8	4
Diseño de modelo con sobre-muestreo	8	0
Implementación de modelo con sobre-muestreo	8	4
Diseño de modelo de agrupamiento binario	8	0
Implementación de modelo de agrupamiento binario	8	4
Conclusiones de proyecto	8	0
TOTAL	256	52

9.5 Costo total

Costo de personal	**Costo de cómputo**	**Costo total**
11520€	94.64 €	11614.64€

Capítulo 10. Bibliografía

1] J. A. G. Yanes, R. B. Villalba y M. S. H. García, «Sistema de georreferenciación para fines estadísticos,» Noviembre 2021. [En línea]. Available: https://jecas.es/wp-content/uploads/2021/11/21.4.ISTAC_Sistema-georreferenciacion.pdf. [Último acceso: 22 Mayo 2023].

2] D. Yates, M. Z. Islam, Y. Zhao, R. Nayak, V. Estivill-Castro y S. Kanhere, «PostMatch: A framework for efficient address matching,» de *Communications in Computer and Information Science*, Singapore, Springer Singapore, 2021, pp. 136-151.

3] S. &. A. D. Comber, «Machine learning innovations in address matching: A practical comparison of word2vec and CRFs.,» de *Transactions in GIS*, 2019, pp. 334-348.

4] J. Feigenbaum, «Jaro-Winkler distance,» [En línea]. Available: https://scholar.harvard.edu/jfeigenbaum/software/jaro-winkler-distance. [Último acceso: 12 Junio 2023].

5] J. Brownlee, «Machine Learning Mastery: How to Calculate Precision, Recall, and F-Measure for Imbalanced Classification,» 3 Enero 2020. [En línea]. Available: https://machinelearningmastery.com/precision-recall-and-f-measure-for-imbalanced-classification/. [Último acceso: 23 Mayo 2023].

6] L. Syne, «Machine learning and NLP approaches in address matching,» 2022. [En línea]. Available: https://riull.ull.es/xmlui/handle/915/31641. [Último acceso: 2023].

7] «TensorFlow Core: Word2vec,» Google, [En línea]. Available: https://www.tensorflow.org/tutorials/text/word2vec. [Último acceso: 22 Mayo 2023].

8] «A gentle introduction to Doc2Vec,» Medium, 26 Julio 2017. [En línea]. Available: https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e. [Último acceso: 22 Mayo 2023].

9] A. Chaudhary, «A Visual Guide to FastText World Embeddings,» 21 Junio 2020. [En línea]. Available: https://amitness.com/2020/06/fasttext-embeddings/. [Último acceso: 25 Mayo 2023].

R. Kulshrestha, «BERT Explained: What it is and how does it work?,» 26 Octubre 2020. [En línea]. Available: https://towardsdatascience.com/keeping-up-with-the-berts-

10] 5b7beb92766. [Último acceso: 22 Mayo 2023].

D. Soni, «Towards Data Science: Supervised vs. Unsupervised learning,» 22 Marzo
11] 2018. [En línea]. Available: https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d. [Último acceso: 12 Junio 2023].

T. Wood, «Random Forests Definition,» 9 Septiembre 2020. [En línea]. Available:
12] https://deepai.org/machine-learning-glossary-and-terms/random-forest. [Último acceso: 25 Mayo 2023].

. S. Abhishek, «Analytics Vidhya,» 26 Abril 2023. [En línea]. Available:
13] https://cdn.analyticsvidhya.com/wp-content/uploads/2020/02/rfc_vs_dt1.png. [Último acceso: 23 Mayo 2023].

S. Hrouda-Rasmussen, «Towards data Science: Gaussian Naive Bayes,» Medium, 7
14] Mayo 2021. [En línea]. Available: https://towardsdatascience.com/gaussian-naive-bayes-4d2895d139a. [Último acceso: 23 Mayo 2023].

«XGBoost,» [En línea]. Available:
15] https://xgboost.readthedocs.io/en/stable/index.html#. [Último acceso: 22 Mayo 2023].

Hiros, «k-Means Clustering,» 19 Mayo 2019. [En línea]. Available:
16] https://h1ros.github.io/posts/k-means-clustering/#disqus_thread. [Último acceso: 31 Mayo 2023].

I. Dabbura, «K-means Clustering: Algorithm, Applications, Evaluation Methods, and
17] Drawbacks,» 17 Septiembre 2018. [En línea]. Available: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. [Último acceso: 31 Mayo 2023].

«Sklearn: Clustering,» [En línea]. Available: https://scikit-
18] learn.org/stable/modules/clustering.html. [Último acceso: 16 Junio 2023].

S. Yildirim, «Towards Data Science: DBSCAN Clustering - Explained,» 22 Abril 2020.
19] [En línea]. Available: https://towardsdatascience.com/dbscan-clustering-explained-97556a2ad556. [Último acceso: 18 Junio 2023].

J. Brownlee, «Machine Learning Mastery: What is a Confusion Matrix in Machine
20] Learning,» 18 Noviembre 2016. [En línea]. Available: https://machinelearningmastery.com/confusion-matrix-machine-learning/. [Último acceso: 28 Mayo 2023].

N. Cristianini y J. Shawe-Taylor, An introduction to support vector machines and
21] other kernel-based learning methods, 2013.

J. Brownlee, «SMOTE for Imbalanced Classification with Python,» 17 Marzo 2021. [En
22] línea]. Available: https://machinelearningmastery.com/smote-oversampling-for-imbalanced-classification/. [Último acceso: 22 Mayo 2023].

«SMOTE로 데이터 불균형 해결하기,» Medium, 11 Abril 2020. [En línea]. Available:
23] https://john-analyst.medium.com/smote%EB%A1%9C-%EB%8D%B0%EC%9D%B4%ED%84%B0-%EB%B6%88%EA%B7%A0%ED%98%95-%ED%95%B4%EA%B2%B0%ED%95%98%EA%B8%B0-5ab674ef0b32. [Último acceso: 31 Mayo 2023].

S. Vajjala, B. Majumder, A. Gupta y H. Surana, Practical natural language processing:
24] A comprehensive guide to building real-world NLP systems, Sebastopol, CA: O'Reilly Media, 2020.

N. B. Subramanian, «Introduction to Bag of Words, N-Gram and TF-IDF,» Aiaspirant,
25] 23 Septiembre 2019. [En línea]. Available: https://aiaspirant.com/bag-of-words/. [Último acceso: 22 Mayo 2023].

G. R. Sahani, «Understanding TF-IDF in NLP,» medium, 7 Octubre 2020. [En línea].
26] Available: https://medium.com/analytics-vidhya/understanding-tf-idf-in-nlp-4a28eebdee6a. [Último acceso: 22 Mayo 2023].

J. Devlin, M.-W. Chang, K. Lee y K. Toutanova, «BERT: Pre-training of deep
27] bidirectional Transformers for language understanding,» 11 Octubre 2018. [En línea]. Available: http://arxiv.org/abs/1810.04805https://arxiv.org/abs/1810.04805. [Último acceso: 18 Junio 2023].

«Keras: Deep Learning for humans,» [En línea]. Available: https://keras.io/. [Último
28] acceso: 22 Mayo 2023].

«NLTK: Natural Language Toolkit,» Team NLTK, [En línea]. Available:
29] https://www.nltk.org/. [Último acceso: 22 Mayo 2023].

A. Hernández y R. Martín, «Preln: A package for preprocessing text in spanish,» 3
30] Abril 2023. [En línea]. Available: https://github.com/Adri-Hdez/Preln. [Último acceso: 22 Mayo 2023].

A. Perez, E. T. Ariza, L. G. Pinto y M. Mazuecos, «Hugging Face: paraphrase-spanish-
31] distilroberta,» [En línea]. Available: https://huggingface.co/hackathon-pln-es/paraphrase-spanish-distilroberta. [Último acceso: 22 Mayo 2023].

«Apache Spark,» Apache Software Foundation, [En línea]. Available:
32] https://spark.apache.org/. [Último acceso: 19 Junio 2023].

«glassdoor: Sueldos para el puesto de Data Scientist en España,» 22 Junio 2023. [En
33] línea]. Available: https://www.glassdoor.es/Sueldos/data-scientist-sueldo-SRCH_KO0,14.htm. [Último acceso: 24 Junio 2023].

«Lambdalabs,» [En línea]. Available: https://lambdalabs.com/service/gpu-cloud.
34] [Último acceso: 24 Junio 2023].

«Gensim: Topic modelling for humans,» RARE Technologies Ltd., [En línea]. Available:

35] https://radimrehurek.com/gensim/s. [Último acceso: 22 Mayo 2023].

«NumPy,» [En línea]. Available: https://numpy.org/. [Último acceso: 22 Mayo 2023].
36]

«pandas - Python Data Analysis Library,» [En línea]. Available:
37] https://pandas.pydata.org/. [Último acceso: 22 Mayo 2023].

«Scikit-learn: Machine Learning in Python,» [En línea]. Available: https://scikit-
38] learn.org/stable/index.html. [Último acceso: 22 Mayo 2023].

J. P. Rojas, J. Cañete y D. Nguyen, «Spanish Word Embeddings,» 9 Octubre 2019. [En
39] línea]. Available: https://github.com/dccuchile/spanish-word-embeddings/tree/master#fasttext-embeddings-from-suc. [Último acceso: 9 Mayo 2023].

C. T. B. Miap, «Towards Data Science: Recurrent neural networks and natural
40] language processing,» 9 Junio 2019. [En línea]. Available: https://towardsdatascience.com/recurrent-neural-networks-and-natural-language-processing-73af640c2aa1. [Último acceso: 19 Junio 2023].