Universidad del País Vasco Euskal Herriko Unibertsitatea
BILBOKO INGENIARITZA ESKOLA
ESCUELA DE INGENIERÍA DE BILBAO
Ingeniaritza Goi Eskola Teknikoa
Escuela Técnica Superior de Ingeniería
Bilbao

MINERÍA DE DATOS

CLUSTERING DE DOCUMENTOS

18 de octubre de 2021

Alvaro Hernandez Rad
Kerman Sanjuan Malaxechevarria
Moritz Tim Pietig

Índice

Índice de figuras

Capítulo 1

Introducción

El *Text mining* es una campo de la minería de datos en la que la fuente de los datos son los textos, ya sean páginas web, correos electrónicos, foros o cualquier otro tipo de fuente de contenido. El objetivo de la minería de textos consiste en ser capaz de extraer conocimiento a partir de los textos de forma artificial.

En esta tarea se nos propone agrupar documentos en grupos en base a sus similitudes, es decir, hacer *clustering* de documentos. El *clustering* es un proceso importante dentro del *Machine Learning*. El objetivo de esta clasificación no-supervisada es descubrir agrupamientos naturales para un conjunto de instancias no clasificadas y de esa manera ofrecer una descripción de esos mismos datos, como si de aportar un nuevo atributo se tratase, siempre en términos de similitud/disimilitud. Dicho de una forma más sencilla, esta tarea tiene como finalidad principal lograr la agrupación de conjuntos de objetos no etiquetados. Estos subconjuntos de datos también se denominan como **clusters**. Cada cluster está formado por una colección de datos que son similares entre sí, pero que son distintos respecto a los objetos de otros clusters. Existen dos reglas principales a tener en cuenta a la hora de crear clusters.

1. Las instancias de un mismo cluster deben presentar similitud interna fuerte. Esta característica se conoce como **homogeneidad intra-cluster**.

2. Las instancias de distintos clusters deben presentar disimilitud fuerte. Esta característica se conoce como **separabilidad inter-cluster**.

Hoy en día, la minería de textos es ampliamente aplicada por una extensa variedad de usuarios y campos, como organizadores gubernamentales, investigadores o empresas, y en diferentes ámbitos. Estos son algunos ejemplos [2]:

1. **Investigación**: Sirve para el descubrimiento de conocimientos, la atención sanitaria y médica. En el pasado era más complicado obtener y analizar información relevante, mientras que con estas técnicas es más sencillo. La minería de textos permite a los investigadores encontrar más información y de forma más rápida y eficiente.

2. **Negocios**: Las grandes empresas utilizan la minería de textos para ayudar en la toma de decisiones y responder rápidamente a las consultas de los clientes en procesos tales como la gestión de riesgos o el filtrado de currículos.
3. **Seguridad**: En anti-terrorismo, el análisis de los blogs y otras fuentes de texto en línea se utiliza para prevenir delitos en Internet y luchar contra el fraude.

Así mismo, diariamente los sitios web que todos usamos usan la minería de texto para crear filtros más confiables y efectivos.

Dada la tarea a realizar, y siempre teniendo en cuenta que estamos trabajando en el campo del *text mining*, hemos decidido dividir esta tarea en los siguientes cinco pasos [3]. De esta forma, hemos creado esta especie de *working-pipeline*

1. **Recogida de datos**: Recoger datos de calidad sobre una temática en concreto.
2. **Preproceso de datos**: Proceso de .ªdecuamiento"de los datos.
3. **Enriquecimiento**: Se pueden añadir etiquetas a los datos con el objetivo de completar con más información.
4. **Transformación**: Conversión del texto a una representación numérica, por ejemplo un vector.
5. **Extracción de información**: Aplicar los algoritmos para la obtención y representación de datos.

1.1. OBJETIVO

Este proyecto tiene dos objetivos principales. Por un lado, realizar el *Document-Clustering* y por el otro, programar sin uso de librerías externas el algoritmo de clasificación no-supervisada *K-means*. Siempre respetando las reglas de homogeneidad intra-cluster y la separabilidad inter-cluster.

Capítulo 2

Obtención y Preprocesamiento de datos

2.1. Obtención de los datos

La repercusión generada por el Covid en nuestro día a día durante el último año y medio y el haber tratado con este conjunto de datos en proyectos anteriores [4] ha hecho que decidamos re-utilizar *Covid-19 Tweet Dataset*, siendo *kaggle* la fuente de datos [5]. Este dataset que en cuestión contiene acerca de 40000 instancias con los siguientes atributos:

1. Usuario
2. Nombre
3. Ubicación
4. Hora de publicación del tweet
5. Texto
6. Sentimiento del tweet acerca del Covid-19

2.2. Preprocesamiento de datos

Como se ha mencionado en la introducción, uno de los primeros pasos en el *text mining* es el preprocesamiento de datos, con el objetivo de limpiar y ordenar los datos. Debido a que nuestro objetivo es agrupar los tweets en base a su contenido, hemos trabajado únicamente con el atributo de **texto**. Sin embargo, aún siendo aprendizaje no-supervisado y la etiqueta de clase no siendo necesaria, hemos decidido mantenerlo para así poder evaluar en un futuro la calidad de nuestros agrupamientos.

Una vez tomada la decisión de los datos con lo que vamos a trabajar, comienza el proceso de "limpieza"de los datos. Este proceso ha sido dividido en pasos diferentes y subsecuentes, los cuales explicamos a continuación [6] [7].

2.2.1. Borrar las URL

Las URL son muy comunes en los tweets, no aportando por lo general información importante. Es por ello que hemos decidido eliminarlos.

```
Beware of counterfeits trying to sell fake masks at cheap prices. Let's defeat coronavirus threat, #Covid_19 colle
ctively. #BeSafe #BeACascader #CoronavirusReachesDelhi
#coronavirusindia

https://t.co/2Ikkmimj4f https://t.co/RB9rtt7Nkc
Beware of counterfeits trying to sell fake masks at cheap prices. Let's defeat coronavirus threat, #Covid_19 colle
ctively. #BeSafe #BeACascader #CoronavirusReachesDelhi
#coronavirusindia
```

Figura 2.1: Ejemplo de eliminación de una URL

2.2.2. Transformar los emoticonos

Hemos sustituido los emojis por palabras equivalentes debido a que en muchas ocasiones pueden ofrecernos información relevante.

```
text = "game is on 🔥"
convert_emojis(text)
```

```
'game is on fire'
```

Figura 2.2: Transformación de emoji a texto

2.2.3. Transformar los emoticonos

De la misma forma que en el paso anterior, los emoticonos han sido sustituidos por sus palabras equivalentes, evitando así perdida de información.

```
text = "Hello :-) :-)"
convert_emoticons(text)
```

```
'Hello Happy_face_smiley Happy_face_smiley'
```

Figura 2.3: Transformación de un emoticono a texto

2.2.4. Transformar las abreviaciones

Las abreviaciones han sido transformadas, por las oraciones o frases a las que sustituyen.

Por ejemplo: CYA ->See you later

2.2.5. Conversión a minúsculas

La letra mayúscula no tiene importancia en los datos. Por ello, se convertirá todo el texto a minúscula. Para ello, se utilizará una técnica de preproceso de datos llamada **'Lower Casing'**. La idea es convertir todos los inputs de texto al mismo formato. De esta manera las palabras 'text', 'Text' y 'TEXT' se tratarán del mismo modo. Gracias al uso de esta técnica lograremos no tener repeticiones de atributos.

2.2.6. Quitar las stopwords

Las stopwords, o palabras vacías, son palabras que no proporcionan información extra, es decir, son palabras sin significado, como los son los artículos, los pronombres, las preposiciones, etcétera. Es por ello que hemos decidido eliminarlas.

2.2.7. Quitar los signos de puntuación

Los signos de puntuación no son de nuestro interés, por lo que hemos decidido suprimirlos.

2.2.8. Corregir las palabras

Es muy importante corregir todas las palabras. Por ello, esta función corregirá las palabras que estén mal escritas.

2.2.9. Lemmanization

La lematización es un proceso lingüístico que consiste en, dada una forma flexionada (es decir, en plural, en femenino, conjugada, etc), hallar el lema correspondiente. El lema es la forma que por convenio se acepta como representante de todas las formas flexionadas de una misma palabra. Gracias a esta técnica de preproceso de textos, se pueden agrupar palabras de mismo significado y reducir así el espacio de atributos.

2.2.10. Corregir las palabras (de nuevo)

Es posible que el proceso de lematización hayamos dejado palabras mal escritas, es por ello que hemos vuelto a corregir todas las palabras. Recordemos que es importante trabajar siempre con palabras correctamente escritas.

2.3. ANÁLISIS DE DATOS

En este apartado vamos a analizar los datos estadísticos de nuestros datos para conocerlos mejor. Una vez terminado el preprocesamiento de datos, estos tendrán dos atributos, el tweet original preprocesado, y el sentimiento, el cual hemos decidido mantener, ya que nos puede resultar útil a la hora de evaluar la calidad de los clusters finales.

count	41157
unique	5
top	Positive
freq	11422

Figura 2.4: Estadísticas del dataset limpio

En la **imagen 2.4** podemos observar las estadísticas principales acerca de los datos del dataset. Se puede destacar que ahora existen instancias repetidas que surgen de la limpieza de los datos y que a la vez no existen valores perdidos ('missing values') en todo el conjunto de datos.

En el caso del clustering los datos son no-supervisados, pero aun así, nos viene bien tener una intuición sobre la frecuencia de cada clase.

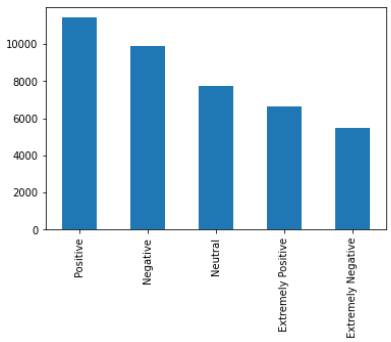

Figura 2.5: Frecuencia de aparición de cada clase.

Capítulo 3

Word Embedding

Hoy en día la representación de documentos de texto como vectores numéricos es un desafío muy grande en el campo del *Machine -Learning*. De hecho hay pocas técnicas que obtengan buenos resultados. Conocemos técnicas como **BOW (bag of words)** o **TF-IDF**, que consiguen resultados mediocres. Estos algoritmos emplean un número alto de atributos. Estas técnicas tratan las palabras como símbolos discretos atómicos asignando a cada palabra etiquetas aleatorias y sin relación entre ellas, sin aportar así, información ninguna.

Es por ello que para este trabajo utilizaremos una técnica bastante más avanzada, que nos permitirá lograr mejores resultados. Su nombre es Doc2Vec [8], y está basada mayormente en Word2Vec. Esta representación tiene notables ventajas frente a las representaciones clásicas. Por un lado, generalmente la dimensión del espacio suele ser de n=200, reduciendo así la talla de las técnicas clásicas que utilizan un número de atributos del orden de la talla del vocabulario ($|\sum|$).
Por otro lado, esta técnica permite hacer operaciones algebraicas con los vectores asociados a las palabras. Así pues, este espacio es apropiado para buscar relaciones entre palabras mediante combinaciones lineales de vectores.

Para entender bien Doc2Vec será necesario tener algún conocimiento acerca de Word2Vec.

Word2Vec es utilizado para representar palabras como vectores numéricos. Como veíamos el año pasado en la asignatura de Sistemas de Apoyo a la Decisión, cuando queríamos crear un modelo usando palabras, un método sencillo pero no suficiente era etiquetar cada palabra. Sin embargo, de esta manera las palabras perdían su significado. Por ejemplo, podíamos representar la palabra **Rojo** como **1**, la palabra **Verde** como **2** y la palabra **Edificio** como **3**. El problema radica en que de esta manera existe la misma relación entre las palabras **Rojo-Verde** y las palabras **Verde-Edificio**, cuando para nuestros intereses sería más conveniente que la representación del primer par de palabras fuese más cercana que la del segundo par.

Eso es lo que consigue *word2vec*. Este método es capaz de dar una representación numérica de cada palabra, siendo así posible ver la relación entre distintas palabras.

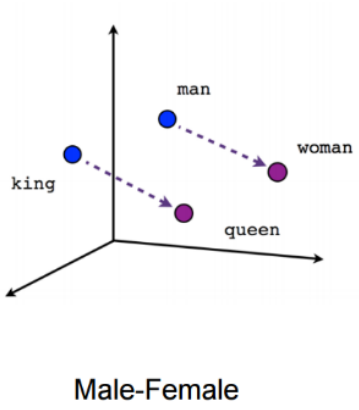

Figura 3.1: Relación entre palabras. 'Rey' es a 'Reina' lo que 'Hombre' a 'Mujer'

La representación Word2Vec funciona usando 2 algoritmos llamados **Continuous Bag-of-Words (CBOW)** y **Skip-Gram**.

Veamos como funcionan estos algoritmos.
Continuous Bag-of-Words crea una ventana que separa la palabra actual, para predecirla en base a su contexto (sus palabras vecinas). Es decir, se utiliza el contexto para predecir la palabra. Cada palabra es representada como un vector de atributos. Tras el entrenamiento, estos vectores de atributos se convierten en vectores de palabras. Como se ha mencionado antes, los vectores que representen palabras similares mantendrán métricas de distancia menores que con palabras que no se asemejen.

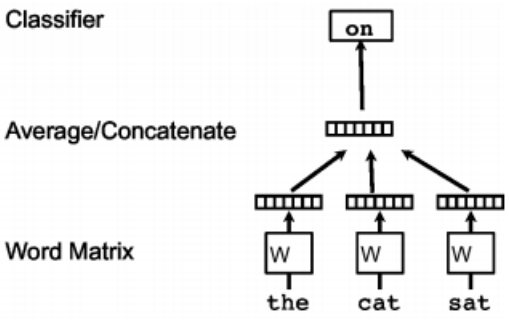

Figura 3.2: Algoritmo CBOW. Las palabras 'the', 'cat' y 'sat' se usan para predecir la palabra 'on'

Skip gram es el otro algoritmo y funciona a la inversa del CBOW. Mientras que el algoritmo anterior predice la palabra en base al contexto, este último utiliza una sola palabra para predecir el contexto. Es más lento que el algoritmo CBOW. El algoritmo funciona de la siguiente manera: Dado un conjunto de texto no-supervisado, se entrena una estructura

neuronal con distintas capas ocultas (hidden-layers) y se ajustan para predecir los contextos de cada palabra. Los parámetros (w_i) de la red sirven para representar las palabras en el espacio R^n. La idea detrás de esta aplicación es que palabras en similares contextos estarán localizadas en zonas similares del espacio.

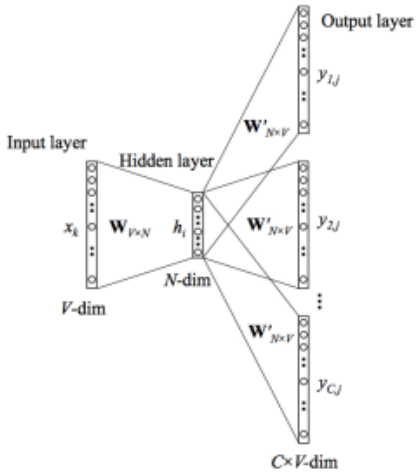

Figura 3.3: Modelo SkipGram [1]

Doc2Vec

Tras haber explicado el funcionamiento de los algoritmos anteriores será más sencillo entender como funciona *doc2vec*. El objetivo principal de *doc2vec* es crear una representación numérica de un documento, sin tener en cuenta su longitud. Sin embargo, los documentos no vienen en estructuras lógicas como las palabras, por lo que hay que encontrar otro método.

El concepto es el siguiente. Se utiliza el mismo método que en el *word2vec*, y se le añade un nuevo vector. La estructura del algoritmo es una extensión de la vista en CBOW, pero en vez de utilizar solamente palabras para predecir la siguiente palabra, se añade un vector de atributos extra, que pertenece únicamente al documento.

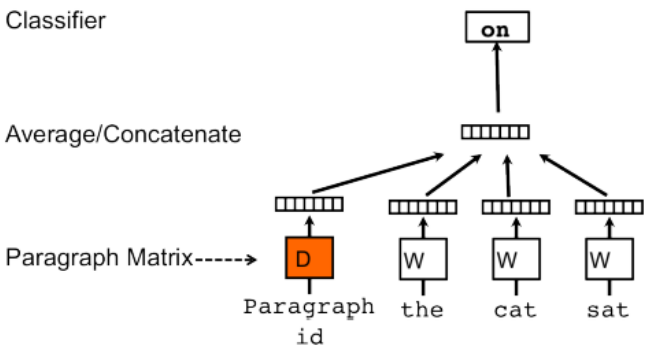

Figura 3.4: Modelo PV-DM

De esta forma cuando se entrenan los vectores de palabras W, también se entrena el vector del documento, D, que al finalizar contiene una representación numérica del documento.

Este modelo tiene el nombre **Distributed memory versión of Paragraph Vector (PV-DM)**. Funciona de manera que recuerda qué falta del contexto actual, el tema del párrafo. Mientras que los vectores de palabra representan el concepto de una palabra, el vector de documento trata de representar el concepto del documento.

El *doc2vec* se utiliza de la siguiente forma. Para el entrenamiento se necesita un set de documentos (en nuestro caso Tweets). Por cada palabra se crea un vector W, y se crea también un vector D por cada documento. El modelo también entrena pesos para una capa oculta softmax. Después, en la fase de inferencia, donde los inputs son nuevos documentos, se utilizan los pesos para calcular el vector de cada nuevo documento.

De esta manera habremos sido capaces de crear vectores numéricos, que utilizaremos en este caso para el proceso de clustering, partiendo de documentos de texto.

Capítulo 4

Clustering

4.1. K-MEANS CLUSTERING

4.1.1. Introducción al K-MEANS

K-means es un algoritmo de clasificación no-supervisada (clustering) que agrupa objetos en *k* grupos basándose en sus características. El agrupamiento se realiza minimizando la suma de distancias entre cada objeto y el centroide de su grupo o cluster. Se suele usar la distancia cuadrática.

El algoritmo consta de tres pasos:

1. **Inicialización**: una vez escogido el número de grupos, k, se establecen k centroides en el espacio de los datos, por ejemplo, escogiéndolos aleatoriamente. En nuestro caso, hemos utilizado una instancia aleatoria para inicializar el centroide.
2. **Asignación objetos a los centroides:** cada objeto de los datos es asignado a su centroide más cercano.
3. **Actualización centroides:** se actualiza la posición del centroide de cada grupo tomando como nuevo centroide la posición del promedio de los objetos pertenecientes a dicho grupo.

Se repiten los pasos dos y tres hasta que los centroides no se mueven, o se mueven por debajo de una distancia umbral en cada paso. A ese caso nosotros lo hemos denominado convergencia y depende de la cantidad de veces que se itere este algoritmo. Este resuelve un problema de optimización, siendo la función a optimizar la distancia intracluster o distancia intercluster.

4.1.2. Pseudo-código

En la realización de este proyecto hemos creado dos algoritmos de clustering diferentes, uno funcionando de forma iterativa y otra de forma matricial. Debido a la optimización que supone el uso de matrices, hemos decidido usar este. Pero ambos serán discutidos con sus pros y contras. Para facilitar el uso y la implementación hemos dividido cada paso en una función diferente.

4.1.2.1. Initialize Centroids

El primer paso ha sido desarrollar la inicialización de los clusters. Para ello hemos asignado K diferentes clusters a instancias aleatorias, dando así comienzo al algoritmo.

Algorithm 1 *Initialize Centroids*

```
while Quedan centroides por inicializar do
    Escoger un elemento aleatorio de las instancias
    if La instancia no está cogida por otro centroide then
        Asignar centroide a esa instancia
        Añadir a la lista de centroides
    else
        Asignar otra instancia al centroide
    end if
end while
return Lista de centroides
```

4.1.2.2. Assign Cluster

Con la posición de los centroides definida, tenemos que asignar un cluster a cada instancia en base al centroide más cercano.

Algorithm 2 *Assign Cluster*

Instancias ← Lista De Instancias
Clusters ← Lista De Clusters
while Quedan instancias por asignar **do**
    Calcular centroide más cercano a la instancia
    Asignar a ese centroide la instancia
end while
    return Lista de centroides con las instancias asignadas

4.1.2.3. Determine New Centroids

Para volver a empezar es necesario recalcular la posición de todos los centroides, utilizando la posición media de los elementos que conforman cada cluster.

Algorithm 3 *Initialize Centroids*

Centroides ← Lista De 0´s
while Centroide por calcular **do**
    Calcular la posición media de las instancias
    Asignar el nuevo centroide a cada cluster
end while
    return Lista de centroides

4.1.3. Resultados experimentales

4.1.3.1. Determinar el número de clusters

Debido a la naturaleza del clustering, es imposible saber cual es el teórico número óptimo de clusters, ya que esto se adapta en la tarea en cuestión. Es por eso que hemos decidido utilizar el número de clusters que determine el método del codo según el *índice de Dunn*, siendo este el resultado. Trataremos el punto del codo y el índice de Dunn en la **sección 4.2**.

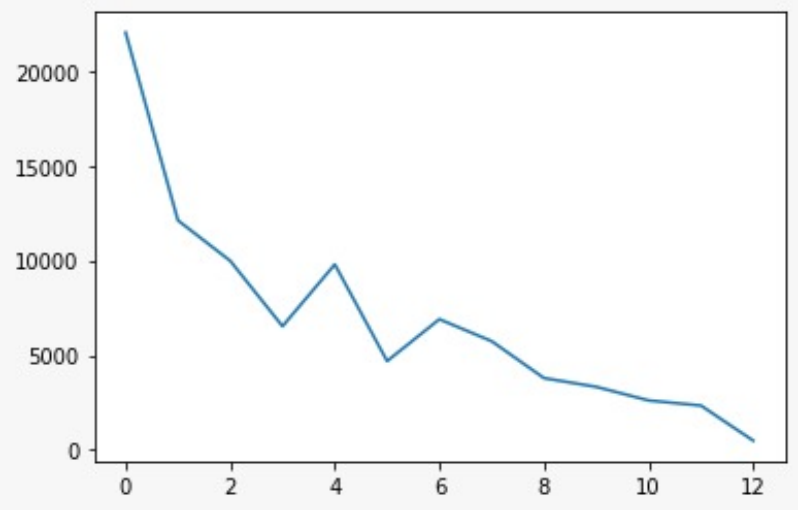

Figura 4.1: Índice de Dunn respecto a la cantidad de clusters

4.1.3.2. Examinar el número de iteraciones

Del mismo modo, también hemos querido examinar el número (media de diferentes ejecuciones) de iteraciones hasta la convergencia. Es por ello que también hemos presentado los resultados en un gráfico. En el gráfico de la **figura 4.2** se puede observar como el algoritmo utiliza las primeras iteraciones para ir colocando adecuadamente los centroides, ya que al haberlos inicializado aleatoriamente las posiciones iniciales estarán lejos de las posiciones que tendrán al final de la ejecución del algoritmo. Sin embargo, a partir de la iteración número 30 aproximadamente, los centroides se moverán muy poco a poco hasta converger. Por como hemos implementado nuestro algoritmo, este seguirá ejecutándose mientras exista alguna diferencia entre las posiciones de los centroides en las 2 últimas iteraciones.

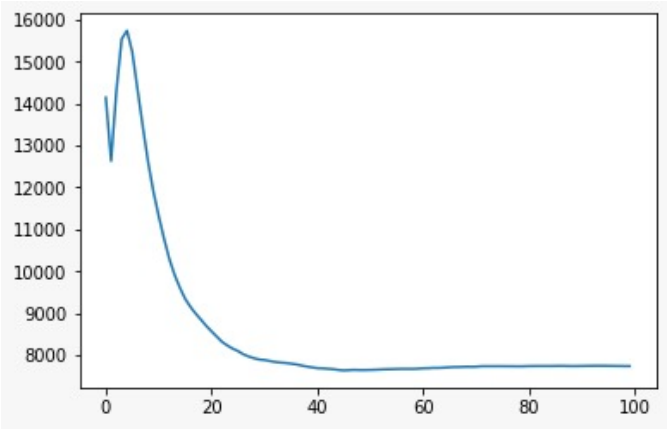

Figura 4.2: Optimalidad dependiendo la cantidad de iteraciones

4.1.3.3. Representación de los datos

Después de aplicar TSNE sobre el conjunto de datos, este es el resultado obtenido. Como se aprecia en la imagen, no se aprecian conjuntos de datos separables. Esto puede ser debido al espacio de representación obtenido o la técnica utilizada.

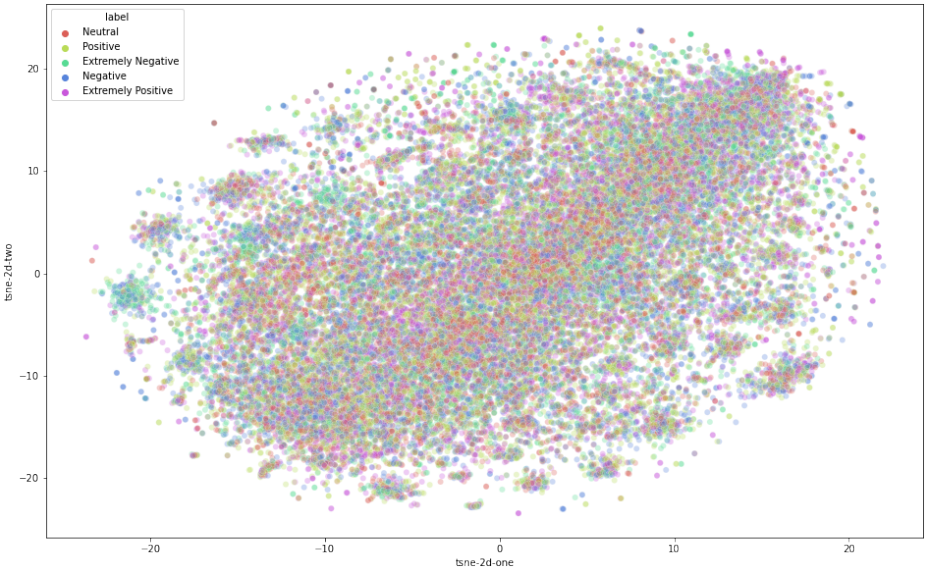

Figura 4.3: Representación del conjunto de datos en dos dimensiones.

4.1.4. Análisis crítico y discusión de resultados

4.1.5. Rendimiento del software

Como hemos mencionado anteriormente, se han desarrollado dos versiones del software. Por un lado computando el proceso de forma iterativa (elementwise) y por el el otro de forma matricial. En la **figura 4.4** se pueden ver los resultados de los tiempos de ejecución dependiendo del algoritmo utilizado y la cantidad de instancias empleadas.

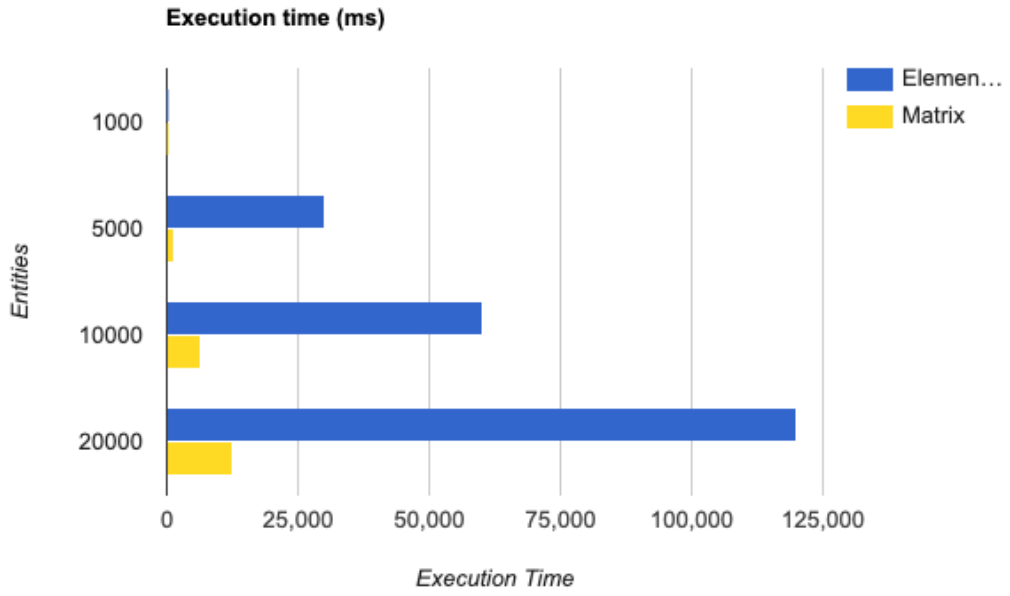

Figura 4.4: Tiempo estimado de ejecución dependiendo del algoritmo utilizado

4.1.6. Conclusiones

Se puede observar como la segunda implementación que hemos creado ejecuta mucho más eficazmente (respecto al tiempo) el algoritmo K-Means. Aún así, la implementación con matrices mantiene características de la implementación iterativa.

4.2. ÍNDICES DE CALIDAD

Debido a que el clustering es un proceso de aprendizaje-no supervisado, no podemos emplear las métricas convencionales que utilizabamos en la clasificación supervisada (*Precision, Recall, F2*), ya que carecemos de la etiqueta de cada clase. Es por ello que hemos utilizado las siguientes dos medidas de evaluación para estimar la calidad de nuestro agrupamiento.

4.2.1. Índice de DUNN

El índice de Dunn es una métrica para evaluar el buen funcionamiento de los algoritmos de clustering. La razón por la que hemos decidido utilizar este índice es por que su objetivo es identificar un conjunto de clusters que sean compactos, con una varianza pequeña entre los miembros del cluster, y que estos estén bien separados de los miembros de los otros clusters, es decir, busca maximizar las reglas principales a la hora de crear clusters, la **homogeneidad intra-cluster** y la **separabilidad inter-cluster** [9]. Técnicamente el índice de Dunn está definido como la proporción entre la mínima distancia inter-grupo y la máxima distancia intra-grupo. Para cada partición de grupo, el índice de Dunn puede ser calculado por la fórmula siguiente:

$$D = \frac{min_{1\leq i\leq j\leq n} d(i,j))}{max_{1\leq k\leq n} d'(k)} \tag{4.1}$$

donde *d(i,j)* representa la distancia entre grupos i y j, y *d '(k)* mide la distancia intra-grupo del grupo k. La distancia inter-grupo d(i,j) y intra-grupo pueden ser cualquier número de medidas de distancia, como por ejemplo la distancia entre los centroides de los clusters y la distancia de una instancia al centroide de su cluster. El criterio interno busca grupos con alta semejanza intra-grupo y baja semejanza inter-grupo, y por ello algoritmos que producen grupos con altos índices de Dunn son más deseables.

4.2.2. Elbow-Point

En el análisis de clusters, el método del codo es un método heurístico utilizado para determinar el número de clusters necesario para lograr mejores resultados. Para utilizar el método se ha de seleccionar un índice de calidad y mostrar gráficamente los resultados, mostrando en cada eje como cambia el valor del índice respecto al número de clusters. El punto que muestre un cambio brusco en la inercia del gráfico indicará cual es la cantidad óptima de clusters.

4.3. APLICACIÓN DEL MODELO INFERIDO

El objetivo del proyecto es crear un modelo que sirva para poder crear predicciones para nuevos datos que se introduzcan después del entrenamiento y creación del modelo. Para aplicar el modelo es importante someter a la nueva instancia al mismo preprocesamiento que han recibido los datos iniciales, así como representarla en el mismo espacio vectorial.

Una vez hemos realizado estos pasos y somos capaces de situar las nuevas instancias en el espacio vectorial junto al resto de instancias, buscamos cual es el centroide más cercano a la nueva instancia, para poder decir a que cluster pertenece.

Capítulo 5

Conclusiones y trabajo futuro

Una vez conseguimos los resultados, no podemos determinar si el clustering ha sido útil y aporta algo para nuestro conjunto de datos. Después de terminar con el proceso de clustering, hemos decidido recopilar las instancias más cercanas al centroide de cada cluster con el objetivo de encontrar algún patrón, pero el resultado no ha sido positivo. También hemos probado a hacer una comparación entre las etiquetas, que decidimos mantener al principio, de las diferentes instancias de un mismo cluster para ver si coincidían los agrupamientos con las etiquetas, pero una vez más el resultado no ha sido positivo.

Tras ver el resultado de la visualización de todas las instancias en el mismo espacio vectorial se podía intuir, por el entremezclado de los datos, que el resultado del clustering sería así, pero no ha sido hasta que hemos recopilado los resultados que hemos sacado esa conclusión.

Para futuros trabajos, viendo la dificultad del agrupamiento de textos, deberíamos consultar otros dataset para ver si los resultados mejoran.

Bibliografía

[1] Departamento de lenguajes y sistemas informáticos de la Escuela de Ingeniería de Bilbao. Clustering de documentos, figura 5. https://egela.ehu.eus/pluginfile.php/5345737/mod_resource/content/3/DM_Project_Clustering.pdf, 2021.

[2] Blog de la biblioteca de Traducción y Documentación de la Universidad de Salamanca. ¿qué es la minería de textos, cómo funciona y por qué es útil? https://universoabierto.org/2018/02/22/que-es-la-mineria-de-textos-como-funciona-y-por-que-es-util/, 2018.

[3] Minería de textos - wikipedia, la enciclopedia libr. *Wikipedia*, 1, 2020.

[4] Adei Arias, Alvaro Hernandez, Ander Prieto, Kerman Sanjuan. Covid-19-tweet-sentiment-analysis. https://github.com/Kerman-Sanjuan/Covid-19-Tweet-Sentiment-Analysis, 2021. [Online; accessed 16-October-2021].

[5] Aman Miglani. Coronavirus tweets nlp - text classification. *Kaggle*, 1, 2020.

[6] Kummar Sudaralaj. Getting started with text preprocessing. *Kaggle*, 2020.

[7] Unipython. CÓmo preparar datos de texto con scikit-learn. *Unipython*, 2020.

[8] Gidi Shperber. Quick introduction to bag-of-words (bow) and tf-idf for creating features from text. https://medium.com/wisio/a-gentle-introduction-to-doc2vec-db3e8c0cce5e, 2017. [Online; accessed 2-October-2021].

[9] Miguel Cárdenas-Monter. Índice de dunn. https://studylib.es/doc/8111568/indice-de-dunn. [Online; accessed 17-October-2021].