Práctica 1: Aplicaciones Agrupación de textos y vectorización de palabras

Castro Elvira D. [2022710039], Pineda Rugerio N. [20222710240], Castañón Hernández V. J. [2022710020], Galicia Cocoletzi N. [2022710234], Sánchez Zanjuampa M. A. [2022710029], y Nava Mendez E. U. [2021710144].

Profesor: Lauro Reyes Cocoletzi

***Resumen*—La agrupación de las palabras para la clasificación de los textos es uno de las aplicaciones más importante dentro del lenguaje natural, esta clasificación de textos se puede llevar a cabo mediante diferentes técnicas como aprendizaje maquina, redes neuronales y aprendizaje profundo. La tecnología que se emplea para la clasificación del texto dependerá de diversos aspectos a considerar como el tipo de dataset o el tipo de clasificación que se requiere levar acabo.**

***Palabras clave*—Knn, Naive Bayes,Dataset, Texto, Vector, Características, Gemsim, Matriz de ocurrencia**

I. Introducción

I-A. Marco teórico

I-A1. ¿Qué es el algoritmo Knn?:

El algoritmo de K vecinos más cercanos (K-nearest neighbors) es un clasificador de aprendizaje supervisado no parametrizado, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual. Para ello se asigna la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado.

I-A2. Algoritmo de Naive Bayes:

Los algoritmo de clasificación de Naive Bayes se basan en una técnica de clasificación estadística llamada "Teorema de Bayes". En ellos se asume que las variables predictoras son independientes entre sí. Proporcionan una manera fácil de construir modelos con un comportamiento muy bueno debido a su simplicidad. Lo consiguen proporcionando una forma de calcular la probabilidad 'posterior' de que ocurra un cierto evento A, dadas algunas probabilidades de eventos 'anteriores'.

I-A3. PCA. Análisis de componentes principales :

Se trata de un método estadístico cuya utilidad radica en la reducción de la dimensionalidad del dataset que se esté utilizando. Esta técnica se utiliza cuando se requiere, simplificar la base de datos, ya sea para elegir un menor número de predictores para pronosticar una variable objetivo, o para comprender un dataset de una forma más simple.

I-A4. Word2Vec:

Es una técnica que hace uso de un modelo de red neuronal que permite el análisis de texto mediante la conversión de palabras en vectores numéricos. Este principio se denomina incrustación de palabras y es una manera de representar matemáticamente un texto, de esta manera es posible aprender asociaciones de palabras para sugerir palabras adicionales para terminar una frase. Cabe aclarar que word2vec no permite una comprensión completa del texto por si solo, sino que requiere su funcionamiento en conjunto con otras técnicas de procesamiento de texto. Esta técnica se centra en la semántica y relaciones entre palabras, las cuales puede detectar mediante un procedimiento de aprendizaje supervisado, basándose en que las palabras similares se utilizan con frecuencia en un contexto parecido y después hace hace una afirmación probabilística basada en los datos de entrenamiento.

I-A5. Bolsa de palabras:

Las redes neuronales utilizadas se centran en las relaciones sintácticas de las palabras de las palabras que extrae de un grupo de palabras de entrada. Considera las palabras sin importar el orden. En este modelo cada documento parece una bolsa que contiene algunas palabras. De esta forma permite un modelados de las palabras basado en diccionarios donde cada bolsa contiene unas cuantas palabras del diccionario. Dado que tanto el entrenamiento usa la sintaxis circundante como información básica, la salida esta relacionada sintácticamente con la palabra principal si no coincide con ella.

I-A6. Modelo Skip Gram:

Con Skip Gram producimos palabras de contexto con las que existe relación semántica para una sola palabra de entrada. El modelo utiliza la palabra actual para pronosticar la ventana de palabras de contexto que la rodean. Aquí hace que las palabras de contexto cercano pesen mas que las palabras del contexto mas lejano.

I-A7. Stopwords:

También conocidas como palabras vacías, se refieren a palabras que no aportan un valor o significado al contexto de un texto.

I-B. Objetivos

- Estudiar e implementar técnicas como word2vec.
- Eliminar stopwords y símbolos de un texto.

- Utilizar algoritmos de clasificación como MNB o KNN para hacer un clasificador.

I-C. Aporte

- Realizar un preprocesamiento del texto a fin de detectar y eliminar palabras o símbolos que no aporten al significado del texto
- Hacer uso de técnicas como word2vec para vectorizar aquellas palabras consideradas como representativas de un texto, analizando los resultados obtenidos.
- Usar distintos algoritmos de clasificación como Knn y Naives Bayes.
- Usar funciones como la función gemsin y la matriz de concurrencia

II. Desarrollo

II-A. Vectorización utilizando gemsim y una matriz de ocurrencia.

La vectorización es un proceso que se lleva a cabo al momento de trabajar con archivos de texto, esto con el fin de agilizar el procesamiento gracias a una preparación de los datos antes de ser metidos en algoritmos como bag of words (bolsa de palabras) o un one hot encoding.

Estos procesos de vectorización consisten en una representación más adecuada para el tipo de procesamiento que se desea obtener, por lo que todas las palabras deben ser valoradas dependiendo de una relación de frecuencia de aparición y con esto poder representar la importancia de una palabra en valores cuantificables.

II-A1. Normalización del texto (eliminación de símbolos y stopwords):

Dentro de un archivo de texto se encontrará información que no se desea considerar al momento de realizar una vectorización, este tipo de información van desde palabras que no aportan nada al contenido del texto, es el caso de las famosas "stopwords", también conocidas como "palabras vacías", al igual que símbolos utilizados para mejorar la comprensión del texto, que, aunque el uso de estos caracteres agiliza la lectura al momento de comprender el texto, para la máquina no es realmente importante conocer todas estas reglas gramaticales, por lo que tampoco son útiles en este proceso de vectorizar.

Además de eliminar aquellas palabras vacías, tales como conectores, o símbolos que ayudan a tener una lectura más comprensible, también es útil hacer una normalización con el fin de tener un formato lo más constante posible y que no le cueste tanto a la máquina poder procesar, es por eso que a este paso se agregan un par de características más las cuales nos ayudan a la correcta normalización del texto, tal como eliminar caracteres especiales propios del idioma (español), como una normalización de letras para trabajar solo con letras minúsculas.

II-A2. Vectorización utilizando Gensim:

Gensim es una librería que esta disponible en Python, la cual ayuda a representar documentos como vectores, por lo que ya hace el trabajo automáticamente. además es una herramienta bastante potente la cual es capaz de identificar el tema principal de cualquier documento haciendo uso de la representación de las palabras dentro de un documento para asignar valores a cada una de las palabras y les asigna una ponderación que representa la importancia de la palabra dentro de todo el texto. Además realiza otros procesos dentro de todas sus funcionalidades tales como la tokenización, la normalización del texto (eliminación de stopwords, normalizar mayúsculas) y buscar relaciones dentro de las palabras para poder identificar algún tema del cual se hable y agilizar búsquedas.

II-A3. Vectorización utilizando matriz de co-ocurrencia:

Por otro lado tenemos otro proceso el cual nos indica una manera más de vectorizar un documento, en donde se hace uso de una matriz llena de valores los cuales nos representan la frecuencia de aparición de una palabra dentro de un corpus o de un documento en general. Esta vectorización se normaliza en valores los cuales puedan representar la importancia de una palabra y con este proceso podemos también identificar el tema principal o un tema en común dentro de un conjunto de archivos de texto. La interpretación de la información que nos arroja esta matriz de co-ocurrencia nos ayuda a procesos como la clasificación de los textos, esto pensando en que se pueden crear clases dentro de un conjunto de documentos, lo cual se conoce como corpus y de ahí agrupar a elementos con características similares, en este caso se trata de aquellos textos los cuales tengan una aparición de palabras similares dentro de su contenido.

II-A4. Normalización de texto (pre-procesamiento):

Eliminación de información irrelevante. En el procesamiento del texto existe un paso previo el cual consiste en "limpiar.[el] texto para que cuando sea el momento de procesar la información, no existan inconvenientes dependiendo del idioma en el que se encuentre el documento a tratar. La eliminación de caracteres especiales y de stopwords (conocidas como palabras vacías) es un proceso que consiste en limpiar todo el texto de aquella información la cual no representa el tema del documento de manera adecuada, esto porque estas palabras vacías están conformadas principalmente por el uso de conectores los cuales ayudan a agilizar la lectura. Mientras que los símbolos o caracteres especiales nos ayudan a tener un control sobre los temas tratados dentro del documento, pero estos no representan información dentro del mismo y es necesario eliminarlos por completo para que esta información no aparezca dentro de los algoritmos como la matriz de co-ocurrencia y la herramienta de Gensim

II-B. Clasificación de reseñas de películas como positivas o negativas.

II-B1. Algoritmo 1. Clasificación por Naive Bayes:

En el primer programa, se lleva a cabo una predicción de reseñas positivas y negativas utilizando el algoritmo de Naive Bayes. Primero, se seleccionan los parámetros de texto y etiqueta del dataset, dividiéndolos en dos grupos para entrenamiento y predicción en una proporción de 20 % y 80 %, respectivamente. Luego, se obtienen los vectores de características numéricas de TF-IDF de los textos, limitando su valor a un máximo de 5000 características.

Estos datos se introducen en el modelo de clasificación de Naive Bayes para ajustarlo según los datos de entrenamiento. Posteriormente, se realizan predicciones del modelo utilizando el 20 % de los datos iniciales. Finalmente, se genera un informe de la clasificación de los datos en función de las etiquetas.

II-B2. Algoritmo 2 - Clasificación con KNN:

En el segundo programa, se utiliza el algoritmo de clasificación de KNN (K-Nearest Neighbors) para clasificar las reseñas de películas como buenas o malas.

El archivo se divide en dos, una parte para el entrenamiento del modelo y otra parte para las pruebas. En este caso se utiliza validación cruzada utilizando la proporción 80/20, en donde el 80 % de los datos se utiliza para entrenamiento y el 20 % restante se utiliza para las pruebas.

Procedemos a asignar un valor de k = 5 y mediante la función KNeighborsClassifier de la librería sklearn y, posteriormente se ajustan los datos de entrenamiento y prueba a la partición que se realizó previamente.

Para una mejor comprensión de los resultados, se aplica una reducción de la dimensionalidad a los datos utilizando PCA para poder mostrar en una gráfica 2D los resultados obtenidos después del entrenamiento del modelo.

II-B3. Mejoras a los algoritmo.:

Para lograr un mejor funcionamiento de los algoritmos se plantean las siguientes soluciones:

1. Para ambos algoritmos, hacer un preprocesamiento de los textos de las reseñas, normalizando el texto, de modo que se realice la conversión a minúsculas, la eliminación de caracteres especiales y números, la tokenización, la eliminación de stop-words, la lematización y el stemming. Para ello se crea una función como la siguiente:

Algorithm 1 Preprocesamiento de Texto

Input: Texto original $text$
Output: Texto preprocesado $preprocessed\_text$
$text \leftarrow$ convertir a minúsculas$(text)$
$text \leftarrow$ eliminar caracteres especiales y números$(text)$
$tokens \leftarrow$ tokenizar$(text)$
$tokens \leftarrow$ eliminar stop words$(tokens, stop_words)$
$tokens \leftarrow$ lematizar$(tokens, lemmatizer)$
$tokens \leftarrow$ stemming$(tokens, stemmer)$
$preprocessed\_text \leftarrow$ unir tokens$(tokens)$
return $preprocessed\_text$

2. Para el algoritmo que utiliza la vectorización de palabras mediante TF-IDF y Naive Bayes como clasificador, se prueban con distintos parámetros del tamaño máximo del vocabulario y la frecuencia mínima de documento en el que debe estar presente una palabra. Con esta búsqueda se realiza nuevamente la vectorización y se prueba el modelo.
   El número total de características obtenidas con TfidfVectorizer es de 36,671, y en el código original únicamente se limita a 5000, por lo que se probará con distintos tamaños de características, que sea un número suficiente, pero no demasiado grande para que no afecte en el tiempo de entrenamiento del modelo. Por ello, se probará con un tamaño máximo de vocabulario de 1000, 5000 y 10000, el valor que obtenga una mejor precisión al ser evaluado, será el que se utilice para entrenar con Naive Bayes.
   Se realizará lo siguiente:

Algorithm 2 Búsqueda en Cuadrícula para Ajuste de Hiperparámetros

Input: X_{train}: Conjunto de entrenamiento, y_{train}: Etiquetas de entrenamiento
Output: Mejores parámetros encontrados
$algorithms \leftarrow$ Algoritmos([('tfidf', TfidfVectorizer()), ('clasificador', MultinomialNB())])

$parameters \leftarrow$ {'tfidf__max_features': [1000, 5000, 10000], 'tfidf__min_df': [1, 2, 5],}

$grid\_search \leftarrow$ GridSearchCV(algorithms, parameters, cv=5, n_jobs=-1)

$grid\_search$.fit(X_{train}, y_{train})

$best\_params \leftarrow grid\_search$.best_params

3. Para el algoritmo que utiliza la vectorización de palabras mediante TF-IDF y Knn como clasificador, se utiliza una validación cruzada similar, pero en los parámetros de los algoritmos solo se cambia el clasificador de Naive Bayes a Knn.
4. Para el algoritmo que utiliza Knn como clasificador también se prpone una búsqueda de los mejores valores

de *k*, mediante validación cruzada, de modo que se tome el valor de *k* que mejore el accuracy.
Se realizará algo como lo siguiente:

Algorithm 3 Búsqueda del Mejor Valor de k

Data: X_{train}: Conjunto de entrenamiento, y_{train}: Etiquetas de entrenamiento, k_values: Lista de valores de k
Result: Mejor valor de k
$cv\_scores \leftarrow []$
for k *in* k_values **do**
  $knn \leftarrow$ KNeighborsClassifier(n$_neighbors$ =k) scores $\leftarrow$ cross_val_score(knn, X_{train}, y_{train}, cv=5, scoring='accuracy') cv_scores.append(mean($scores$))
end
$best\_k \leftarrow k\_values[cv\_scores.\text{index}(\text{máx}(cv\_scores))]$

III. Resultados

III-A. Vectorización utilizando gemsim y una matriz de concurrencia.

III-A1. Vectorización utilizando gemsim:

Partiendo del texto limpio (sin simbolos ni stop words), realizamos la tokenizacion de cada oracion en palabras y cremos el modelo Word2Vec con los siguientes parámetros:

- **Tamaño del Vector**: Este parámetro define la dimensionalidad de los vectores de palabra generados por el modelo. En este caso, se ha establecido en 32, lo que significa que cada palabra se representa como un vector de 32 dimensiones en el espacio vectorial.
- **Ventana**: Este parámetro especifica la cantidad máxima de palabras que el modelo considerará en contexto, tanto antes como después de la palabra actual, durante el entrenamiento. En este caso, se ha fijado en 10, lo que significa que el modelo considerará hasta 10 palabras a la izquierda y 10 palabras a la derecha de la palabra objetivo dentro de una ventana de contexto.
- **Frecuencia Mínima**: Este parámetro determina la frecuencia mínima que una palabra debe tener en el corpus de texto para ser considerada durante el entrenamiento. Aquí, se ha establecido en 1, lo que significa que incluso las palabras que aparecen solo una vez en el corpus serán incluidas en el vocabulario y consideradas por el modelo.
- **workers**: Este parámetro especifica la cantidad de subprocesos utilizados para entrenar el modelo. Al asignar 4 trabajadores, el proceso de entrenamiento se paraleliza para acelerar el tiempo de entrenamiento en sistemas con múltiples núcleos de CPU.

Para entrenar el modelo se uso:

- **sentences**: Este parámetro representa la lista de oraciones o textos utilizados para entrenar el modelo. Durante el entrenamiento, el modelo aprende a representar las palabras en vectores basados en el contexto proporcionado por estas oraciones.
- **total examples**: Este parámetro indica el número total de oraciones presentes en la lista 'sentences'. En este caso, se ha establecido en 1, lo que significa que se considera que hay una única oración en el conjunto de datos de entrenamiento.
- **epochs**: Este parámetro especifica la cantidad de iteraciones completas sobre el conjunto de datos de entrenamiento. Aquí, se ha configurado en 100, lo que significa que el modelo realizará 100 pasadas completas sobre la única oración proporcionada para el entrenamiento.

Una vez creado y entrenado nuestro modelo, comprobamos las palabras similares:

Tabla I
MAS SIMILAR A: **PEATÓN**

Palabra	Porcentaje
solicitar	0.857
diaria	0.845
reformada	0.841
municipio	0.838
boleta	0.837
derecha	0.833
fracciones	0.833
para	0.832

Tabla II
MAS SIMILAR A: **CARRETERA**

Palabra	Porcentaje
capítulo	0.919
caso	0.918
presentar	0.918
peligro	0.915
atrás	0.915
extremo	0.914
superficie	0.914
conductor	0.913

III-A2. Vectorización utilizando matriz de concurrencia:

Tabla III
MAS SIMILAR A: **PEATÓN**

Palabra	Porcentaje
disfrutará	0.666
cediendo	0.625
pasar	0.603
cederles	0.579
Ceder	0.574
cederá	0.561
peatones	0.559
marcada	0.557

Tabla IV
MAS SIMILAR A: **CARRETERA**

Palabra	Porcentaje
demarcados	0.666
rayas	0.632
sentido	0.628
valores	0.588
estacionamientos	0.510
tomas	0.471
señalización	0.457
bancarios	0.385

III-B. Clasificación de reseñas de películas como positivas o negativas.

III-B1. Algoritmo 1. Clasificación por Naive Bayes.:

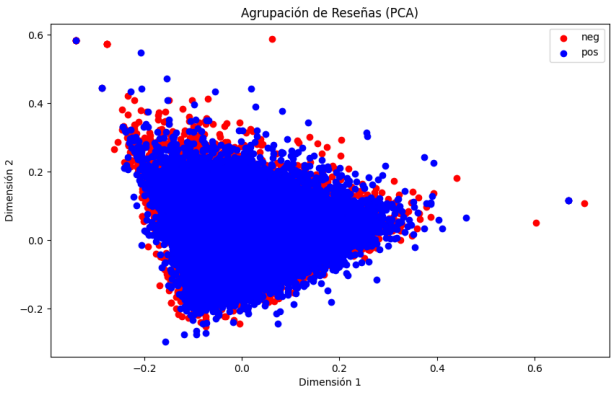
Con los datos preliminares se obtuvieron los siguientes resultados:

Precision del modelo: 0.669

Tabla V
INFORME DE CLASIFICACIÓN.

	precision	recall	f1-score	support
neg	0.67	0.65	0.66	6371
pos	0.67	0.69	0.68	6573
accuracy			0.67	12944
macro avg	0.67	0.67	0.67	12944
weighted avg	0.67	0.67	0.67	12944

III-B2. Algoritmo 2. Clasificación con KNN.:



Visualización de datos del algoritmo KNN después de aplicar PCA

Precision del modelo: 0.52

III-B3. Mejoras a los algoritmos.:

Las implementaciones mencionadas anteriormente se aplicaron a los dos códigos proporcionados para observar su resultado.

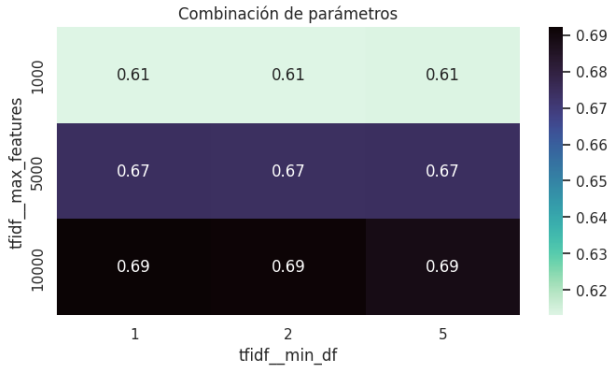
1. **Primer código:** Se realizó el preprocesamiento de las reseñas, estandarizando el texto, y se aplicó la validación cruzada de los distintos valores de tamaño máximo del vocabulario y la frecuencia mínima de documento. en Tfidfvectorizer. Después del preprocesamiento, se obtuvieron los siguientes resultados:

Precision del modelo: 0.67

Tabla VI
INFORME DE CLASIFICACION.

	precision	recall	f1-score	support
neg	0.68	0.65	0.66	6371
pos	0.67	0.70	0.69	6573
accuracy			0.68	12944
macro avg	0.68	0.68	0.68	12944
weighted avg	0.68	0.68	0.68	12944

Sin embargo, al no tener un cambio tan significante, se aplicó validación cruzada para los parámetros de vectorización con TF-IDF. Los resultados de la validación para el tamaño máximo de vocabulario (*tfidf_max_features*) y la frecuencia mínima de documento (*tfidf_min_df*), son los siguientes:



Validación cruzada para parámetros de Tfidfvectorizer.

Como se observa en la gráfica, la mejor combinación de parámetros es de 10,000 el tamaño máximo del vocabulario, y de 1 aparición mínima en el documento. Por lo que se realizaron las pruebas nuevamente con Naive Bayes y se obtuvo lo siguiente:

Precision del modelo: 0.70

Tabla VII
INFORME DE CLASIFICACION.

	precision	recall	f1-score	support
neg	0.70	0.68	0.69	6371
pos	0.70	0.72	0.71	6573
accuracy			0.70	12944
macro avg	0.70	0.70	0.70	12944
weighted avg	0.70	0.70	0.70	12944

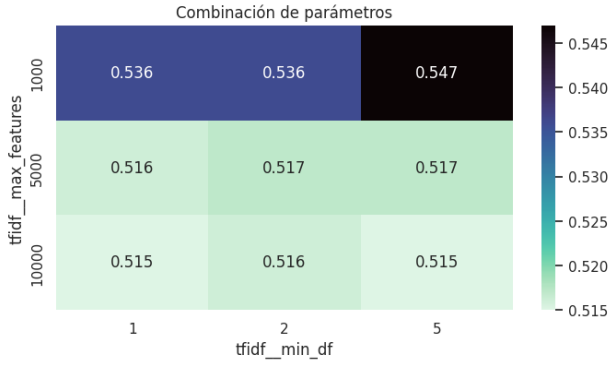
2. **Segundo código:** Tal como el primer caso, se realizó el preprocesamiento de las reseñas, estandarizando el texto, y se aplicó la validación cruzada para los parámetros de Tfidfvectorizer. Por último, también se hizo una validación cruzada, la búsqueda del mejor valor de k. Al aplicar el preprocesamiento de datos se obtuvo lo siguiente:

Precision del modelo: 0.522

Tabla VIII
INFORME DE CLASIFICACION.

	precision	recall	f1-score	support
neg	0.52	0.89	0.64	6371
pos	0.59	0.16	0.25	6573
accuracy			0.52	12944
macro avg	0.55	0.52	0.45	12944
weighted avg	0.55	0.52	0.44	12944

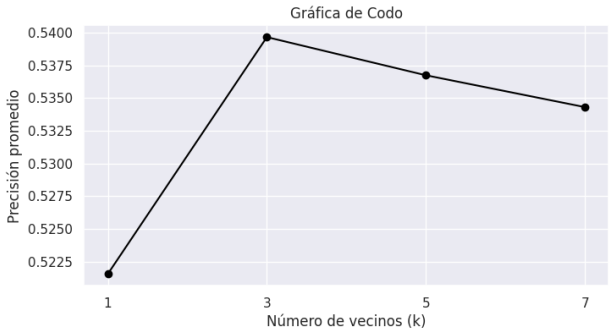
La validación cruzada de Tfidfvectorizer arrojó lo siguiente:



Validación cruzada para parámetros de Tfidfvectorizer.

Como se observa, la mejor combinación de parámetros es de un tamaño de vocabulario de 1000 y una frecuencia de documento de 3.

Y finalmente, con la validación cruzada de Knn, se obtuvo un valor de $k = 3$.



Gráfica de codo, para obtener el mejor valor *k*.

Combinando todas estas mejoras se obtuvo el siguiente reporte:

Precision del modelo: 0.547

Tabla IX
INFORME DE CLASIFICACION.

	precision	recall	f1-score	support
neg	0.55	0.42	0.48	6371
pos	0.54	0.67	0.60	6573
accuracy			0.55	12944
macro avg	0.55	0.55	0.54	12944
weighted avg	0.55	0.55	0.54	12944

IV. CONCLUSIONES

La comparación entre el enfoque de vectorización utilizando Gensim y la matriz de concurrencia revela diferencias significativas en los resultados obtenidos. Mientras que Gensim sugiere asociaciones más abstractas o menos precisas, como 'solicitar', 'diaria' o 'reformada' para la palabra 'peatón', la matriz de concurrencia proporciona términos más concretos y directamente relacionados, como 'disfrutara', 'cediendo' y 'pasar'.

Esta disparidad se evidencia aún más con la palabra 'carretera', donde Gensim sugiere términos como 'capítulo', 'caso' y 'presentar', que parecen menos pertinentes, mientras que la matriz de concurrencia arroja palabras como 'desmarcados', 'rayas' y 'sentido', que están más alineadas con el contexto buscado.

Estas discrepancias pueden atribuirse al entrenamiento realizado en Gensim, que puede ser insuficiente o carecer de datos relevantes para capturar las asociaciones específicas que busca el análisis. Sin embargo, esto se compensa con el tiempo de ejecución, donde la función de Gensim demuestra una ejecución más rápida (0.4 segundos) en comparación con la matriz de concurrencia (58.6 segundos).

En cuanto a la vectorización y clasificación de reseñas de películas, se observó que es posible realizar predicciones de textos considerados positivos y negativos, en función de las palabras que contienen. En primera instancia, se consideraron todos los datos en bruto para obtener un corpus completo. Estos datos fueron vectorizados mediante TF-IDF, para luego entrenar un modelo de Naïve Bayes y KNN. Sin embargo, se decidió realizar un preprocesamiento de los datos con el fin de eliminar palabras que no se consideran importantes para la clasificación. De esta manera, se logró mejorar el porcentaje de clasificación. Si bien la mejora no fue la esperada, permitió comprender la importancia de un preprocesamiento de los datos, que, acompañado de un modelo de clasificación más robusto, permitiría incrementar de forma considerable el porcentaje de clasificación.

REFERENCIAS

[1] Vectorización de Textos https://ucd-dnp.github.io/ConTexto/versiones/master/ejemplos/07_vectorizacion_de_textos.html. Recuperado el 10 de marzo de 2024
[2] Word2vec: Análisis de textos mediante incrustación de palabras https://konfuzio.com/es/wordtovec/. Recuperado el 10 de marzo de 2024
[3] Languaje Processing and Python https://www.nltk.org/book/ch01.html. Recuperado el 10 de marzo de 2024
[4] N-grama https://es.wikipedia.org/wiki/N-grama. Recuperado el 10 de marzo de 2024
[5] Algoritmos Naive Bayes: Fundamentos de implementación https://medium.com/datos-y-ciencia/algoritmos-naive-bayes-fudamentos-e-implementaci%C3%B3n-4bcb24b307f. Recuperado el 10 de marzo de 2024
[6] Embedding Techniques on Text Data using KNN https://www.analyticsvidhya.com/blog/2022/01/embedding-techniques-on-text-data-using-knn/. Recuperado el 11 de marzo de 2024