Análisis de clasificación de documentos mediante modelo word2vec

Daniel Gallardo Martos
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo de contacto: dangalmar@alum.us.es UVUS: dangalmar

Fernando José Mateos Gómez
dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
Correo de contacto: fermatgom@alum.us.es UVUS: fermatgom

En este trabajo de investigación, se aborda el problema de la clasificación de géneros de películas a partir de sus sinopsis utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático. Se destaca la importancia de esta tarea en la industria cinematográfica, ya que permite organizar y etiquetar eficientemente el contenido audiovisual, facilitando la búsqueda y recomendación de películas a los usuarios. Se propone el uso de modelos basados en Word2Vec, una técnica para el aprendizaje de representaciones vectoriales de palabras, para capturar el significado y la semántica de las palabras en un espacio de dimensionalidad reducida. Se emplean las arquitecturas Skip-gram y CBOW de Word2Vec para entrenar modelos de representación de palabras, y se utiliza el algoritmo de optimización de descenso de gradiente estocástico para ajustar los parámetros de la red neuronal. Además, se realiza la preparación del corpus de texto, incluyendo la construcción del corpus inicial, la limpieza de datos, el proceso de stemming y el filtrado de palabras por su frecuencia en el corpus. Finalmente, se entrena y evalúa el modelo Word2vec utilizando clasificadores como SVM y Random Forest y el modelo de Naive Bayes Multinomial.

Palabras Clave—Inteligencia Artificial, PLN, Word2Vec, Bolsa de Palabras, recall, accuracy, f1-score

I. Introducción al problema y aplicaciones

El procesamiento de lenguaje natural ha experimentado un crecimiento significativo en las últimas décadas, abriendo nuevas oportunidades para el análisis y comprensión de grandes volúmenes de texto. En este trabajo, nos centraremos en el problema de clasificación de documentos y su aplicación específica en la categorización de géneros de películas a partir de sus sinopsis.

El objetivo principal de esta investigación es desarrollar y analizar modelos capaces de predecir con precisión los géneros a los que pertenecen las películas utilizando técnicas de procesamiento de lenguaje natural y aprendizaje automático. Esta tarea reviste una gran importancia en la industria cinematográfica, ya que permite organizar y etiquetar de manera eficiente el contenido audiovisual, facilitando la búsqueda y recomendación de películas a los usuarios.

El enfoque propuesto en este trabajo se basa en el uso de técnicas de vectorización de texto y clasificación automática. Para lograrlo, se emplean modelos basados en Word2vec y se analizan posteriormente comparando sus variantes entre sí y el modelo de Naive Bayes multinomial para determinar qué modelo es mejor.

II. funcionamiento de word2vec

Word2Vec es una técnica popular para el aprendizaje de representaciones vectoriales de palabras, también conocidas como embeddings. Fue desarrollado por Tomas Mikolov y su equipo en Google en 2013. El objetivo principal de Word2Vec es capturar el significado y la semántica de las palabras mediante la representación de las mismas como vectores numéricos densos en un espacio de dimensionalidad reducida.

El funcionamiento de Word2Vec se basa en dos arquitecturas principales: Skip-gram y Continuous Bag-of-Words (CBOW). Estas arquitecturas utilizan una red neuronal poco profunda para entrenar modelos de representación de palabras.

En la arquitectura Skip-gram, el objetivo es predecir las palabras de contexto a partir de una palabra objetivo. El modelo recibe como entrada una palabra objetivo y genera vectores de contexto para las palabras que suelen aparecer cerca de la palabra objetivo en un contexto determinado. Por ejemplo, dada la oración "El gato atrapó al ratón", el modelo intentaría predecir "El", "gato", "atrapó" y "ratón" a partir de la palabra objetivo "al". El proceso se repite para múltiples pares de palabras objetivo y contexto en el corpus de entrenamiento.

En la arquitectura CBOW, el enfoque es inverso al de Skip-gram. En lugar de predecir el contexto a partir de una palabra objetivo, el modelo intenta predecir la palabra objetivo dada una ventana de palabras de contexto. Es decir, dado un contexto como "El gato atrapó al __", el modelo intentaría predecir la palabra objetivo "ratón". Nuevamente, se repite este proceso para varias instancias de contexto y objetivo en el corpus de entrenamiento.

Ambas arquitecturas utilizan una capa oculta con una representación de palabras (embedding) que se aprende durante el entrenamiento. Estos embeddings capturan información sobre la similitud y las relaciones semánticas entre las palabras.

Para entrenar el modelo Word2Vec, se utiliza el algoritmo de optimización de descenso de gradiente estocástico (SGD) o variantes más avanzadas como Adam. Durante el entrenamiento, se ajustan los parámetros de la red neuronal para maximizar la probabilidad de predecir correctamente las palabras objetivo o contexto.

III. PREPARACIÓN DEL CORPUS

Esta sección se dedica a la explicación y justificación de cada etapa en el desarrollo de la práctica:

A. Construcción del corpus inicial

Para la creación del corpus se utilizó una API (Application Programming Interface) para obtener un conjunto de datos de sinopsis y géneros de películas. La API proporciona acceso a una base de datos amplia y actualizada que contenía información detallada sobre películas de diferentes géneros.

Tras consumir la API, se accedió a las sinopsis y géneros de las películas en formato json. Después, se procedió a extraer y procesar estos datos para crear el corpus necesario para nuestro análisis y clasificación de documentos. Esto se realizó creando una función que se aloja en el fichero ***createCorpus*** y que recopila el corpus inicial en el fichero *data.txt* de la carpeta data del proyecto.

B. Limpieza de datos del corpus

La importancia de la limpieza del corpus radica en mejorar la calidad de los datos y reducir el ruido. Al limpiar el corpus, se eliminan elementos irrelevantes como símbolos de puntuación, números y caracteres especiales. Además, se eliminan palabras vacías que no aportan información significativa al análisis. Esta limpieza contribuye a obtener resultados más precisos y significativos en el análisis de texto y la clasificación de documentos al reducir la cantidad de ruido presente en los datos. Al tener un corpus limpio, se proporciona una representación más clara y coherente de los documentos, lo que mejora el rendimiento del modelo y acelera el procesamiento al reducir la dimensionalidad de los datos.

La limpieza de estos datos se llevó a cabo en la función que se encuentra en el fichero ***textCleaner*** y cuyos pasos se explican a continuación:

1) Tokenización: Es el proceso de dividir un texto en unidades más pequeñas llamadas tokens. Los tokens pueden ser palabras individuales, frases o caracteres. En nuestro caso dividimos las sinopsis en palabras utilizando la función *word_tokenize()* de la librería NLTK [1].

2) Eliminación stopwords: Las stopwords son palabras comunes que se consideran irrelevantes para el análisis de texto debido a su alta frecuencia de aparición en el lenguaje. Ejemplos de stopwords son "el", "la", "y", "en". Al eliminar las stopwords durante el procesamiento de texto, se reduce la dimensionalidad de los datos y se elimina el ruido innecesario. En nuestro caso utilizamos la lista de stopwords (español) que proporciona NLTK.

3) Eliminar mayúsculas y signos de puntuación: La eliminación de mayúsculas es importante en el procesamiento de texto para estandarizar y simplificar los datos. Al convertir todo el texto a minúsculas, se evita la distinción entre palabras escritas en mayúsculas y minúsculas, lo que mejora la coherencia y consistencia en el análisis. Al mismo tiempo, eliminar los signos de puntuación ayuda a reducir el ruido en el texto, ya que estos símbolos no aportan información relevante para muchas tareas de procesamiento de texto, como análisis de sentimiento o clasificación de documentos.

C. Aplicar stemming al corpus

El stemming es un proceso utilizado en el procesamiento de texto para reducir las palabras a su forma base o raíz, eliminando los sufijos y afijos gramaticales. El objetivo del stemming es agrupar palabras relacionadas y reducir la dimensionalidad de los datos.

Al realizar el stemming, se eliminan las terminaciones de las palabras, como plurales, verbos conjugados, tiempos verbales, entre otros. Esto permite que palabras con la misma raíz se traten como una única unidad, lo que simplifica el análisis y mejora la precisión de las tareas de procesamiento de texto, como la clasificación de documentos o la recuperación de información.

Por ejemplo, si tenemos las palabras "corriendo", "corre" y "correrá", el stemming las reduciría todas a la raíz "correr". Esto nos permite tratar todas estas variantes como la misma palabra, lo que simplifica el análisis y reduce la complejidad de los datos.

Es importante tener en cuenta que el stemming no siempre produce resultados perfectos, ya que puede generar palabras truncadas o raíces que no son palabras reales. Sin embargo, en muchos casos, el stemming es una técnica útil y ampliamente utilizada para normalizar el texto y simplificar su procesamiento.

Existen varios algoritmos para aplicar stemming, entre ellos están el algoritmo de Porter, el algoritmo de Snowball y el algoritmo de Lancaster. En nuestro caso elegimos el algoritmo de Snowball utilizando la clase ***SnowballStemmer*** de la librería NLTK, aplicándolo en la función que se encuentra en el fichero ***textStemmer*** y guardando el corpus resultante en el fichero *stemming_data.txt*.

Una vez entrenado, el modelo Word2Vec permite obtener representaciones vectoriales de palabras para cualquier palabra en el vocabulario, incluso para palabras que no están presentes en el corpus de entrenamiento. Estos vectores de palabras se pueden utilizar para diversas tareas de procesamiento de lenguaje natural como la clasificación de documentos utilizando dichos embeddings para entrenar y predecir.

Elegimos este algoritmo debido a que los algoritmos de Porter y Lancaster son más agresivos en la reducción de palabras lo que puede dificultar el desarrollo de la práctica. Además, el algoritmo de Snowball es más preciso y se maneja mejor en distintos idiomas.

D. Filtrar palabras por su frecuencia en el corpus

Filtrar las palabras del corpus por su frecuencia es importante debido a las siguientes razones:

1) Reducción del ruido: Los corpus de texto suelen contener numerosas palabras poco frecuentes o palabras que carecen de relevancia semántica, tales como nombres propios o alias. La filtración por frecuencia permite la eliminación de estas palabras, lo que a su vez reduce el ruido presente en el corpus y permite centrarse en las palabras más pertinentes.

2) Enfoque en palabras clave: La filtración por frecuencia permite identificar las palabras que aparecen con mayor frecuencia en el corpus. Estas palabras clave resultan útiles para llevar a cabo análisis temáticos, generación de resúmenes automáticos o identificación de los conceptos principales presentes en el corpus.

3) Mejora del rendimiento computacional: Al reducir el tamaño del corpus mediante la eliminación de palabras poco frecuentes, es posible mejorar el rendimiento computacional en tareas posteriores, tales como el procesamiento del lenguaje natural (PLN) o el entrenamiento de modelos de aprendizaje automático. Un corpus más reducido y enfocado acelera los algoritmos y disminuye los requerimientos de memoria.

4) Normalización del texto: La filtración por frecuencia también forma parte del proceso de normalización del texto. La eliminación de palabras poco frecuentes o raras permite simplificar el vocabulario del corpus, facilitando así su análisis posterior.

Es importante tener en cuenta que el umbral de frecuencia utilizado para filtrar palabras puede variar dependiendo del contexto y del objetivo específico del análisis. En algunos casos, puede resultar necesario conservar palabras menos frecuentes si estas son relevantes para la tarea en cuestión.

En nuestro caso, el filtrado de palabras se realizó con la función corpus en el fichero trainingModel. Creamos un fichero *corpus_data.txt* el cual guarda el corpus final que usaremos para el entrenamiento y pruebas, usando un umbral de una frecuencia 5 palabras por defecto, para así eliminar palabras no deseadas del corpus(nombres, alias, palabras que no aporten información porque aparezcan pocas veces, etc…)

IV. ENTRENAMIENTO

Para realizar el entrenamiento de los modelos usamos las librerías *gensim* y *SKLearn*. Comenzaremos a explicar como fue el entrenamiento de los modelos basados en Word2Vec.

A. Word2Vec

Accediendo al fichero de **trainingModel** tenemos 2 secciones separadas, una para el propio entrenamiento y otra para las predicciones.

Tras hacer este filtrado inicial se llama a la función *training*, la cual recibiendo el tipo de modelo y el fichero que usará para crear su vocabulario, será capaz de crear un modelo con *gensim* con el constructor **Word2Vec**. Cabe mencionar que dentro de los parámetros relevantes, los argumentos de “epochs”, “seed” y “sentences” son los más importantes (siendo los dos primeros aquellos que facilitarán el aprendizaje del modelo, y el último crear el propio vocabulario. No aprovechamos intencionadamente la capacidad para delimitar un umbral ya que trabaja con porcentajes, y nuestro corpus podría ser alterado en el tiempo, por lo que usamos un valor fijo de ocurrencias). Además seleccionamos el tipo de modelo usando el argumento “sg”, para indicar si queremos usar Skip Gram o Bolsa de Palabras

Posteriormente debemos de crear un vector de palabras, con los pesos de cada una de las palabras de nuestro vocabulario, para esto creamos una función **vectorization** la cual crea un array bidimensional, que guarda las características de cada una de las palabras de cada uno de los ejemplos del corpus, para luego hacer una media y tener arrays uniformes y unidimensionales (evitando así problemas debido a que hay ejemplos de mayor tamaño que otros).

Para que el modelo sea capaz de analizar nuevas entradas, fue necesario crear una función **categorize** la cual asigna una alias a cada uno de los géneros existentes en nuestro corpus.

Finalmente, debido a las limitaciones de gensim, usamos la librería de SKLearn para crear un flujo con *make_pipeline*, tal que será capaz de analizar cada uno de nuestros vectores de palabras, usando una estandarización con *StandardScaler* y varios clasificadores que son explicados a continuación:

1) Support Vector Machines (SVM): Es un algoritmo de aprendizaje supervisado utilizado para clasificación y regresión. Busca encontrar un hiperplano que mejor separe las instancias de diferentes clases en un espacio de alta dimensionalidad. Puede utilizar diferentes funciones de kernel para mapear los datos a espacios de mayor dimensión. Busca maximizar el margen entre las clases y clasificar nuevas instancias en base a su ubicación relativa al hiperplano de separación.

2) Random Forest: Es un algoritmo de aprendizaje supervisado utilizado para problemas de

clasificación y regresión. Combina múltiples árboles de decisión individuales para realizar predicciones. Cada árbol se entrena con un subconjunto aleatorio de características y datos de entrenamiento. Las predicciones finales se obtienen mediante la votación o promedio de las predicciones de los árboles individuales.

3) Gradient Boosting: Es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación y regresión. Construye un modelo de ensamblado de árboles de decisión en forma secuencial. Cada árbol se entrena para corregir los errores del modelo anterior. Se centra en las instancias con errores más altos en cada iteración para mejorar la precisión general del modelo. Las predicciones finales se obtienen mediante la suma ponderada de las predicciones de todos los árboles.

4) Logistic Regression: Es un algoritmo de aprendizaje supervisado utilizado principalmente para problemas de clasificación binaria. Utiliza la función logística (también conocida como sigmoide) para modelar la probabilidad de que una instancia pertenezca a una determinada clase. Durante el entrenamiento, se ajustan los parámetros del modelo para maximizar la verosimilitud de los datos de entrenamiento. Durante la predicción, se utiliza el umbral de probabilidad para asignar las instancias a una clase.

B. Naive Bayes Multinomial

Es un algoritmo de aprendizaje supervisado utilizado para problemas de clasificación. Se basa en el teorema de Bayes y asume independencia condicional entre las características. Es especialmente adecuado para datos con representación discreta, como conteos de palabras. Estima la probabilidad de que una instancia pertenezca a una clase utilizando la frecuencia de las características en el conjunto de entrenamiento.

Para trabajar con este modelo, previamente se crea una categorización, de forma dinámica, usando **categorize**.

Habiendo hecho esto, dividimos el corpus, de tal forma que usamos el 25% para entrenar y el 75% para predecir usando la función **train_test_split** de SKLearn.

Tras esto, vectorizados las palabras del corpus, con **CountVectorizer** de SKLearn, y entrenamos el modelo usando la clase **MultinomialNB**

Para analizar los resultados de los modelos, tomamos un segmento de *data.txt* y es analizado por nuestra función **predict**, la cual, dependiendo del modelo, vectorizará los datos y usará los modelos previamente entrenados para predecir utilizando la función **predict**. Como entrada se introducirán los vectores de pesos de palabras.

V. RESULTADOS

Esta sección se dedica a mostrar los distintos resultados obtenidos al entrenar los modelos. Todo el proceso de entrenamiento y pruebas se encuentran en el fichero ***trainingModel***.

Para el análisis de los resultados se proporciona el accuracy, qué es el porcentaje de acierto del modelo, la matriz de confusión y un reporte de clasificación para cada clase indicando su precisión, su recall, su f1-score y su soporte, es decir, número de muestras de cada clase

A. Word2Vec - Bolsa de Palabras

Tras vectorizar cada palabra del vocabulario, habiéndose filtrado con un umbral para eliminar nombres propios y palabras que aparecen muy pocas veces, se analizan los resultados obtenidos para los distintos clasificadores a continuación:

LogisticRegression (LR):

Obtuvo una precisión global del 79.6% en la clasificación de textos. Al analizar la matriz de confusión, se observa que algunas clases presentan una mayor confusión, como la clase 0 con una precisión baja (47%) y la clase 5 con una precisión perfecta pero un recall bajo (50%). El reporte de clasificación muestra que las clases con mejores resultados son la 6, 7, 8, 9, 10, 11 y 13, con precisión, recall y f1-score perfectos (100%). Sin embargo, algunas clases tienen un f1-score más bajo, como la clase 0 con 0.61 y la clase 5 con 0.67. Estos resultados indican que el modelo tiene dificultades para clasificar algunas clases específicas, lo que puede requerir una mayor exploración y ajuste del modelo para mejorar su rendimiento en esas clases en particular.

Support Vector Machines (SVM):

Obtuvo una precisión global del 71.7% en la clasificación de textos. Al analizar la matriz de confusión, se observa que hay varias clases con resultados variados. Algunas clases tienen una precisión y recall altos, como las clases 5, 6, 7 y 11, con valores perfectos (100%). Sin embargo, otras clases presentan dificultades en la clasificación, como las clases 0, 1, 4, 8, 9, 10, 13 y 16, con valores de precisión, recall y f1-score más bajos. Esto indica que el modelo tiene dificultades para clasificar correctamente estas clases, lo que puede deberse a la falta de información o características distintivas en los textos correspondientes. En general, se puede concluir que el modelo tiene un rendimiento moderado, pero podría beneficiarse de ajustes o mejoras para obtener resultados más precisos y equilibrados en todas las clases.

Random Forest (RF):

Este modelo logró una alta precisión global del 93.8% en la clasificación de textos. Al analizar la matriz de confusión, se observa un rendimiento sobresaliente en la mayoría de las clases. La mayoría de las clases tienen valores de precisión, recall y f1-score cercanos o iguales a 1, lo que indica una clasificación muy precisa y completa. Solo la clase 9 muestra un rendimiento ligeramente inferior, con una precisión, recall y f1-score de 1.0, 0.4 y 0.57 respectivamente. Esto sugiere que el modelo puede tener dificultades para clasificar correctamente la clase 9 debido a

la falta de información o características distintivas en los textos correspondientes. En general, el modelo muestra un rendimiento muy sólido y confiable en la mayoría de las clases, lo que demuestra su capacidad para clasificar eficazmente los textos en diferentes categorías.

Gradient Boosting (GB):

Alcanzó una precisión global del 80.5% en la clasificación de textos. Al examinar la matriz de confusión, se puede observar que el rendimiento del modelo es variable en las diferentes clases. Algunas clases muestran un rendimiento sólido, como las clases 2, 4, 5, 6, 7 y 10, con valores de precisión, recall y f1-score cercanos o iguales a 1. Sin embargo, otras clases tienen un rendimiento más bajo, como las clases 0, 1, 3, 8 y 9, donde se observa una disminución en la precisión, recall y f1-score. Esto sugiere que el modelo puede tener dificultades para clasificar correctamente algunas de estas clases debido a la falta de información o características distintivas en los textos correspondientes. En general, el modelo muestra un rendimiento aceptable, pero puede haber margen de mejora en la clasificación de algunas clases específicas.

B. Word2Vec - Skip Gram

Al igual que "Bolsa de Palabras", se construye usando las mismas premisas, sin embargo debido a cómo funciona este modelo las predicciones cambian ligeramente:

LogisticRegression (LR):

Ha mejorado su rendimiento en comparación con la LR de la arquitectura anterior. Alcanzó una precisión global del 88.5% en la clasificación de textos. La matriz de confusión muestra un rendimiento sólido en la mayoría de las clases, con valores altos de precisión, recall y f1-score. Algunas clases, como la clase 4, muestran un rendimiento perfecto con valores de 1 en todas las métricas. Otras clases también tienen un rendimiento alto, como las clases 2, 5, 6, 7, 10 y 13, con valores cercanos o iguales a 1 en las métricas. Sin embargo, la clase 9 muestra un rendimiento más bajo en comparación con las otras clases, con una precisión del 100% pero un recall y f1-score del 60%. Esto sugiere que el modelo puede tener dificultades para clasificar correctamente la clase 9 y podría ser un área de mejora. En general, el modelo ha mejorado su capacidad de clasificación en comparación con el caso anterior, pero aún puede haber margen para optimizaciones adicionales.

Support Vector Machines (SVM):

Tiene una precisión global del 72.6% en la clasificación de textos lo que mejora un poco el modelo de la arquitectura anterior. La matriz de confusión muestra resultados mixtos en las diferentes clases. Algunas clases, como la clase 5, 6 y 11, muestran un rendimiento perfecto con valores de 1 en todas las métricas. Sin embargo, otras clases tienen un rendimiento más bajo, como las clases 0, 1, 2, 8 y 9, con valores de precisión, recall y f1-score por debajo del 60%. Esto sugiere que el modelo tiene dificultades para clasificar correctamente esas clases y podría ser un área de mejora. Además, algunas clases tienen valores de recall y f1-score de 0, lo que indica que el modelo no pudo identificar correctamente ninguna instancia de esas clases. En general, este modelo no es tan efectivo como los casos anteriores y puede requerir ajustes o mejoras adicionales para mejorar su rendimiento.

Random Forest (RF):

Tiene una precisión global del 98.2% en la clasificación de textos lo que lo hace el mejor modelo para este corpus. La matriz de confusión muestra resultados sobresalientes en todas las clases. Todas las clases tienen valores de precisión, recall y f1-score de 1, lo que indica que el modelo clasificó correctamente todas las instancias de cada clase. Esto sugiere que el modelo ha aprendido eficazmente las características semánticas de los textos y puede realizar una clasificación precisa en este conjunto de datos. En general, este modelo muestra un rendimiento excelente y puede considerarse altamente confiable para la clasificación de textos en este contexto específico.

Gradient Boosting (GB):

Tiene una precisión global del 88.5% en la clasificación de textos. La matriz de confusión muestra que algunas clases tienen un rendimiento más bajo en términos de precisión, recall y f1-score. Por ejemplo, las clases 2, 4 y 5 tienen valores inferiores en estas métricas, lo que indica que el modelo tuvo dificultades para clasificar correctamente algunas instancias en esas clases. Esto podría ser atribuido a la complejidad de los patrones semánticos presentes en los textos de esas clases, lo que podría haber dificultado la captura de todas las características relevantes por parte del modelo. El modelo muestra un rendimiento sólido en general, con una precisión y recall superiores al 75% en la mayoría de las clases.

C. Naive Bayes

El modelo de Naive Bayes Multinomial ha obtenido un accuracy de 0.8761 en la clasificación de tu conjunto de datos. La matriz de confusión revela que el modelo ha logrado una precisión razonablemente alta para la mayoría de las clases, con algunas excepciones. Por ejemplo, las clases 8 y 16 presentan un desempeño deficiente debido a la falta de ejemplos de entrenamiento. En general, el modelo ha logrado resultados sólidos, pero ligeramente inferiores en términos de precisión global en comparación con el modelo de regresión logística (0.8849) y el modelo de árbol de decisión (0.8938).

El modelo de Naive Bayes Multinomial es una buena opción cuando se trabaja con datos de texto, especialmente en problemas de clasificación de palabras o frecuencias. Sin embargo, en este caso particular, parece haber algunas clases con un rendimiento inferior, lo que indica la necesidad de una mayor atención a esos casos.

En comparación con los modelos anteriores, el modelo de Naive Bayes Multinomial obtuvo una precisión global ligeramente inferior, pero aún así logró un rendimiento sólido en términos generales. Es importante considerar el contexto y las características del problema antes de seleccionar un modelo específico, ya que cada uno tiene sus fortalezas y debilidades.

VI. CONCLUSIONES

En general, los modelos que utilizaron Word2Vec - Skip Gram obtuvieron mejores resultados en comparación con los que utilizaron Word2Vec - Bolsa de Palabras. Específicamente, Logistic Regression (LR) y Gradient Boosting (GB) mostraron mejoras notables en su precisión global. Sin embargo, el modelo Random Forest (RF) con Word2Vec - Skip Gram se destacó como el modelo con la mayor precisión global, alcanzando un impresionante 98.2%.

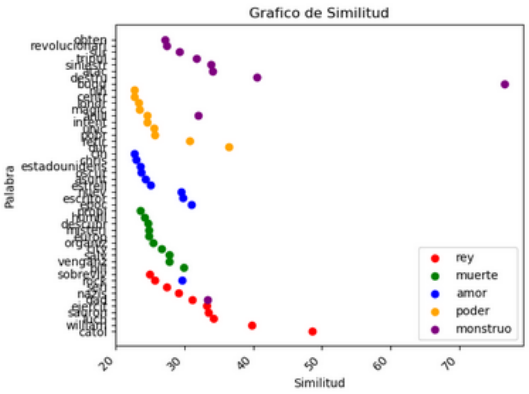
Para comprobar que esto fuera así, aumentamos nuestro corpus de entrenamiento, de esta forma el modelo de Naive Bayes podía tener más datos con los que entrenar, y aumentamos el número de épocas a 500 para que los modelos que usaban Word2Vec pudieran tener una mayor precisión.

Estas decisiones fueron beneficiosas ya que pudimos observar que nuestra conclusiones iniciales fueron correctas, al aumentar tanto los ejemplos como las épocas, observamos un aumento en la tasa de aciertos y la precisión debido a una mayor diversidad de datos.

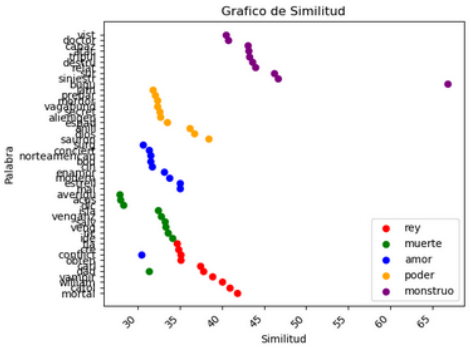
Por lo que podemos concluir que de entre los modelos entrenados, el clasificador Random Forest con Word2vec es el que mejor responde a este corpus.

Para mostrar la relación entre algunas palabras en base a sus embeddings tomamos 5 palabras aleatorias y buscamos su similitud en los modelos que usaba Word2Vec.

A continuación mostramos la comparativa:



Este primer gráfico muestra la similitud de las palabras “rey”, “muerte”, “amor”, “poder” y “monstruo” con Bolsa de Palabras. A partir de haber hecho steeming, tenemos un listado de raíces de palabras que se asemejan en mayor o menor medida a las palabras deseadas pero comprobamos que la gran mayoría no se asemejan.



Para Skip Gram tomamos las mismas palabras y buscamos su similitud. A parte de sacar las mismas conclusiones que en el anterior, podemos decir que la similitud entre las palabras obtenidas, es mayor, entre ellas, que en el modelo anterior.

VII. REFERENCIAS

[1] Librería NLTK: https://www.nltk.org

[2] Word2Vec: https://towardsdatascience.com/word2vec-explained-49c52b4ccb71

[3] Gensim: https://radimrehurek.com/gensim/models/word2vec.html

[4] Datos: https://www.themoviedb.org/