Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas

E-Commerce Text Classification

Avance 2

Chaparro Sicardo Tanibeth, Malváez Flores Axel Daniel

Project #1

2023

Definición del problema

Con el crecimiento exponencial de los datos y de las ventas en línea, lo que proponemos es hacer un clasificador de texto correspondiente a los productos de venta en línea y su descripción. El volumen de información disponible hace que la revisión y clasificación manual de los productos se vuelva extremadamente ineficiente, consumiendo una cantidad significativa de tiempo y recursos. En este caso lo que se desea es poder hacer este tipo de clasificaciones de manera eficiente y correcta.

Objetivos

`Objetivo general`

Se tiene como objetivo desarrollar un clasificador que automatice el proceso de categorización de productos de venta en línea a partir del análisis de su descripción. Se busca determinar si el uso de técnicas avanzadas de Procesamiento de Lenguaje Natural (PLN) logran mejorar el proceso de clasificación (a diferencia de los trabajos que se tienen como "estado del arte" que no realizan un amplio preprocesamiento del texto) al extraer la información relevante de la descripción de los productos. Por otro lado, se desea identificar el modelo de clasificación más eficaz para el problema en cuestión, para esto se realiazará una comparación entre los resultados del modelo usando diferentes métricas de evaluación. Por último se desea mejorar los scores que se tienen en otros trabajos similares a este (arriba del 93%).

`Objetivos especificos`

- Procesamiento del Lenguaje Natural: Aplicación de técnicas de PLN para la extracción de información relevante de las descripciones de los productos. Generación de etiquetas. Extracción de características y vectorización.
- Selección del algoritmo: Elegir el mejor algoritmo de clasificación de textos para nuestra tarea. Los algoritmos con los que se trabajará son Naive Bayes, Support Vector Machines y XGBoost.
- Entrenamiento y evaluación: Utilizar el conjunto de datos para entrenar el modelo seleccionado. Realizar múltiples pruebas con métricas, como **accuracy**, **f1-score** y **confussion matrix**, para evaluar la precisión del modelo y a partir de los resultados realizar los ajustes necesarios al modelo.

Descripción del método propuesto

`Limpieza y Transformacion (Lema y Stem)`

Recordemos que este proyecto utilizará el conjunto de datos obtenido de la plataforma Kaggle [Sha19] que contiene la descripción de productos pertenecientes a las categorías *Electronics*, *Household*, *Books* y *Clothing & Accesories*.

Una vez que ta tenemos el conjunto de datos, procedemos a cargarlo en nuestro Jupyter Notebook para poder realizar una exploración de los mismos. Una vez que dicha exploración se llevara a cabo, se procedió a crear dos data sets distintos. El primero es un data set que contiene la columna *Description* tokenizada y **lematizada** de nuestro texto original, mientras que el segundo data set cuenta con la columna *Description* tokenizada y con **stemming** nuevamente de

nuestro texto original. Ambos con su etiqueta correspondiente a cada documento.

Vectorizacion (Lemma y Stem)

El paso siguiente fue transformar nuestros documentos de texto a vectores de representación numérica de dichos documentos. Para este proceso, en principio se decidió probar tres formas de representar a nuestros datos : dos utilizando modelos embebidos de Word2Vec y el otro una matriz de términos TF-IDF. El primer modelo **word2vec** [Mik+13] que usamos fue uno entrenado por nosotros con nuestro corpus (el conjunto de todos los documentos), este entrenamiento se realiza con un sistema de redes neuronales usando la función **Word2Vec** de **gensim** [Řeh23]. Una vez entrenado obtenemos los vectores correspondientes a cada documento (50424×300), obtenemos una muestra y la visualizamos:

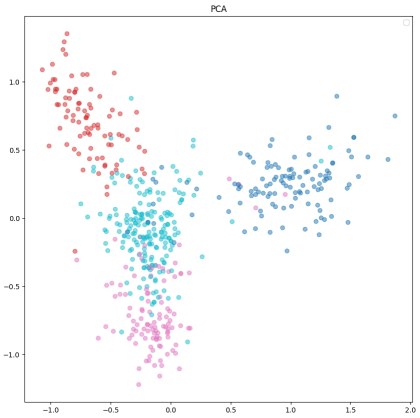

Figure 1: Muestra (10%) de Vectores de Lemma Reducidos con PCA

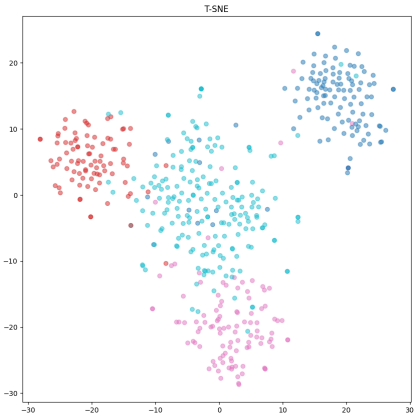

Figure 2: Muestra (10%) de Vectores de Lemma Reducidos con T-SNE

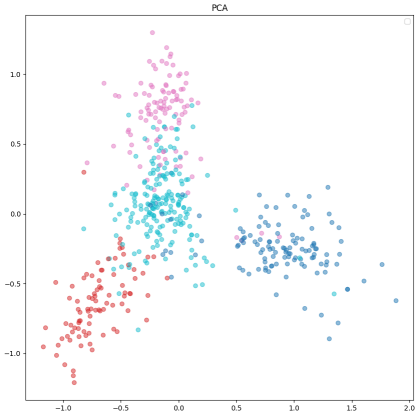

Figure 3: Muestra (10%) de Vectores de Stemming Reducidos con PCA

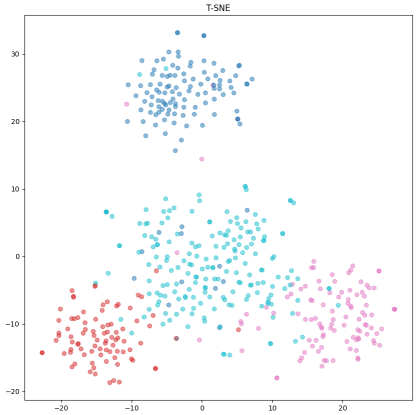

Figure 4: Muestra (10%) de Vectores de Stemming Reducidos con T-SNE

El segundo modelo **word2vec** que usamos fue el de Google, el cuál ya está preentrenado y cuenta con $3,000,000$ de registros de dimensión 300. Nuevamente sumamos cada vector de cada palabra dentro de un documento y dividimos entre el total de palabras para obtener el vector de

cada documento. De nuevo obtenemos una muestra para visualizar:

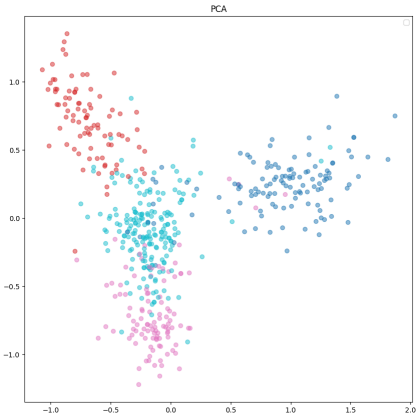

Figure 5: Muestra (10%) de Vectores de Lemma (Google) Reducidos con PCA

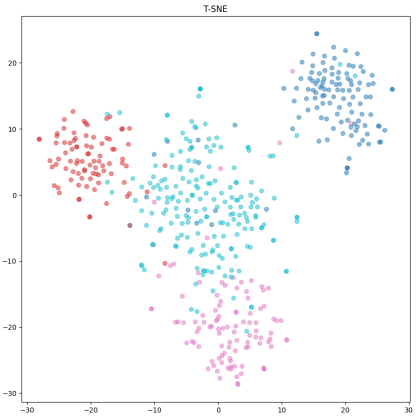

Figure 6: Muestra (10%) de Vectores de Lemma (Google) Reducidos con T-SNE

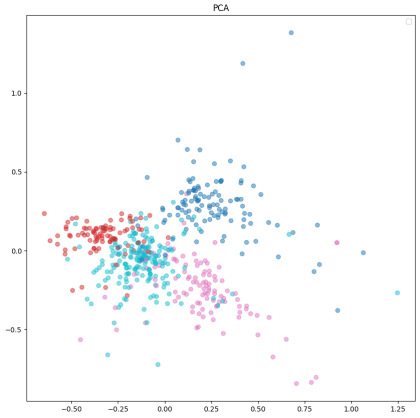

Figure 7: Muestra (10%) de Vectores de Stemming (Google) Reducidos con PCA

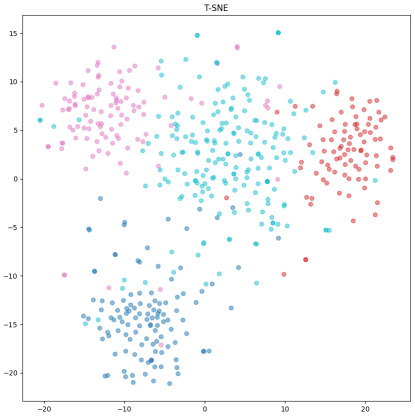

Figure 8: Muestra (10%) de Vectores de Stemming (Google) Reducidos con T-SNE

Finalmente la última representación numérica, la matriz de términos **TF-IDF**, se creó con la ayuda de la librería *sklearn*. Esta matriz tiene como columnas a cada una de las palabras distintas en el corpus y como entrada a cada uno de los documentos del mismo. (Una especie de one-hot). Sin embargo, debido al significante uso de memoria, fue un método que incluso no pudimos probar debido a la falta de recursos. Además dado que en principio nos enfocamos en los vectores de los documentos, basándonos en los scores obtenidos ya no fue necesario implementar esta alternativa.

`Similitud Coseno`

La fórmula de la similitud de coseno se utiliza para calcular la similitud entre dos vectores en un espacio vectorial. En nuestro caso de Word2Vec, esta fórmula se utiliza para medir la similitud entre vectores de palabras o documentos. Es decir qué tan cerca se encuentran dos palabras dentro de un contexto, etc. La fórmula es la siguiente:

$$similitud\ coseno(A, B) = \frac{A \cdot B}{||A|| * ||B||}$$

Un ejemplo de la similitud coseno entre dos vectores (lemma) de las palabras *cat* y *dog* son:

- Similitud coseno (vectores propios) : 0.40

  Esto indica una similitud moderada, pero no muy alta, entre los dos términos. En este caso, el valor sugiere que dichas palabras tienen cierta relación, pero no son términos altamente similares.

- Similitud coseno (vectores google) : 0.76

  Por otro lado, esto nos indica una similitud mucho más fuerte en el contexto de estos vectores en particular. Este valor nos dice que dichos términos son bastante similares en ese contexto y están relacionados de manera significativa.

Aplicando el mismo ejemplo de la similitud coseno entre dos vectores (stemming) de las palabras *cat* y *dog* en este caso son:

- Similitud coseno (vectores propios) : 0.36

  Esto indica una similitud media baja, entre los dos términos. En este caso, el valor sugiere que dichas palabras tienen poca relación.

- Similitud coseno (vectores google) : 0.76

  Nuevamente obtenemos que la similitud es mucho más fuerte en el contexto de estos vectores en particular. Este valor nos dice que dichos términos son bastante similares en ese contexto y están relacionados de manera significativa.

Dado que tenemos similitudes diferentes y estando conscientes de que una similitud de coseno alta no garantiza que un modelo sea mejor que otro para una tarea de clasificación específica. Utilizaremos ambos y compararemos su rendimiento.

`Clasificadores (XGBoost, Naive Bayes y SVM)`

Como bien vimos en las gráficas, ambos modelos de vectorización nos dejan resultados en la transformación de los documentos bastante similares. Esto posiblemente nos indicará que tanto el modelo entrenado con nuestro corpus como el modelo de Google tendrán scores relativamente similares.

Por lo tanto, una vez que ya tenemos nuestros documentos en vectores numéricos es momento de comenzar a entrenar a los algoritmos propuestos con el objetivo mismo de elegir aquél modelo que tenga los scores más altos y a su vez el preprocesamiento más adecuado (**lemmatization** o **stemming**). Primero debemos ver si nuestros algoritmos pueden ser optimizados de los hiperparámetros (y si bien aún no hemos terminado con el proceso de *fine-tuning* de los modelos) a fin de elegir la mejor estructura. De momento dichas estrucutras utilzadas han sido:

- XGBoost Clasificador. El clasificador utilizado fue el ya implementado en la librería *sklearn* y por lo tanto su estructura hasta el momento es la siguiente:

```
from xgboost import XGBClassifier
model = XGBClassifier(objective = 'multi:softmax',
                                    n_estimators = 100,
                                    early_stopping_rounds = 5,
                                    random_state = 42
)
```

- Gaussian Naive Bayes. Nuevamente utilizamos el clasificador implementado en *sklearn*. La estructura fue la siguiente:

```
from sklearn.naive_bayes import GaussianNB
model = GaussianNB()
```

- Support Vector Machines. Para finalizar, de nueva cuenta utilizamos el clasificador implementado en *sklearn*. La estructura fue la siguiente:

```
from sklearn.svm import SVC
model = SVC(kernel='rbf',
              C=7)
```

Volvemos a reiterar que el proceso de *fine-tuning* será llevado a cabo en la semana siguiente, debido a que ya es de los últimos pasos que nos permitirán elegir un buen modelo para clasificar.

Descripción del diseño experimental

Dado que entre los objetivos de este proyecto está determinar el mejor modelo de clasificación, el diseño experimental incluye la validación cruzada para el remuestreo los datos, así como tres métricas de evaluación diferentes que se aplicarán a las predicciones obtenidas de nuestros modelos de aprendizaje automático. Con lo anterior se busca hacer una búsqueda y evaluación exhaustiva de las configuraciones de los datos, con el fin de encontrar aquella que obtenga las mejores predicciones.

Remuestreo y particion de los datos

Nuestro conjunto de datos contiene cuatro categorías de productos: 'Hosehold', 'Books', 'Clothing & Accesories' y 'Electronics'. Sin embargo, las clases no están balanceadas, pues cada categoría contiene 19313, 11820, 1062 y 8670 observaciones respectivamente. Para lidiar con este desbalanceo en los datos, fue necesario usar la técnica de remuestreo de datos conocida como validación cruzada (*cross-validation*), específicamente *K-Folds Cross Validation*. Esta técnica consiste en dividir nuestro conjunto de datos en K sunconjuntos (*folds*) del mismo tamaño. Posteriormente se repite el procesos de train-test estandar k veces, de tal modo que en cada iteración un subconjunto diferente es usado como conjunto de prueba mientras que el resto de los $k-1$ conjunto son usados como conjunto de entrenamiento. El rendimiento del modelo se estima obteniendo las puntuaciones de las métricas de evaluación de las k iteraciones.

Debido a la alta dimensionalidad de nuestro conjunto de datos, resultado de la vectorización, nuestro *K-Folds Cross Validation* se definió con $k = 5$.

Metricas de evaluacion

Para cada uno de los *cross-validation* se utilizaron tres métricas diferentes para evaluar el rendimiento del modelo

1. **Precisión (*Accuracy*)**: La precisión del modelo está dada por el número correcto de ejemplos clasificados correctamente y divididos por el total de ejemplos clasificados. En términos de la matriz de confusión, queda definido por la expresión:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

2. **Recall**: El recall es la proporción de predicciones positivas y el número total de predicciones realizadas.En términos de la matriz de confusión, queda definido por la expresión:

$$Accuracy = \frac{TP}{TP + FN}$$

3. **F1-Score**: Es la media armónica del *precision* y el *accuracy* del modelo:

$$F1Score = \frac{2(Precision)(Recall)}{Precision + Recall}$$

Donde

$$Precision = \frac{TP}{TP + FP}$$

Resultados alcanzados por el método propuesto

MODELO	MÉTRICA	LEMMATIZATION	STEMMING
XGBoost Classifier	Accuracy	0.9791	0.9789
	Recall	0.9781	0.9789
	F1-Score	0.9794	0.9789
Naive Bayes Classification	Accuracy	0.8679	0.8671
	Recall	0.8688	0.8671
	F1- Score	0.8714	0.8670
Support Vector Classifier	Accuracy	0.9749	0.9611
	Recall	0.9743	0.9611
	F1- Score	0.9749	0.9611

Dentro del estado del arte, habíamos encontrado un estudio donde se entrenaron los modelos de Multinomial Naïve Bayes, Complement Naive Bayes, Linear Support Vector Classifier, Stochastic Gradient Descent, Ridge Classifier. Sin embargo, el más destacado fue LinearSVC logrando la tasa de precisión media más alta de 96,08%. Se utilizó Stemming y CountVectorizer como un preprocesamiento destacable. Si bien teníamos otros estudios, este fue el que obtuvo el score más alto. Sin embargo, al no encontrar artículos que explícitamente reportaran hacer uso del conjunto de datos que nosotros utilizamos, fue necesario hacer una revisión en **Kaggle** y ver qué otros enfoques existen en la red. Las clasificaciones más destacables son:

- LSTM - Accuracy : 0.9742
- (MultinomialNB) - Accuracy : 0.9438
- FastText - Accuracy : 0.9723
- TF-IDF - RandomForest : 0.9525
- TF-IDF - XGBoost : 0.9558

Al comparar los resultados anteriores con los obtenidos de nuestras implementaciones podemos observar una mejora en cuanto al puntaje de la métrica *accuracy*. Los scores del modelo XGBoost Classifier lograron superar aquellos obtenidos en con LinearSVC y FastText, las dos clasificaciones con mayores puntajes en la literatura. Es importante mencionar que la clasificación hecha con FastText no utilizó un preprocesamiento de texto exhaustivo para alcanzar su score alto de **accuracy**. Sin embargo, nuestros resultados muestran que las técnicas avanzadas de PLN sí son capaces de proporcionar mejoras en las predicciones, aún cuando ya contamos con puntajes de evaluación altos.

Por otro lado, si comparamos los scores obtenidos en nuestras implementaciones usando los dos tipos de vectorización, observamos que los scores correspondientes a *lemmatization* son ligeramente mayores a los scores del *stemming*. Tomando en cuenta que el proceso de stemming puede provocar la pérdida de contexto debido a la reducción de las palabras, el rendimiento del modelo no parece verse tan afectado por el proceso. Como la reducción de las palabras en el proceso de stemming provoca que el costo computacional del algoritmo sea menor que el de lemmatization, la ligera diferencia en nuestros resultados nos indica que el stemming sigue siendo una opción preprocesamiento viable aún cuando deseamos reducir los costos computacionales, por ejemplo, cuando tenemos un conjunto de datos de alta dimensionalidad.

Cronograma

TAREAS	FECHA DE INICIO	FECHA DE ENTREGA	% AVANCE
Definir el problema que queremos resolver	21/08/23	25/08/23	100%
Obtener los datos que necesitamos	21/08/23	25/08/23	100%
Propuesta del Proyecto Final	28/08/23	31/08/23	100%
Presentar la Propuesta	05/09/23	05/09/23	100%
Preparar y explorar los datos	11/09/23	17/09/23	100%
Limpieza (stopwords, caractéres especiales)	18/09/23	24/09/23	100%
Stemming y Lemmatizer	25/09/23	30/09/23	100%
Convertir texto en vectores	25/09/23	01/10/23	100%
Comprender los algoritmos (SVM, XGBoost, NN)	02/10/23	22/10/23	100%
Crear algoritmos con datos de entrenamiento y prueba	02/10/23	22/10/23	100%
Fine-tuning de los algoritmos	23/10/23	29/10/23	60%
Comprender los resultados	30/10/23	05/11/23	50%
Escribir reporte final de actividades y resultados y una presentación	06/11/23	17/11/23	0%
Presentación y entrega del proyecto	20/11/24	24/11/23	0%

Bibliography

word2vec [Mik+13] Mikolov, T. et al. *Efficient Estimation of Word Representations in Vector Space*. https://code.google.com/archive/p/word2vec/. Accessed on October 3, 2023. 2013.

gensim_word2vec [Řeh23] Řehůřek, R. *Gensim: models.word2vec – Word2vec embeddings*. https://radimrehurek.com/gensim/models/word2vec.html. Accessed on October 3, 2023. 2023.

shahane2019 [Sha19] Shahane, S. *E-commerce text dataset*. Version 2. 2019. URL: https://doi.org/10.5281/zenodo.3355823.