FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico N°2

"Predicciones de un set de datos"

- **Carrera:** Ingeniería en Informática.
- **Materia:** Organización de Datos.
- **Profesor:** Argerich, Luis
- **JTP**: Golmar, Natalia.
- **Cuatrimestre**: 1er Cuatrimestre 2020.
- **Fecha de entrega:** 21/05/2020
- **Nombre del grupo:** *Team_Undav*
- **Integrantes:**
    - Calonge, Federico Matías.
    - Ceballos Pardo, Sarah.
    - Flores, Matías.
    - Loiseau, Matías.
- **Repositorio:**
  https://github.com/MatiasLoiseau/nlp_dissaster_tweet

Índice

1 Objetivos

Este trabajo surge a partir de los resultados obtenidos del análisis exploratorio elaborados en el Trabajo Práctico N°1, donde buscamos comprender los datos mediante distintos análisis estadísticos.

El objetivo de este Trabajo Práctico N°2 es participar en la competencia de Kaggle 'Real or Not' [1] buscando posibles soluciones a la misma tratando de lograr el mejor Score/puntaje. Para esta competencia debemos utilizar distintos datos de Tweets que nos brinda Kaggle en 2 archivos CSV; teniendo que clasificar los Tweets que hablan sobre desastres naturales (target=1) contra los que no hablan de estos (target=0). Al tratarse de datos "etiquetados", debemos utilizar algoritmos de Clasificación (no de Regresión y/o Clustering).

Clasificar los tweets no es una tarea sencilla debido a la ambigüedad en la estructura lingüística de los tweets y, por lo tanto, no siempre está claro si las palabras de una persona realmente están anunciando un desastre o no. Por ejemplo, si una persona twittea: "On the plus side look at the sky last night, it was ablaze" (En español: "En el lado positivo, miré el cielo anoche, estaba en llamas"). La expresión 'ablaze' no significa que está en llamas realmente, sino que es una metáfora indicando que el cielo está anaranjado. Para nosotros es fácil entenderlo, pero para las máquinas no lo es.

De esta manera aplicaremos los conocimientos aprendidos durante la cursada teniendo en cuenta que trabajamos sobre la rama de *Natural Language Processing* (NLP). Más concretamente, durante la elaboración de este trabajo desarrollaremos varias pruebas de algoritmos de limpieza de datos, transformaciones, feature engineering, modelos de machine learning, entre otros.

[1] *https://www.kaggle.com/c/nlp-getting-started*

2 Desarrollo

2.1 Dataset

El dataset que analizaremos en este Trabajo Práctico serán 2 archivos CSVs.

●Test.csv: con el cual predeciremos los distintos tweets utilizando diferentes algoritmos.

●Train.csv: con el cual entrenaremos a los distintos algoritmos.

Ambos datasets[2] cuentan con datos de tweets de distintas personas hechas en distintas ubicaciones ('location') que fueron o no tweets que tratan sobre desastres. La diferencia entre ambos dataset, es que 'train.csv' tiene una columna 'target' que indica si el tweet verdaderamente habla sobre un desastre ('1') o si NO se trata de un desastre ('0').

id	keyword	location	text	target
32			London is cool ;)	0
33			Love skiing	0
34			What a wonderful day!	0
36			LOOOOOOL	0
37			No way...I can't eat that shit	0
38			Was in NYC last week!	0
39			Love my girlfriend	0
40			Cooool :)	0
41			Do you like pasta?	0
44			The end!	0
48	ablaze	Birmingham	@bbcmtd Wholesale Markets ablaze http:/	1
49	ablaze	Est. September 2(	We always try to bring the heavy. #metal #	0
50	ablaze	AFRICA	#AFRICANBAZE: Breaking news:Nigeria flag	1
52	ablaze	Philadelphia, PA	Crying out for more! Set me ablaze	0
53	ablaze	London, UK	On plus side LOOK AT THE SKY LAST NIGHT	0
54	ablaze	Pretoria	@PhDSquares #mufc they've built so much	0
55	ablaze	World Wide!!	INEC Office in Abia Set Ablaze - http://t.co/	1
56	ablaze		Barbados #Bridgetown JAMAICA Ã‰Ã›Ã' T	1
57	ablaze	Paranaque City	Ablaze for you Lord :D	0
59	ablaze	Live On Webcam	Check these out: http://t.co/rOI2NSmEJJ h	0
61	ablaze		on the outside you're ablaze and alive	0
62	ablaze	milky way	Had an awesome time visiting the CFC hea	0
63	ablaze		SOOOO PUMPED FOR ABLAZE ???? @south	0

Imagen 1. Data Set 'train.csv' (ejemplo primeros registros).

Columnas de nuestro Dataset:

- id - identificador único para cada tweet.
- keyword - una palabra clave del tweet (puede no tener valor: Nan).
- location - la ubicación de donde el tweet fue emitido (puede no tener valor: Nan).
- text - texto del tweet.
- target - permite saber si el tweet trata acerca de desastres (1) o no (0): presente solo en train.csv.

[2] Disponibles en *https://www.kaggle.com/c/nlp-getting-started/data*

2.2 Pruebas realizadas

2.2.1 Limpieza de datos

Primero realizamos la limpieza de los datos. Para no reescribir la columna text del dataset decidimos crear una nueva llamada “cleaned_text”, con la cual vamos a trabajar a lo largo de los algoritmos. Este feature va a almacenar el nuevo texto luego de eliminar palabras innecesarias y limpiarlo.

2.2.1.1 Features Keyword y Location

En primer lugar tratamos de verificar si nos convenía utilizar estas features en nuestro modelo. Ambos contienen muchos valores nulos, “Location” en mayor medida, por lo cual decidimos no añadirlo ya que nos iba a jugar en contra. En cambio, “Keyword” tenía una mayor cantidad de palabras que podían brindar información al modelo, entonces decidimos incluirla en el texto como una palabra más al principio del mismo.

A lo largo de las pruebas con distintos modelos probamos[3] agregando y sin agregar estas palabras. Aunque, en todos los casos, los resultados fueron peores agregando estas keywords, tanto en el set de entrenamiento como el de prueba. Por lo cual en los mejores scores no tuvimos en cuenta las keywords. Cabe destacar, que probamos de dos maneras distintas el uso de este feature. En un primer lugar decidimos incluir las palabras nulas en el texto como “unknown”, mientras que en segundo lugar incluimos solamente las palabras que estaban en el feature.

2.2.1.2 Eliminación de palabras innecesarias

Luego nos fijamos en lo que mostraban más adelante nuestros algoritmos de vectorización. Viendo y analizando las palabras que se encontraban en los textos pudimos apreciar que había una gran cantidad de ellas que eran valores numéricos que no aportan información al análisis de los datos, por lo cual decidimos eliminarlos.

Además, también eliminamos signos de puntuación, urls, símbolos o caracteres especiales y emojis, conservando las palabras que se encontraban en conjunto con estos. Por ejemplo de “#earthquake”, termina quedando solo “earthquake”. También pusimos todas las palabras en minúscula. Realizando esta extracción, pudimos comprobar un incremento en la eficiencia de nuestro modelo. Por lo cual en el modelo con la mejor solución, hicimos uso de esto.

[3] https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba02.1_MNB_GS_TFIDF_CV(StopWords)_Semi-Cleaned_Keywords.ipynb
https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba13.06_AgregandoKeyoword_na%3D'unknown'.ipynb
https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba17-BERT_Cleaned_Keywords.ipynb

2.2.1.3 StopWords

Además de lo mencionado en las secciones anteriores, decidimos eliminar las "palabras vacías". Estas palabras pueden ser artículos, preposiciones y pronombres, que no brindan información útil al modelo. Por esto, en los primeros modelos hicimos pruebas, eliminandolos mediante el algoritmo CountVectorizer()[4]. Esto influyó positivamente en el modelo, mejorando la eficiencia del mismo. Más adelante, en los modelos de redes neuronales secuenciales[5], también realizamos la prueba de eliminarlo, aunque en esos casos, el accuracy disminuye. Por ello, en las últimas pruebas no hacemos uso de esto.

2.2.1.4 Stemming

En los primeros modelos, también decidimos probar de hacer uso de la función de stemming[6]. Esto lo que hace es modificar las palabras de los textos a su "forma" raíz mediante la cual el modelo pueda encontrar una mejor relación entre las palabras. Su implementación influyó en los resultados de forma negativa, por eso decidimos no utilizarlo más adelante.

2.2.2 Algoritmos de Vectorización

Hacemos uso de distintos algoritmos a lo largo de nuestras pruebas para vectorizar las palabras de los textos de forma que los algoritmos clasificadores puedan interpretarlas de mejor manera.

2.2.2.1 Count Vectorizer

Para los primeros modelos decidimos utilizar este algoritmo, el cual segmenta cada texto en palabras y cuenta la cantidad de veces que cada palabra aparece en el texto. Devuelve una matriz, cuya cantidad de filas es el número de textos del dataset, y la cantidad de columnas son las palabras.

En los modelos que hicimos uso de este algoritmo no removimos previamente del dataset los stopwords, ni los signos de puntuación ya que este no los tomaba en cuenta.

2.2.2.2 TF-IDF

Este algoritmos asigna valores numéricos a las palabras en función a la frecuencia en la que aparecen en los textos y la cantidad de palabras totales en los

[4]https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba02_MNB_GS_TFIDF_CV(StopWords)_Semi-Cleaned.ipynb

[5]https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba13.04%20con%20StopWords%2C%20split_0.25%2C%20epoch_10.ipynb

[6]https://github.com/MatiasLoiseau/nlp_dissaster_tweet/blob/master/Codigo/Prueba06-07_MNB_GS_TFIDF_CV(Stemming%20-%20StopWords)_Semi-Cleaned.ipynb

mismos. Estos algoritmos los utilizamos en los primeros modelos hasta que llegamos a implementar los modelos de redes neuronales secuenciales.

2.2.2.3- Word Embeddings - Glove

Para los modelos de redes neuronales secuenciales, decidimos utilizar GloVe, ya que veíamos que en muchos trabajos realizados hacían uso de este modelo con su set de datos pre entrenados.

Decidimos utilizar este, y no otros como Word2Vec porque vimos que era el más utilizado con el tipo de dataset con el que estamos trabajando. Este tipo de embedding trabaja creando una matriz de ocurrencia generada a partir de un corpus.

Como trabajamos con los datos ya entrenados de los archivos txt de GloVe, nos tocó probar cual era más eficiente para nuestro caso, siendo este el de 100 dimensiones. Por lo cual, la matriz generada tiene como filas las palabras del texto, y como columnas las distancias.

2.2.3 Funciones utilizadas en los algoritmos

2.2.3.1 Train_Test_Split()

Para todos los casos dividimos nuestro set de entrenamiento, en subsets de entrenamiento y validación, mediante la función train_test_split(). Esto lo hacemos para prevenir el overfitting en el modelo. En la gran mayoría, probamos de cambiar los tamaños de cada uno, para comprobar cuál era más eficiente. Finalmente, en la mayoría de los casos nos quedamos con un 80% de entrenamiento y 20% de validación.

2.2.3.1 GridSearchCV()

Utilizamos Grid Search con Cross Validation en la mayoría de los modelos para buscar los mejores hiper parámetros de los algoritmos. Con esto, podemos verificar que en la mayoría de los casos utilizando esos parámetros obtuvimos mejores resultados.

2.2.4 Algoritmos en conjunto con Count Vectorizer y TFIDF:

2.2.4.1 Multinomial Naive Bayes

El primer clasificador con el que decidimos probar es con el multinomial Naive Bayes. NB es un algoritmo muy flexible que se puede usar en diferentes tipos de datasets fácilmente y sin restricciones, ya que solo se deben usar las características numéricas (esto lo logramos previamente vectorizando nuestros textos en números).

De todos estos algoritmos clasificadores (sin utilizar redes neuronales), NB fue el que mejor puntaje nos dio llegando a 0.79558 en nuestro mejor submit.

En un principio probamos con los parámetros por defecto. Más adelante implementamos el uso de grid search con cross validation para buscar los mejores hiper parámetros para nuestro subset de entrenamiento. En el caso de Naive Bayes, solo teníamos un parámetro para modificar, “alpha”, por ende realizamos la búsqueda con ese único parámetro. Una vez encontrado el mejor valor, guardamos el modelo con el mejor parámetro y lo probamos nuevamente con el subset de validación. En este subset de validación nos dio un resultado mayor al obtenido en el de entrenamiento. Al subir las predicciones hechas por este modelo nos dio el score mencionado anteriormente.

2.2.4.2 Stochastic Gradient Descent (SGD Classifier)

Luego de haber conseguido el mejor resultado con el modelo anterior, decidimos hacer las mismas pruebas con distintos modelos. Primero probamos con SGD Classifier. Este modelo a diferencia del anterior, tiene muchos más parámetros con los que trabajar. Ya vemos parámetros como regularizadores y learning rate, por lo cual estamos trabajando con un modelo más complejo.

Probamos varias veces con distintos parámetros, para finalmente probarlo con grid search. Una vez hecho esto, efectivamente con esos parámetros daba un mejor resultado que con otros probados. Con este modelo llegamos a un score de 0.79313. Siendo menor que el obtenido por el anterior modelo.

2.2.4.3 SVC (Support Vector Classifier)

Este algoritmo es parecido al anterior, aunque se utiliza para trabajar con muestras no más grandes de 10000, lo cual nos viene bien, ya que tenemos una menor cantidad de textos. En este caso trabajamos con los parámetros “C” (Término de regularización) y “gamma”. Probamos varias combinaciones de parámetros con grid search, y la que nos dio mejor resultado fue la que subimos a Kaggle, recibiendo un score de 0.78639, siendo el más bajo hasta el momento.

2.2.4.4 Regresión Logística

La técnica de Clasificación de Regresión Logística consiste en una red neuronal “en miniatura” (una red neuronal con exactamente una neurona). Utiliza una función logística para predecir las probabilidades de que el resultado sea de una clase u otra. Probamos cambiando diversos valores, como el de regularización, y dio un valor de score de 0.78608.

2.2.4.5 Árboles de decisión (Decision Tree Classifier)

El único parámetro que modificamos en este caso fue el de “max_depth” dándole valores de 0 a 100, de 10 en 10. Con el mejor parámetro llegó a un score de 0.74563 siendo el peor resultado hasta el momento. Debido a esto no probamos más con este modelo y pasamos al siguiente.

2.2.4.6 Árboles de decisión (XGBoost)

XGBoost es una implementación de árboles de decisión con Gradient boosting diseñada para minimizar la velocidad de ejecución y maximizar el rendimiento.

Este fue el modelo que más tardó en encontrar los parámetros con grid search, tal vez sea porque pusimos muchos parámetros a buscar. Aunque el resultado final no fue tan bueno como con otros modelos, generando un score de 0.76984.

2.2.4.7 KNN (K Nearest Neighbour)

Este algoritmo nos permite clasificar datos calculando las distancias de un nuevo punto de datos a todos los demás puntos de datos de entrenamiento.

En este algoritmo las pruebas que hicimos fue modificando la cantidad de vecinos, probamos en un rango de 0 a 100, de 10 en 10. Nos dio un resultado bastante bajo a comparación de los anteriores modelos, tal vez con una mayor cantidad de vecinos hubiera podido dar un mejor resultado.

2.2.5 Utilizando Redes Neuronales Secuenciales:

Como con los modelos anteriores no pudimos superar una puntuación de 0.795, decidimos pasar a otros tipos de modelos. En estos casos utilizamos modelos secuenciales, en donde el input fue la matriz generada mediante GloVe.

La implementación de las soluciones con redes neuronales fueron desarrolladas con las bibliotecas Tensorflow y Keras, dando así mucha versatilidad para la arquitectura de las redes elaboradas.

2.2.5.1 CNN

El primer método que probamos fueron las redes neuronales convolucionales (CNN), las cuales son una evolución de las redes neuronales. En síntesis, mediante el proceso de convolución se aplican filtros para extraer características de los textos (en este caso tweets) y luego se aplican mecanismos de droptout y max pooling para poder divergir en la clasificación del tweet.

Sin importar la arquitectura de CNN desarrollada, no pudimos hacer un modelo que predijera aceptablemente, ya que el resultado del score fue 0.57.

2.2.5.2 LSTM

Luego de las pruebas fallidas de CNN, probamos una segunda implementación de redes neuronales llamadas Long Short-Term Memory (LSTM). La misma es una evolución de las redes neuronales recurrentes donde en cada una de sus unidades retienen información por un lapso corto de tiempo. Esta implementación es normalmente utilizada para resolver problemas de NLP. Además, en este trabajo se utilizó la técnica Bidirectional-LSTM, la cual entrena dos entradas LSTM. La primera capa es entrenada tal cual como es, pero la segunda es

una copia al reverso de la primera. Esto genera adquirir un contexto más general a la red.

Gracias a las bibliotecas Tensorflow y Keras, pudimos desarrollar nuestras propias arquitecturas y de esa manera analizar cuál era la mejor combinación de hiper parámetros, herramientas y métodos para que produzca el mayor puntaje.

Las primeras pruebas fueron combinando CNN y LSTM como muestra el siguiente resumen:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, None, 100)         2270100
_________________________________________________________________
spatial_dropout1d (SpatialDr (None, None, 100)         0
_________________________________________________________________
conv1d (Conv1D)              (None, None, 32)          25632
_________________________________________________________________
max_pooling1d (MaxPooling1D) (None, None, 32)          0
_________________________________________________________________
bidirectional (Bidirectional (None, None, 64)          16640
_________________________________________________________________
global_max_pooling1d (Global (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense (Dense)                (None, 1)                 65
=================================================================
Total params: 2,312,437
Trainable params: 2,312,437
Non-trainable params: 0
```

Imagen 2. Modelo secuencial CNN y LSTM..

El resultado de esta arquitectura fue parecido como el de las primeras pruebas con puntajes de 0.78 y 0.79, pero intuimos que podíamos mejorarlo agregando más capas. Además la combinación CNN + LSTM no fue del todo buena y optamos por quedarnos solo con Bidirectional-LSTM.

Nuestras experiencias con las pruebas fueron que habían problemas con el overfitting. Los valores de accuracy y loss entre los set de test y train se bifurcaban muy rápido. Por esa razón agregamos una capa más de Bidirectional-LSTM y luego capas “Dense” (o capas de redes neuronales completamente conectadas) para que logre representar información más compleja. Además se agregaron varias capas de dropout para combatir el overfitting. Dicha arquitectura puede verse representada en la siguiente figura:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_5 (Embedding)      (None, 31, 100)           1683400
_________________________________________________________________
bidirectional_6 (Bidirection (None, 31, 62)            32736
_________________________________________________________________
bidirectional_7 (Bidirection (None, 31, 62)            23312
_________________________________________________________________
global_max_pooling1d_5 (Glob (None, 62)                0
_________________________________________________________________
batch_normalization_5 (Batch (None, 62)                248
_________________________________________________________________
dropout_11 (Dropout)         (None, 62)                0
_________________________________________________________________
dense_11 (Dense)             (None, 31)                1953
_________________________________________________________________
dropout_12 (Dropout)         (None, 31)                0
_________________________________________________________________
dense_12 (Dense)             (None, 31)                992
_________________________________________________________________
dropout_13 (Dropout)         (None, 31)                0
_________________________________________________________________
dense_13 (Dense)             (None, 1)                 32
=================================================================
Total params: 1,742,673
Trainable params: 1,742,549
Non-trainable params: 124
```

Imagen 3. Modelo secuencial BLSTM.

Si bien el problema del overfitting mejoró, durante el proceso de entrenamiento se usaron tres herramientas de Keras que nos pareció importante usar durante el desarrollo. Estas son el EarlyStop, CheckPoint y ReduceLR. El EarlyStop es un mecanismo para que el modelo deje de entrenarse cuando vea que algún valor (en nuestra caso el valor de loss del set de validación) no mejore. El CheckPoint es un mecanismo para guardar el mejor resultado del modelo en algún epoch. El reduceLR reduce el learning rate del modelo si ve que durante cierto tiempo no hubo mejoras.

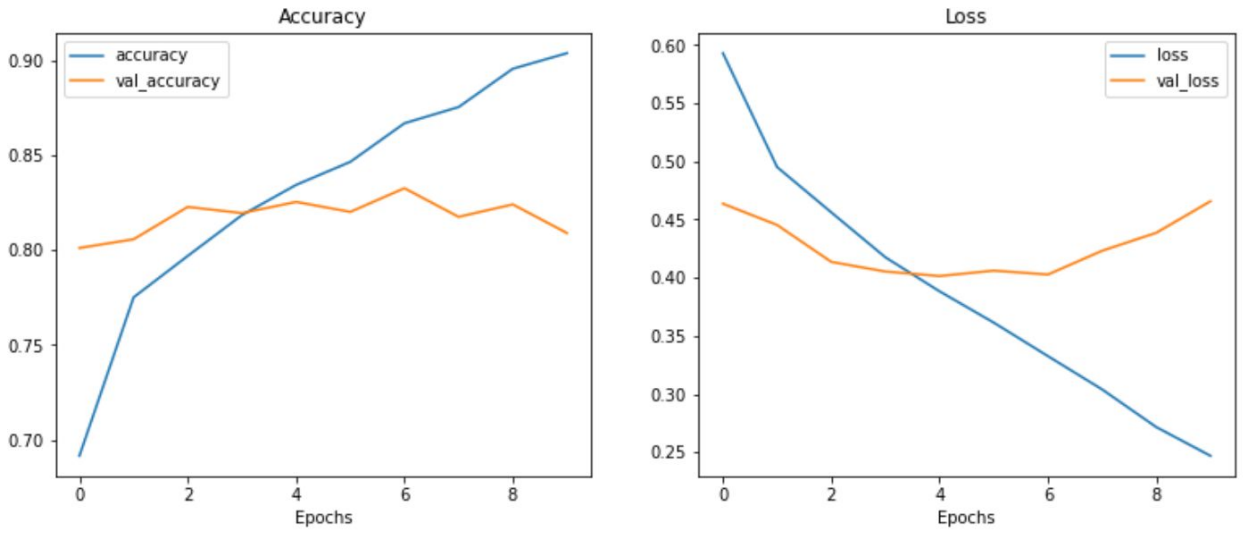

Imagen 4. Gráficos de Loss y Acuracy.

En la *imagen* 4 se puede apreciar cómo se produce el overfitting en el modelo pero gracias al ModelCheckPoint se guardó el mejor momento del entrenamiento que fue, en este caso, en el cuarto epoch.
Con los últimos modelos llegamos a un score arriba de 0.8, llegando a 0.81673 en la mejor prueba.

2.2.5.4 BERT (Bidirectional Encoder Representations from Transformers)

Por último decidimos utilizar algo más novedoso y que, en teoría, mejoraría significativamente los resultados obtenidos con representaciones tales como word embeddings con Glove. En estos últimos casos, se genera una representación de una sola palabra para cada palabra en el vocabulario, mientras que BERT tiene en cuenta el contexto para cada aparición de una palabra determinada. Por ejemplo, mientras que el vector de la palabra "estrella" tendrá la misma representación vectorial de Glove para sus dos ocurrencias en las oraciones "Es una estrella de rock" y "El sol es una estrella", BERT proporciona una representación diferente para cada oración.

BERT (Representación de Codificador Bidireccional de Transformers)[7] es una técnica basada en redes neuronales para el pre-entrenamiento de NLP. Fue desarrollada por Google y publicada en 2018.

Que sea una representación de lenguaje bidireccional quiere decir que se analizan las palabras ubicadas a la izquierda y a la derecha de cada uno de los términos; comprendiendo así más en detalle la profundidad e intención de los textos analizados. Además, BERT es una técnica de ML NO supervisada (no requiere un corpus con las respuestas correctas sino que se infieren directamente) y es pre-entrenado utilizando solo un corpus de texto plano.

En síntesis, BERT consiste en 2 pasos:

1. Entrenar un modelo de lenguaje en un gran Corpus de texto sin etiquetar (aprendizaje no supervisado).
2. Ajustar/tunear este modelo a tareas específicas de NLP para utilizar este "depósito de conocimiento" que este modelo adquirió. Utilizando en este paso nuestros datos etiquetados (aprendizaje SUPERVISADO).

En nuestro caso utilizamos la librería Ktrain[8] para implementar de una manera sencilla la compleja arquitectura y modelo de BERT. Al final fuimos capaces de lograr un score final de 0.083726.

[7] *https://arxiv.org/abs/1810.04805*
[8] *https://github.com/amaiya/ktrain*

Las 3 pruebas que realizamos son las siguientes:

1. "*Prueba_SoloBERT.ipynb*": Para esta prueba únicamente aplicamos BERT a nuestros textos sin realizar ninguna limpieza ni tratamiento con keywords. El preprocesamiento lo realiza internamente la librería ktrain. Igualmente, con esta prueba obtuvimos un gran Score.

   Score obtenido: 0.83389

2. "*Prueba BERT Cleaned.ipynb*": Esta fue nuestra mejor prueba. Lo que hicimos fue agregar un preprocesamiento al texto previo al modelo de BERT. En este preprocesamiento eliminamos, como hicimos anteriormente: símbolos, urls, emojis, números, etc. Además, realizamos un cut en los tweets de mayor longitud; dándole más importancia al inicio del tweet que al final. Ya que, generalmente, en los tweets de gran tamaño, sus últimas palabras generalmente son links o datos muy largos que son irrelevantes para el análisis.

   ***Score obtenido*: 0.083726**

3. "*Prueba BERT Cleaned Keywords.ipynb*": Además de la limpieza previa, lo que hicimos fue, como en los casos anteriores, añadir la columna 'Keyword' delante del texto del tweet. Esto no nos llevó a ninguna mejora, sino que nuestro score bajó.

   Score obtenido: 0.82807

3 Conclusión

Durante la elaboración de este trabajo práctico se trató de resolver un problema de clasificación de texto. Se probaron métodos comunes con aplicaciones varias como lo son XGBoost, SGDClassifier, KNN, CNN, entre otros. Además se probaron modelos más específicos de NLP como LSTM y BERT, los cuales fueron los que mejor se ajustaron a la competencia.

Encontramos varias conclusiones en este trabajo:

1. No siempre los pre-procesamientos del texto mejoran el rendimiento del modelo. Ya que se probaron múltiples soluciones de cleaning text, stemming, stopwords y dependiendo del modelo, habían o no mejores resultados.
2. Grid Search es una poderosa herramienta para encontrar la mejor combinación de hiper parámetros para los algoritmos más generales.
3. Agregar capas de redes neuronales aumenta (hasta cierto punto) el rendimiento del modelo.
4. Los algoritmos probados con mayor rendimiento y escalables son LSTM y BERT.

De todos los modelos que probamos la mejor solución la obtuvimos con BERT, con el cual pudimos superar 0.02 puntos en comparación con el anterior modelo con mayor puntuación (BLSTM). Nuestro score final fue de **0.83726.**

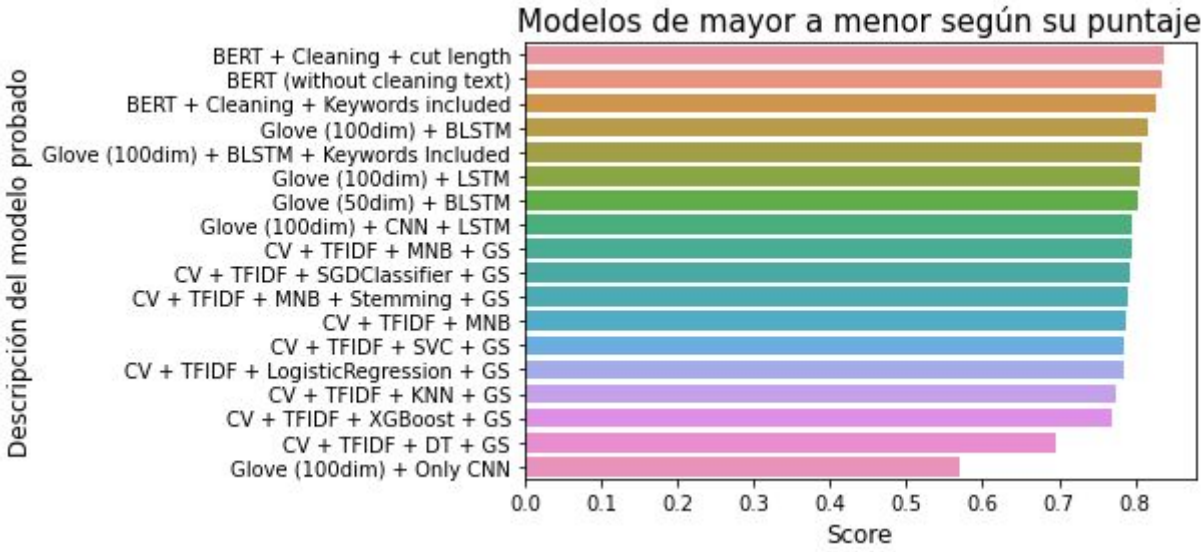

Imagen 5. Tabla de puntuación.

Uno de los mayores problemas y desventajas que tuvimos al querer utilizar el modelo BERT es que requiere de mucho tiempo de entrenamiento, siendo muy costoso computacionalmente. Aún teniendo el modelo de BERT pre-entrenado, la fase de 'tuning' nos tardó muchísimo tiempo: en Google Colab tardamos aproximadamente 4 hs en el entrenamiento. Debido a esto tuvimos que utilizar una PC con una placa de video de gran capacidad de memoria (6GB) para poder realizar estos entrenamientos, sino, esto nos hubiese sido una tarea prácticamente imposible. Otra opción para mejorar esto era aplicar una versión más ligera de BERT como distilBERT.

4 Bibliografía

- Apunte de la materia: https://piazza.com/class_profile/get_resource/k7s41iiajq271y/k7s577n4h1g14s
- https://www.aprendemachinelearning.com/procesamiento-del-lenguaje-natural-nlp/
- https://medium.com/dair-ai/a-light-introduction-to-bert-2da54f96b68c
- Ktrain: https://github.com/amaiya/ktrain