FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 2: Machine learning

[75.06] Organización de Datos
Primer cuatrimestre de 2020

BERTOLOTTO, Francisco	fbertolotto@fi.uba.ar	102671
LÓPEZ NÚÑEZ, Agustín	alopezn@fi.uba.ar	101826
MARCHESE, Milena	mmarchese@fi.uba.ar	100962
SANTONI, Mauro	msantoni@fi.uba.ar	102654

Repositorio de Github: https://github.com/milenamarchese/OrganizacionDeDatos
Set de Datos: https://www.kaggle.com/c/nlp-getting-started

Índice

1. Feature Engineering

1.1. Procesado de texto

Para comenzar el entrenamiento de modelos predictivos se realizó un trabajo de limpieza de datos. El primer filtro de texto está compuesto por una serie de regex que imitan a un script de Ruby utilizado para un embedding específicamente para tweets por GloVe, estas regex se encargan de reemplazar:

- URLs por 'url'.
- Emoticones como ':)' o '<3' por 'smile' y 'heart' respectivamente.
- Menciones de twitter (@username) por 'user'.
- Números por 'number'.

Luego se eliminan los signos de puntuación y se pasa todo el texto a minúsculas. Este procesado del texto dio resultados significativamente mejores a lo largo de todo el trabajo en comparación con el texto sin procesar (únicamente sin NaNs y duplicados).

1.2. Extracción de Features

1.2.1. Features a partir de la palabra clave

De la columna 'keyword' se extrajeron vectores de 100 dimensiones con un tokenizador preentrenado: GloVe. GloVe dispone de varios grupos de vectores preentrenados, en este caso se utilizó el set entrenado con Wikipedia compuesto por 6B de tokens.

1.2.2. Features a partir de la ubicación

De la columna 'location' de ambos datasets se extrajeron dos grupos diferentes de features:

- Coordenadas: Previamente, en el análisis exploratorio de datos se había utilizado la librería GeoPy como herramienta para eliminar ubicaciones redundantes como "new york", "nycz "new york, ny", y además descartar ubicaciones falsas. El dataset que se obtuvo con el servicio de geocoding estaba compuesto por dos columnas: 'address' y 'point', siendo 'point' las coordenadas del lugar indicado en 'address'. Se decidió incluir las columnas x e y al dataset de features, así obteniendo información de la columna 'location' de forma numérica.
- Vectorización con GloVe, utilizando el mismo grupo de vectores que se utilizó con las palabras claves.

1.2.3. Features a partir de los tweets

Una parte muy importante del procesamiento de los textos son las denominadas stopwords, las cuales se definen como palabras sin significado o muy repetidas en el idioma que no aportan en mucho más que ruido para la clasificación. Para el trabajo relacionado con este tipo de palabras se manejó la libreria NLTK resultando en las features siguientes: la cantidad de stopwords, la relación stopword/cantidad de palabras y la longitud de la stopword más larga entre todas las contenidas en el texto.

Un feature interesante es el del sentimiento del tweet, que previamente se había concluído que los tweets basados en desastres reales tendían a tener un sentimiento más negativo y los tweets no basados en desastres reales tendían a tener un sentimiento distribuído más equitativamente. Creemos que ésta información fue útil para el entrenamiento del modelo. Este análisis se realizó con la librería de NLTK, SentimentAnalyzer.

Los features relacionados con la cantidad de adjetivos, sustantivos, verbos y adverbios contenidos en el texto del tweet fueron obtenido mediante la utilización de la librería TextBlob la cual

permite analizar un texto y, entre otras cosas, clasifica a las palabras con tags (VB para verbos, NN para sustantivos, etc).

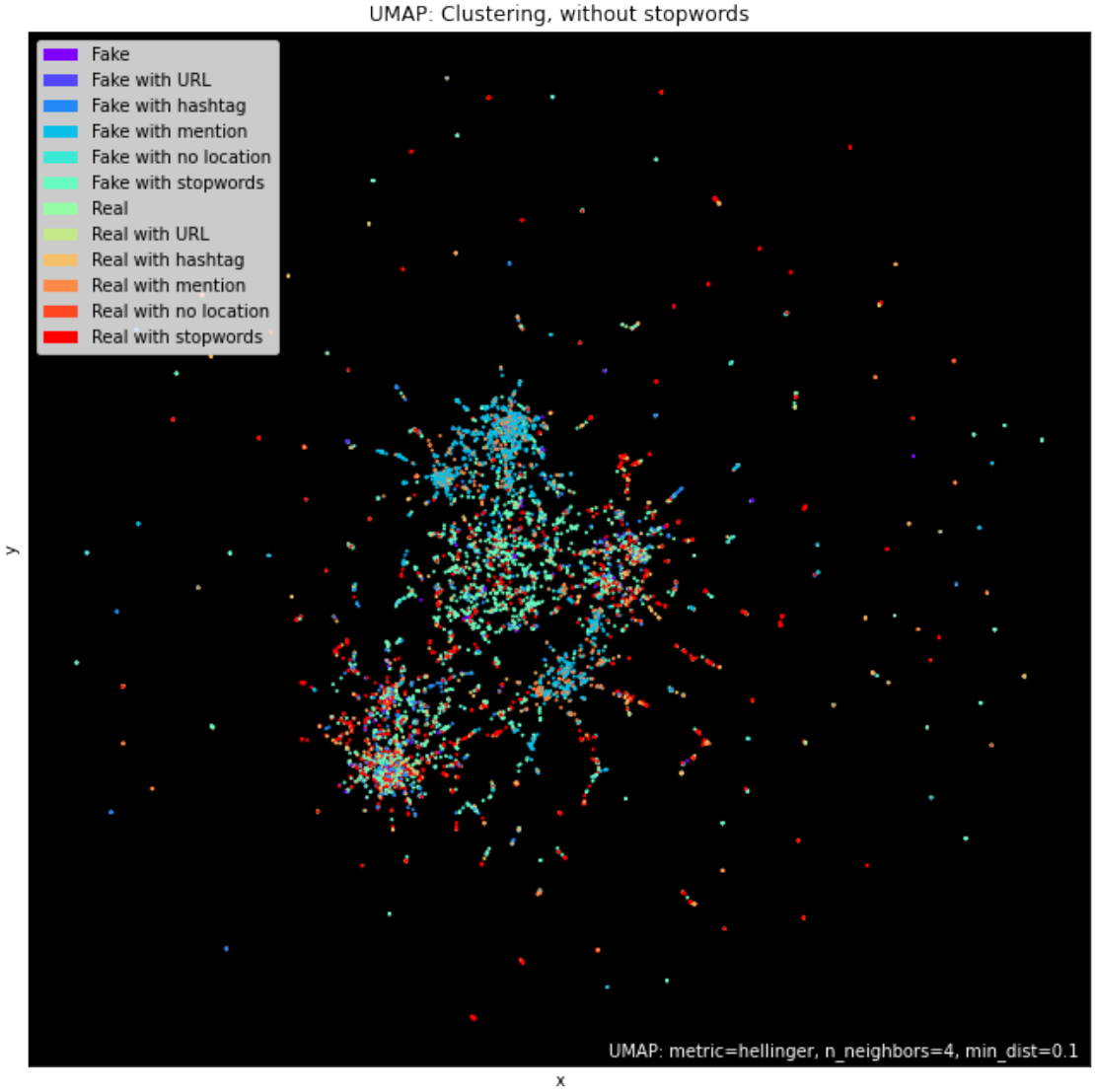
Los features restantes fueron obtenidos por scripts simples buscando promedios de longitud de las palabras, cantidad de palabras unicas, cantidad de hashtags o menciones, entre otras.

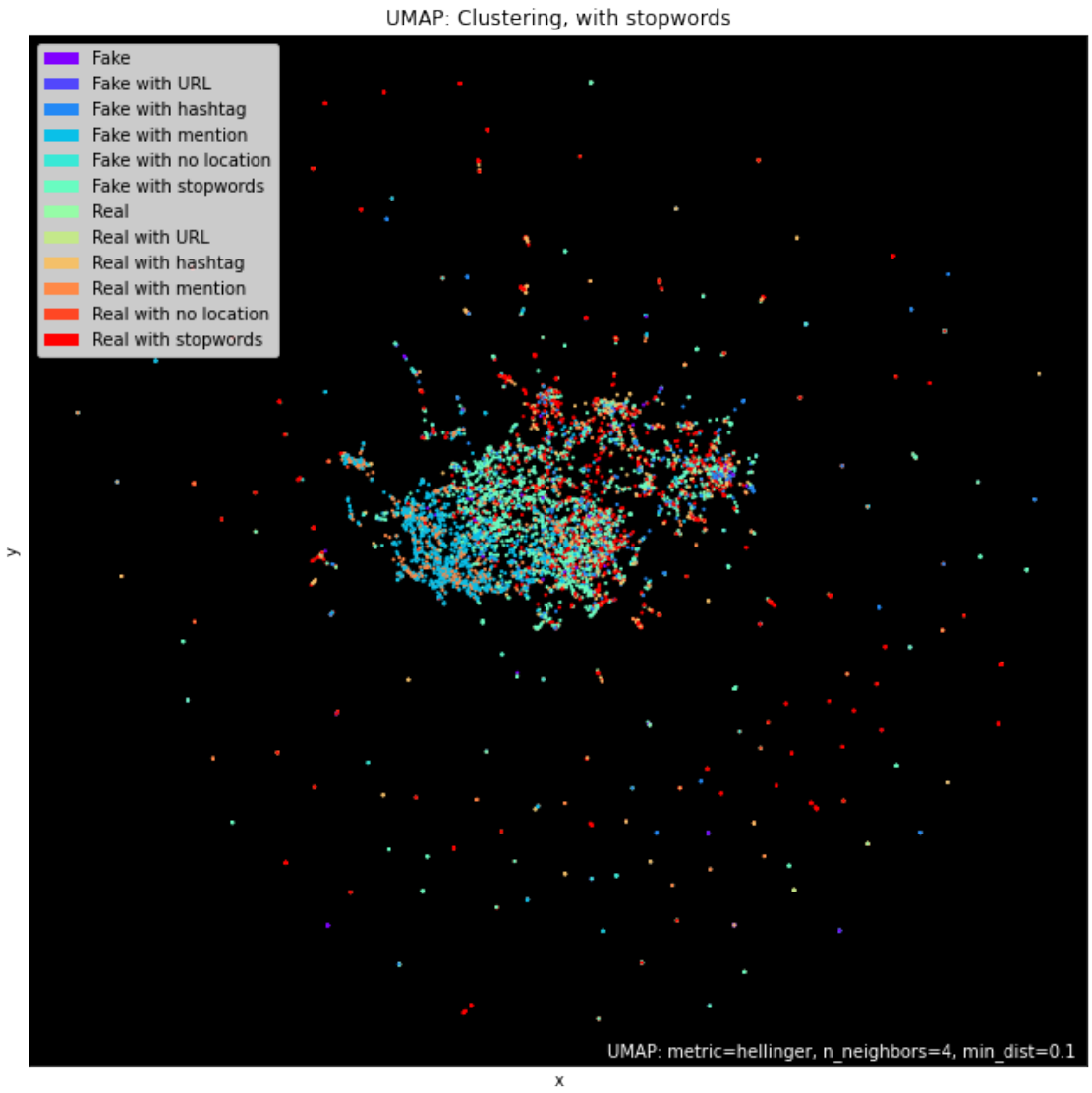
1.2.4. ID como feature

Usar los ID como feature fue ignorado la mayor parte del tiempo del trabajo dado que teóricamente no debería existir correlación entre los ID del set de train con los del set de Test. Aún así, este feature rankeó relativamente alto en el feature importance consolidado final, lo cual fue tomado en cuenta y aplicado casi especialmente para el modelo de BERT, con el que se obtuvo el mejor resultado. Con lo cual, aunque a priori este feature no debería aportar información relevante, lo hace para los modelos contribuyendo al resultado. Finalmente, las diferencias entre su utilización y su no utilización fueron mínimas, pero considerables para una competencia como ésta.

2. Análisis de Clustering

Como primera aproximación a la distribución del set de datos, junto a la información obtenida del análisis previo se decidió visualizar los datos mediante UMAP. Se procedió procesar el texto como fue explicado en la sección de Feature Engineering y luego se obtuvo una representación de 5000 dimensiones de los tweets mediante CountVectorizer, se realizaron distintas pruebas: texto procesado mateniendo stopwords, texto procesado quitando stopwords y se agregaron features numéricos. En las primeros intentos de visualizar clusters no se observaba un comportamiento definido entre tweets sobre desastres reales y falsos, entonces se decidió generar otra columna que cumpla la función de target para el set. Estas nuevas labels contemplan si el tweet está basado en un desastre real o falso pero también si contienen una URL, una mención, o un hashtag, o si sus columnas 'location' y 'keyword' contienen NaN. En muchas de las pruebas aún afinando el parámetro 'n_neighbors' (tomado como parámetro de resolución) no se observaban grupos definidos, concluyendo estos fueron los gráficos donde mejor se distinguían los clusters:

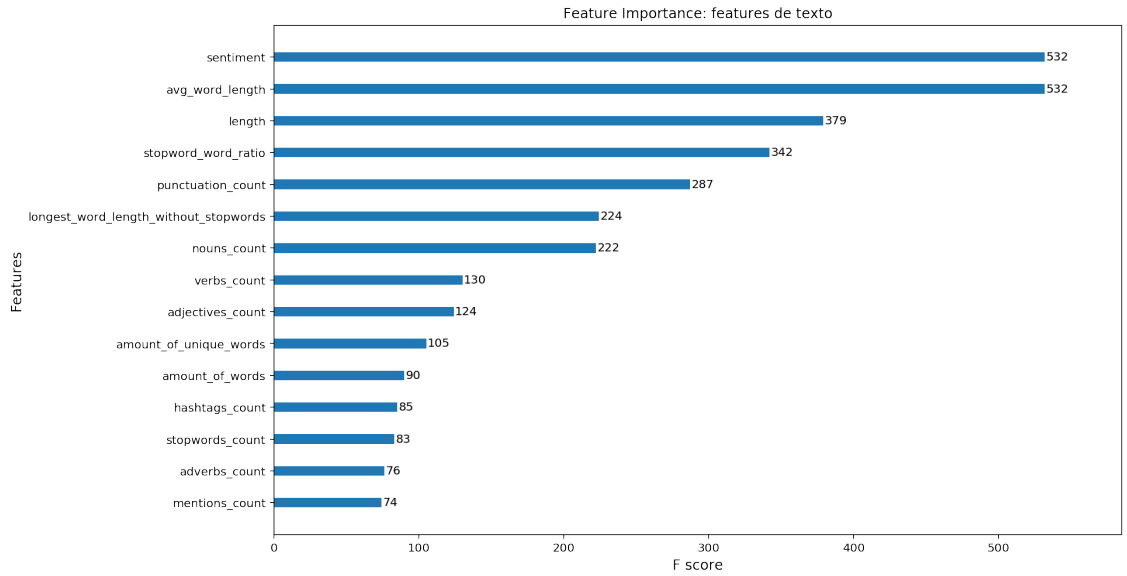




Como conclusión observamos que quitando las stopwords los clusters se separan mucho más que conservándolas, también hay labels que resaltan más que otras, como la presencia de stopwords y menciones en tweets sobre desastres falsos. El resto de las labels se ven mucho más dispersas. Con esta información tenemos una intución de los features importantes y como pueden influir las stopwords sobre el set.

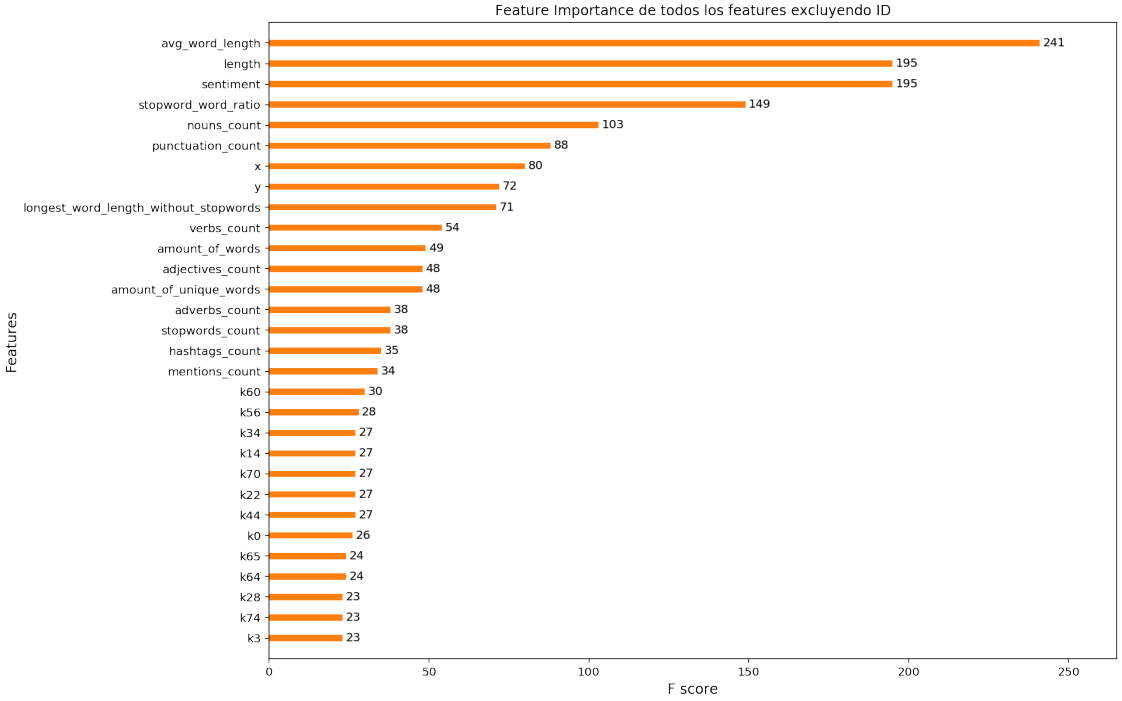
3. Feature Importance

En esta sección se analiza el feature importance de todos los features utilizados a través del trabajo. Los gráficos fueron producidos gracias al *plot importance* de XGBoost, que es reconocido como el algoritmo que provee superior determinación de la importancia de features en general. Como hipótesis y no como generalidad se puede suponer que esta importancia de features provista por XGBoost sea extrapolable a redes neuronales y los resultados demuestran que al agregar los features la performance del modelo final subió, con lo cual resultó confirmada la hipótesis, que puede no llegar a cumplirse en otro caso.



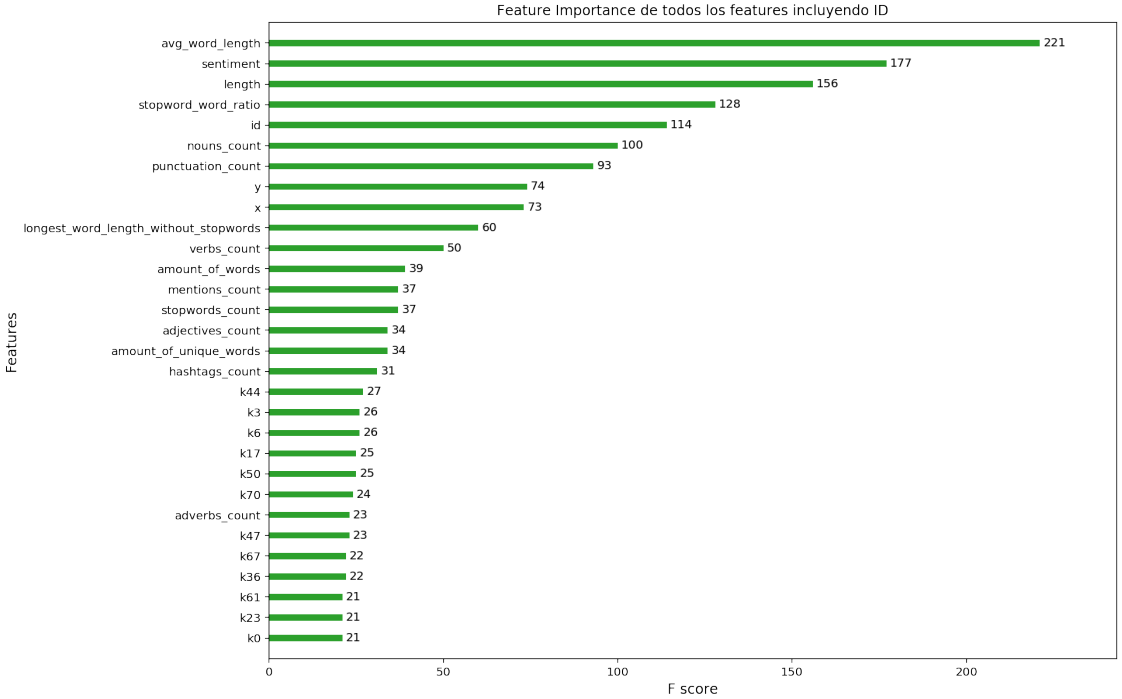
Feature importance utilizando sólo features de texto.

En este caso se tomaron sólamente los features de texto para observar que relevancia tenía cada uno para el entrenamiento del modelo. Es notable el caso del largo promedio de las palabras de un tweet que consistentemente en éste y los gráficos subsiguientes demuestra ser uno de los features más importantes; seguido por el largo del tweet; el sentimiento, cuyo análisis del análisis exploratorio demuestra ser de gran utilidad; y los features de puntuación y stopwords. También hay que hacer una mención notable al feature de cantidad de sustantivos que contiene un tweet, que fue un feature novedoso agregado en este trabajo y que no tuvo análisis previo, pero sin embargo demuestra proveer mucha información al modelo.



Feature importance utilizando features de texto, locations y keywords. Se tomó un top 30 ya que los embeddings son de 100 dimensiones.

Se puede observar que las coordenadas de las location aportan mucha información ya que aparecen por encima de varios features del texto en sí, lo cual determina que el trabajo tomado en el análisis exploratorio para obtener las coordenadas de las location fue fructífero.



Feature importance utilizando features de texto, locations, keywords y ID. Se tomó un top 30 ya que los embeddings son de 100 dimensiones.

Aquí es evidente que aunque no pareciera, el feature de ID aporta mucha información que le resultó útil al modelo para predecir con mayor *accuracy* los tweets del set de Test.

4. Modelos Probados

En esta sección se detallan los modelos que fueron probados, su funcionamiento básico y sus resultados preliminares.

4.1. Modelos descartados

Estos modelos fueron probados y por su mal desempeño fueron discontinuados.

4.1.1. KNN

KNN es un algoritmo muy conocido en las prácticas de machine learning y una gran ventaja que posee es su rápido aprendizaje (trade off con la lentitud con la que predice). Para textos es muy común la utilización de TF-IDF en combinación con KNN dando resultados aceptables pero terminaron sirviendo más de base que para una gran predicción (el score logrado fue de 0.77352). Según el siguiente trabajo publicado aparantemente KNN no da muy buenos resultados con noticias diarias (que está muy relacionado con el tipo de tweets). Se utilizaron otros tokenizadores como CountVectorizer o Hash Vectorizer pero los resultados fueron similares o peores.

4.1.2. XGBoost

Algoritmo ganador en muchas competiciones de Kaggle, basado en un ensamble de árboles de decisión que va corrigiendo los errores de los árboles que antes procesaron los datos. En general funciona muy bien con TF-IDF para clasificar textos y aunque logró superar a KNN no fue por mucha cantidad y se quedó con un score de 0.78179.

4.1.3. Voting classifier

Se usó el algoritmo de majority voting con modelos de KNN y XGBoost, éste usa a cada modelo como un peso para cada feature y en eso basa la votación. Como se utilizaron varios modelos de XGBoost con parámetros encontrados mediante cross-validation se optó por una votacion 'soft' que es recomendada para modelos calibrados. Como se implementó luego de CNN y BERT (y sus resultados no se acercaban) se descartó.

4.1.4. RNN - LSTM

RNN permite mantener la información a lo largo de varios pasos (epochs) para ir aprendiendo sobre los mismos. Esto genera que el entrenamiento sea bastante lento en comparación a otros métodos y según los parámetros puede overfittear muy rápidamente. Se utilizó tanto LSTM como BiLSTM, donde esta última mostró mejores resultados ya que podía recordar tanto las palabras previas como posteriores de alguna en particular. Sin embargo, a pesar de haber modificado varios parámetros, layers y agregar o no features, no se logró superar un score de 0.80.

4.1.5. Logistic Regression

Se basa en la probabilidad de que el objeto a predecir perteneza o no a una clase en particular, para ello es importante pasar el texto a un vector numérico que lo represente. Se probaron 3 métodos distintos:

- TF-IDF
- CountVectorizer (BoW)
- HashingVectorizer

Después de las pruebas se puede ver que tanto TF-IDF como HashingV son buenas opciones (muy similares entre sí) mientras que CountV queda muy por detrás, lo cual podría deberse a la cantidad de palabras que existen en todos los tweets, dando así vectores muy poco densos.

4.1.6. SVM

Este método intenta separar los puntos mediante la figura más simple según la dimensión en la que se trabaja (2D - Línea // 3D - Hiperplano,etc...). De nuevo, es necesario convertir nuestros textos en puntos y para ello, como en el caso anterior utilizamos los 3 métodos. Al igual que el caso anterior obtuvimos un promedio de 0.79, con TF-IDF y HashingV a la cabeza.

4.1.7. Light GBM

Es un modelo de optimización de gradiente que utiliza árboles basado en algoritmos de aprendizaje. Se utilizaron dos formas de tokenizar el texto: TF-IDF y CountVectorizer, y luego se agregaron los features obtenidos. Se realizaron varias pruebas y búsquedas de optimización de parámetros, con y sin features, y todos los resultados dieron debajo del 0.79.

4.1.8. Tensorflow hub layers

Se probaron los embeddings 20-dim y 50-dim disponibles en la librería TensorFlow Hub en dos modelos secuenciales. Ambos tokenizadores se probaron con diversos parámetros y los resultados fueron muy bajos y tomaban numerosos epochs para mejorar la accuracy.

4.1.9. Character level CNN - Conv 1D

Esta red neuronal convolucional se focaliza en la clasificación de texto a nivel de caracteres. Consiste en la capa de embedding, una capa Conv 1D, max pooling y una capa densa. Se implementó fácilmente con el tokenizador que provee la librería Keras de TensorFlow y se probaron diferentes hiperparámetros. Las pruebas realizadas no superaron el 0.77 de accuracy con un subset de prueba y los submits a kaggle dieron aún más bajo.

4.2. Modelos Elegidos

Los siguientes modelos dieron resultados considerablemente mejores que los mencionados previamente. Éstos son considerados como los que mejor performaron a lo largo de todo el trabajo práctico y son mencionados a continuación.

4.2.1. Redes Convolucionales: primeros pasos

Para estos modelos se escaló de menor a mayor agregando mayor complejidad a medida que se probaban distintos tipos de embeddings, filtros y capas. Se puede encontrar en esta publicación una pequeña guía de CNNs para clasificación de texto.

Se comenzó probando modelos básicos:

CountVectorizer + Logistic Regression: Se trata de un modelo básico de BoW seguido de una instancia del modelo lineal LogisticRegression provisto por la librería Sklearn. Este modelo permitía modificar pocos parámetros tales como el rango de n-gramas a utilizar.

CountVectorizer + Capas Densas de Keras: Este modelo parte de la base del anterior pero comienza a utilizar las herramientas provistas por Keras. Se utilizaron dos capas Densas de unas pocas neuronas. Como resultado overfitteaba muy rápido (al tercer epoch).

Tokenizer + Embedding + Maxpool: A partir de este momento se comenzó a utilizar una forma de tokenizar el texto que es ampliamente utilizada, a través del Tokenizer. Este vectoriza el texto, convirtiendo cada palabra en una secuencia de enteros (cada uno siendo el índice de un token en un diccionario). Posteriormente se realiza el padding que deja todos los vectores de un mismo largo.
Una vez realizado esto, utilizando las herramientas de Keras, se realizó el embedding, max pooling y aplicación de capas densas. A través de todo el trabajo se utilizó una última capa densa de 1 neurona con activación sigmoide que permitió realizar las predicciones binarias.

Tokenizer + Embedding utilizando GloVe + Maxpool: Este modelo es muy similar al anterior con la única diferencia que el embedding se realizó utilizando GloVe.

La elección de GloVe se basó en una evaluación de las palabras pre entrenadas provistas. Se comparó su cobertura con el embedding de FastText llamado Crawl.
Ésta evaluación arrojó como resultado que GloVe cubría un 52.17 % del vocabulario y un 82.73 % del texto en el set de Train, mientras que cubría un 57.21 % del vocabulario y un 81.85 % del texto en el set de Test. Análogamente, FastText cubría 51.63 % y 81.88 % para el set de Train, y 56.55 % y 81.12 % para el set de Test. Esto significa una diferencia mínima pero considerable para la realización del trabajo. Cabe destacar que esta cobertura es la reflejada sobre el texto sin procesar de ambos datasets, que podría luego ser extrapolada para el texto procesado. Adicionalmente, la decisión de la utilización de GloVe no fue enteramente por la cobertura de textos sino que el embedding que finalmente sería utilizado fue entrenado con palabras de Twitter (2 Billones de tweets) que oportunamente está relacionado al tópico de este trabajo.

Los modelos básicos mencionados previamente promediaron un score entre 0.795 y 0.809. Para ser básicos son relativamente buenos y muy rápidos. Sin embargo, se comenzó a utilizar modelos más complejos que permitieron obtener mejores resultados.

4.2.2. Redes Convolucionales: Conv1D

Estos modelos permitieron obtener los mejores resultados durante la mayoría del trabajo, hasta la utilización de BERT.

La CNN es un tipo de Red Neuronal Artificial con aprendizaje supervisado que procesa sus capas imitando al cortex visual del ojo humano para identificar distintas características en las

entradas que en definitiva hacen que pueda identificar objetos y "ver". Para ello, la CNN contiene varias capas ocultas especializadas y con una jerarquía: esto quiere decir que las primeras capas pueden detectar líneas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro o la silueta de un animal.

Las CNN son desarrollos de machine learning que revolucionaron la clasificación de imágenes ya que permite extraer features de dichas imágenes y utilizarlas en redes neuronales. Las propiedades que las hacen útiles en procesamiento de imágenes también son útiles para procesamiento de secuencias de texto, ya que puede detectar ciertos patrones.

Se comenzó utilizando modelos básicos que consistían en Tokenizer + Embedding utilizando GloVe + Conv1D + Maxpool. El embedding utilizado fue el de Twitter de 27 billones de tokens y 100 dimensiones. A partir de ésta adición los resultados mejoraron progresivamente en el tiempo a medida que los integrantes del grupo aprendieron a refinar este tipo de modelo.

Progresivamente se fueron añadiendo features al modelo, lo cual mejoró los resultados notablemente. Con el modelo en este estado se realizaron diversas pruebas:

Conv1Ds *en serie*: Esta forma de armar el modelo consistió en utilizar una o más redes convolucionales cuya salida es la entrada de la siguiente, similar a un circuito eléctrico en serie.

Multi-Channel CNN *(Conv1Ds en paralelo)*: Esta forma de armar el modelo consistió en utilizar dos o más redes convolucionales seguidas de una etapa de max pooling que luego serían concatenadas para luego pasar por la capa densa del modelo. En general, no hubo gran diferencia entre esta manera de organizar las redes y la manera *en serie*. Sin embargo, esta forma fue la que mejor resultados obtuvo.

Multi-Channel CNN con multi-word embedding: Esta forma de armar el modelo consistió en utilizar redes convolucionales en paralelo pero además se le añadieron múltiples embeddings distintos para cada par de redes (cada una de ese par en paralelo). Se utilizaron los embeddings de GloVe de Twitter de 27 billones de tokens, de Wikipedia de 6 Billones de tokens con 200 dimensiones y el embedding de 300 dimensiones de 840 Billones de tokens. Aunque este modelo fue costoso en cuanto a espacio (el embedding de 300 dimensiones pesaba 5,5GB) y tiempo de entrenamiento, y dado su aparente complejidad, los resultados no fueron mejores que con cualquier otro modelo de redes convolucionales probado.

Conv1D + LSTM: Este modelo consistió en utilizar una red convolucional previa a la LSTM, con el objetivo de agilizar el tiempo de entrenamiento de la LSTM. No arrojó resultados prometedores con lo cual se descartó rápidamente.

En general todos los modelos de redes convolucionales performaron muy bien a través de todo el trabajo, considerando sus altibajos, y llevaron poco tiempo de entrenamiento y predicción lo que permitió realizar una cantidad considerable de búsqueda de hiper-parámetros en busca de un mejor score (cuyos detalles serán explayados en su respectiva sección). Aunque los resultados fueron buenos, éstos fueron muy variables entre distintas ejecuciones de la misma configuración del modelo.

Adicionalmente, este tipo de modelos permitió un muy amplio espectro de configuraciones y distintas combinaciones posibles lo cual lo convirtió en el modelo que más profundamente fue analizado durante el trabajo con el objetivo de mejorar su performance de score en Kaggle. Esto permitió incorporar variados conocimientos sobre estos tipos de modelos.

Entre los métodos adicionales que fueron probados se utilizaron:

Gaussian Noise: El Gaussian Noise consiste en añadir ruido con una media preestablecida a la entrada de capas densas o embeddings con el objetivo de prevenir y aminorar el overfitting.

Layer weight constraint: Consiste en añadir cierto peso a la capa densa con el objetivo de reducir su media según el parámetro utilizado.

Kernel regularizers: Para estos modelos se eligió regularización L2 basada en la regresión ridge la cual regulariza el modelo resultante imponiendo una penalización al tamaño de los coeficientes de la relación lineal entre las características predictivas y la variable objetivo. La norma L2 es equivalente a la distancia Euclidea del vector al centro de coordenadas, por lo que la penalización es proporcional a los cuadrados de los coeficientes.

Dropout: Consiste en no tomar una cierta cantidad de datos durante el entrenamiento para reducir el overfitting.

Capas de Activación: Consistió en añadir una capa de activación extra subsiguiente a las redes convolucionales.

Mejor modelo de redes convolucionales (Conv1D)

El modelo que mejor performó fue uno que consistió en dos redes convolucionales en paralelo con una posterior capa de activación, seguido de max pooling, que luego fueron concatenadas junto con los features de texto, keywords y location utilizando sólo sus coordenadas. Finalmente se tenía una capa densa con activación Relu. La cantidad de parámetros a entrenar revelados por la función *summary* fueron 1.568.092.

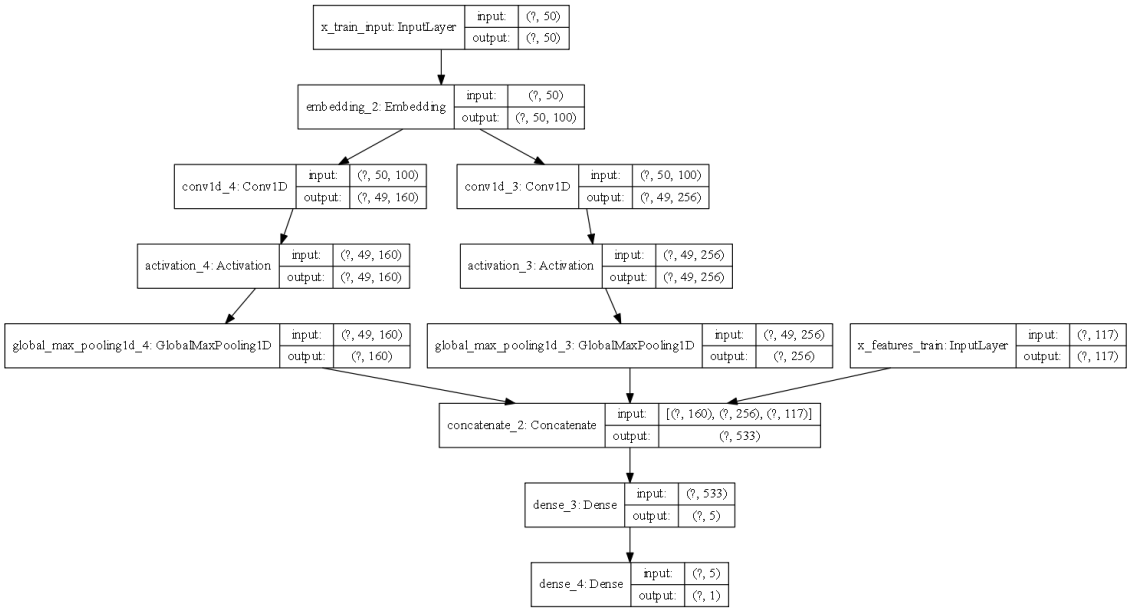
Se logró un score en Kaggle de **0.82715** y fue el mejor logrado por una red convolucional en este trabajo. Posteriormente se comenzaría a utilizar BERT, cuyos resultados fueron mejores inmediatamente.

Esta configuración de la red fue hallada gracias a una Random Search de hiper-parámetros de cantidad de filtros de las redes, cantidad de neuronas de la capa densa y cantidad de mini batches de entrenamiento.

Durante todo el trabajo, consistentemente la cantidad de filtros para redes neuronales dieron mejor resultado con potencias de 2 (128, 256) altas. No existe una razón particular pero en general los resultados fueron mejores al utilizar valores entre 100 y 300 filtros en las redes, así como también los valores de las capas densas dieron mejores resultados alrededor de 100 neuronas, con excepción de éste caso que se utilizaron sólamente 5.

Los features extras utilizados fueron los de texto, los embeddings de keywords y las coordenadas de las locations. No se utilizaron los embeddings de las locations. Esta elección de features aporta una gran cantidad de información ya que los embeddings de la columna locations aportan muy poca información según el feature importance, en contraste con las coordenadas, que figuran dentro de los que más aportan.

A continuación se muestra una representación gráfica de la configuración del modelo final de Conv1D.



Plot del modelo utilizando Conv1D que mejor performó utilizando la herramienta de ploteo de Keras.

5. Parameter Tuning

La velocidad de entrenamiento de las redes convolucionales Conv1D permitió realizar diversas búsquedas tanto GridSearch como RandomSearch, que permitieron mejorar el resultado en Kaggle en múltiples oportunidades.

Como fue mencionado previamente, el modelo performó mejor utilizando una alta cantidad de filtros y en repetidas ocasiones con una cantidad potencia de 2 (128, 256). No existe razón aparente pero consistentemente generó resultados buenos. Las RandomSearch produjeron resultados muy variados y produjeron resultados muy buenos en varias oportunidades. Por lo general la elección de hiperparámetros es lo más difícil del *parameter tuning*. Si se hubiera querido probar una cantidad considerable de combinaciones de filtros, batch sizes y cantidad de neuronas en capas densas la GridSearch hubiera requerido tiempos por encima de un día para ejecutarse y aún así esto no significa que arrojara resultados prometedores.

Por lo general las configuraciones óptimas de hiperparámetros dependen del dataset que se utilice en cada caso. Esto se debe a que además, existen interacciones complejas entre dichos hiperparámetros que pueden generar que si se *prueba* uno a la vez, al modificar otro pueda hacer que afecte el recién probado.

Si se tiene experiencia previa con un modelo o dataset, se puede saber de antemano un subespacio donde los hiperparámetros sean más efectivos, pero éste no era el caso, ya que se comenzó de cero. Es por esto que se suele comenzar con una RandomSearch de amplias combinaciones para aproximar subespacios donde los hiperparámetros funcionan mejor. RandomSearch es mucho más eficiente que GridSearch para comenzar ya que puede encontrar con mayor efectividad aproximaciones a combinaciones de hiperparámetros. Ésta publicación explica con más detalles lo recién mencionado.

Adicionalmente, tanto como para las búsquedas de hiperparámetros como para las pruebas individuales de distintos modelos se utilizó el Early Stopping como ayuda para entender más la cantidad de entrenamiento requerido por un modelo. Ésta herramienta puede monitorear varios aspectos del entrenamiento de un modelo y es configurable para que cese su entrenamiento si una variable comienza a comportarse de manera contraria a lo esperado. Se utilizó para monitorear la *validation loss* que representa la pérdida sobre el set de validación utilizado al entrenar un modelo. Si la validation loss comienza a aumentar, significa que el modelo está comenzando a overfittear, y si decrece, significa que el modelo está aprendiendo como es esperado.

A través de todo el trabajo se pudo llegar a un cantidad de epochs que optimizaban los resultados del entrenamiento tanto como para redes convolucionales como para BERT, la cual fue de 2 (dos). Esta cantidad fue incrementada en 1 para los casos que se utilizó el regularizador de kernel L2 sobre la capa densa tanto para redes convolucionales como para BERT.

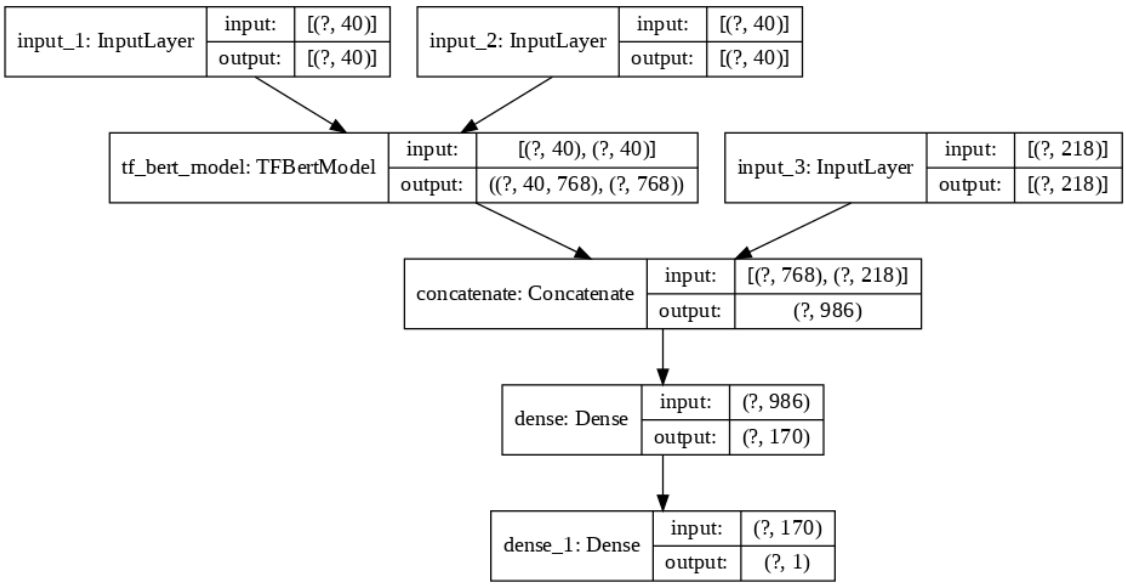
6. BERT como mejor modelo

BERT es una técnica de preentrenamiento para procesamiento de lenguaje natural desarrollada por Google. Google utiliza este modelo para entender las búsquedas de los usuarios. Es bidireccional ya que la capa de self-attention ejecuta self-attention en las dos direcciones, entendemos esto como computar la importancia sobre cada palabra de la frase. La meta es buscar las dependencias entre las palabras de las oraciones y capturar la estructura interna de las mismas. Tiene una forma interesante de entrenar el modelo enmascarando ciertos tokens, de esta forma el modelo se concentra en el contexto, y por otro lado, prediciendo la siguiente frase, gracias a esto el modelo aprende a relacionar oraciones. A diferencia de word2vec y GloVe que solo generan una representación de cada palabra en el vocabulario, BERT tiene en cuenta el contexto.

El armado del modelo utilizando BERT no conllevó un gran esfuerzo por fuera de los parámetros y capas preestablecidas, lo que permitió comenzar a trabajar con él de manera rápida. Desde el primer momento los resultados obtenidos fueron superiores a los de CNN - Conv1D. De esta manera, se procedió a buscar los mejores parámetros para mejorar la accuracy sobre el set de prueba.

6.1. Mejor Submit

Por sus características, BERT no conlleva un amplio espectro de combinaciones y configuraciones como sucedía en las redes convolucionales, con lo que con pocas iteraciones de búsqueda se obtuvieron resultados muy favorables instantáneamente. Adicionalmente, la publicación citada más arriba proporciona un subespacio acotado de hiperparámetros recomendados, que en este caso fueron de gran ayuda. El modelo que dió mejores resultados fue el siguiente, utilizando sólamente una capa Densa de 110 neuronas y un batch size de 19:



Plot del modelo utilizando BERT que mejor performó utilizando la herramienta de ploteo de Keras.

7. Conclusiones y Resultados obtenidos

Como fue mencionado previamente, el modelo que mejor resultados produjo fue el de BERT, seguido por las redes convolucionales Conv1D. El score submitteado a Kaggle fue de **0.84308** con el de BERT mientras que el mejor resultado de las convolucionales fue de **0.82715**, que fue considerablemente alto.

Cronológicamente, BERT fue el último y mejor performante modelo utilizado, que además requirió poca configuración de hiperparámetros para obtener resultados mejores a las redes convolucionales. Consideramos que de haber utilizado BERT antes nos hubiera privado de aprender en profundidad sobre el amplio espectro de configuraciones y combinaciones que proveen las redes convolucionales a través de Keras. En contraste, hubo menor tiempo de optimización del algoritmo de BERT para lograr mejorar su score.

A diferencia de los otros algoritmos y modelos implementados, BERT lleva mucho tiempo de entrenamiento y predicción lo cual hace más difícil optimizar sus -pocos- hiperparámetros en comparación con modelos como las redes convolucionales. La comparación en tiempos es de dos horas versus aproximadamente treinta segundos.

A modo de conclusión, pudimos comprobar empíricamente que Machine Learning tiene una curva de aprendizaje compleja al principio pero una vez aprendido lo básico se vuelve más rápido aprender nuevos algoritmos relacionados al tema.

Se pudo observar que con el agregado de features y utilizando el texto procesado el score mejoró notablemente. Sin embargo se pudo notar que las repetidas ejecuciones de un modelo pueden arrojar distintos resultados, lo cual puede depender de distintos factores entre los cuales está la forma de seleccionar subsets de datos que puede realizar el algoritmo.

Por otro lado, consideramos que aunque se probaron una gran cantidad de opciones de modelos, faltó incursionar en mayor proporción a modelos de atención como lo es BERT y modelos más profundos de redes neuronales. Sin embargo, a partir de la implementación de esta serie de algoritmos se logró un nivel de predicción relativamente alto.