75.06/95.58 Organización de Datos

Primer Cuatrimestre de 2020

Trabajo Práctico 2: Machine Learning

Fecha de entrega: 06/08/2020

Grupo n° 16 Grupo python fiuba

Integrantes:

Apellido y nombres	Padrón
Balladares Alejandro	101118
Del Pino Cardenas Ronnie	93575

Link al repositorio de Github:

https://github.com/delpinor/datos7506

Índice

1. Introducción

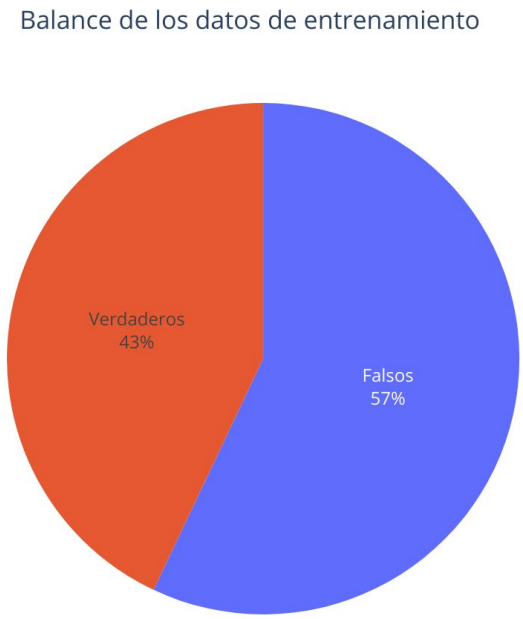
El objetivo del presente trabajo es intentar predecir si el contenido generado por usuarios de la red social Twitter está basado en hechos reales o no. Los algoritmos utilizados fueron tanto tradicionales como así también los más sofisticados. Se usaron algoritmos como Naïve Bayes y KNN obteniendo resultados modestos. De la misma forma al utilizar algoritmos basados en árboles no se lograron grandes resultados más bien levemente superiores a los algoritmos tradicionales.

Con respecto a los algoritmos basados en aprendizaje profundo basados en redes neuronales, en diversos papers o publicaciones se ha demostrado que las redes neuronales convolucionales aprenden características locales de las frases, mientras que las redes neuronales recurrentes (RNN) pueden aprender temporal dependencias en datos secuenciales, dado que las características de los tweets, en general, es de tener pocas palabras o frases(en algunos casos solo 'stopwords'), casi necesariamente nos vimos obligados a usar incrustaciones de palabras como Glove, FastText, wordvec, etc. logrando mejores resultados.

Finalmente para poder usar los algoritmos mencionados anteriormente, se hizo un pre procesamiento exhaustivo que consistió en eliminar ruido del texto original dado que contenían caracteres o palabras sin información relevante.

2. Balance de las clases

Para evitar realizar una clasificación desequilibrada, analizamos las clases para ver si hay una clase dominante en los datos de entrenamiento.



Podemos apreciar que los datos están balanceados entre ambas clases: Tweets Verdaderos y Falsos. En principio no será no será necesario el uso de técnicas como oversampling, undersampling y weighting para compensar la clase minoritaria.

3. Procesamiento de texto

Nuestro set de datos tiene características particulares que agregan mucho ruido a nuestros clasificadores. La tarea principal es la extracción de características que permitan representar las clases que queremos predecir.

El campo keyword no agrega información relevante e incluso no necesariamente está relacionado con el sentido que trata de comunicar el usuario que escribió el tweet. Debido a esto decidimos no tomarlo en cuenta.

Del análisis exploratorio hecho en el TP1 el campo location no mostraba una tendencia que haga inclinar sobre alguna de las clases a predecir, a raíz de esto decidimos no tomarla en cuenta.

3.1 Signos de puntuación

El texto a analizar contenía muchos signos de puntuación, algunos usados correctamente, otros eran a causa a la codificación(html, emojis, etc). De hecho el signo más usado es una barra('/') que en un texto redactado en un medio de comunicación o libro es menos común que otros de uso más frecuente. Debido a esto fueron reemplazados por espacios dado que existían palabras separadas por barras o guiones.

Ej: Typhoon-Devastated, In-game.

Top 10 signos más comunes

Signo de puntuación	Cantidad
/	14585
.	11696
:	6910
#	3403
'	3157
?	3126
@	2759
-	1753
!	1173
_	863
;	587
&	457
)	373
(	349
*	179

3.2 Stop words

Haciendo un análisis exhaustivo de los stops words detectamos que el eliminarlos significaba perder características importantes en nuestros datos de entrenamiento dado que en muchos tweets gran parte de su contenido estaban formados por estos e incluso algunos casos el 100% de las palabras estaban conformados por stopwords.

Otra de las características representativas obtenidas a partir del Trabajo Práctico 1, es que algunos stopwords en los tweets verdaderos tienen menos presencia en los falsos y viceversa.

Stopwords comunes en tweets verdaderos

Stopwords comunes en tweets falsos

3.3 Análisis del léxico

Con la ayuda de la biblioteca NLTK detectamos que existían palabras que no eran encontradas, debido a esto asumimos que requerían un procesamiento manual. También filtramos palabras no encontradas en los vectores pre entrenados(Embeddings) para su análisis.

Top 10 palabras no encontradas en el diccionario

Palabra	Cantidad
legionnaires	62
thunderstorm	39
typhoondevastated	25
terrorismism	18
electrocuted	14
philippines	14
quarantined	13
mediterranean	12
palestinian	12
repatriated	10

En la tabla podemos apreciar que algunas palabras están concatenadas por lo que hay que reemplazarlas por su valor correcto. Ej. ***'typhoondevasted'*** por ***'typhoon devastated'***.

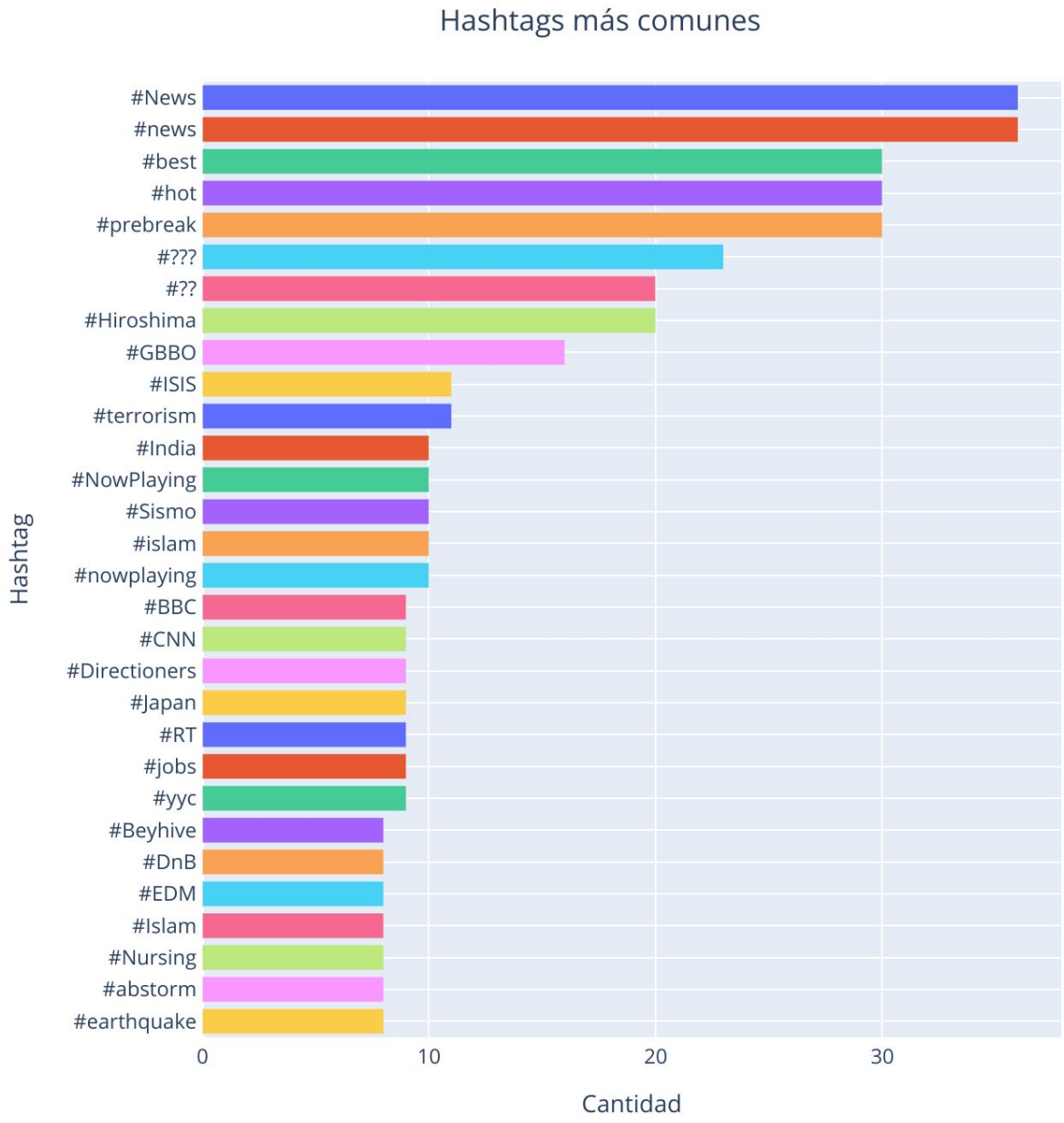
De la misma forma se reemplazaron stopword cortos por su forma completa. También se reemplazaron abreviaciones o palabras usadas vulgarmente por su significado:

Abreviaciones

En este gráfico de ejemplo vemos algunas siglas referidas a medios de comunicación y en otros casos palabras como 'pm' de datos del tiempo o 'pic' relacionado con fotografías. En

algunos casos también se usan monosílabos o usadas en comunidades. Ej.***'u'*** en vez de ***'you'*** , ***'da'*** en vez de **'the'.** En todos los casos fueron reemplazados por su valor real.

Con respecto a los Hastags decidimos usarlos ya que en la mayoría de los casos su uso está relacionado con la importancia o propósito del tweet:



Los tweets con hashtag compuesto como por ejemplo **#NowPlaying** fueron reemplazados por '**now playing**'. Esto para eliminar ruido en nuestros clasificadores.

Con respecto a los **Links** y los **@usuarios** hemos decidido eliminarlos ya que no contenían información representativa.

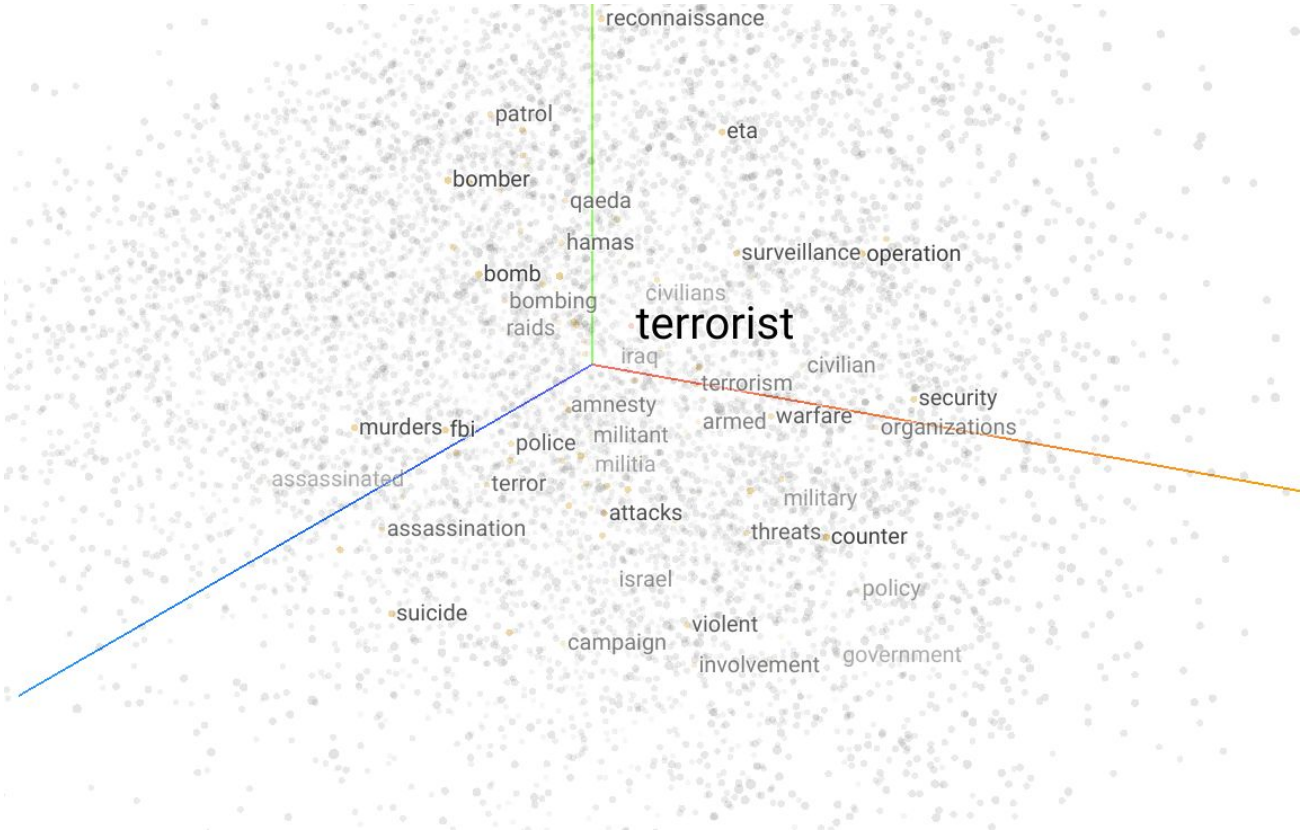
Los números y fechas decidimos no eliminarlos ya que se repetían patrones en los tweets.

4. Extracción de características

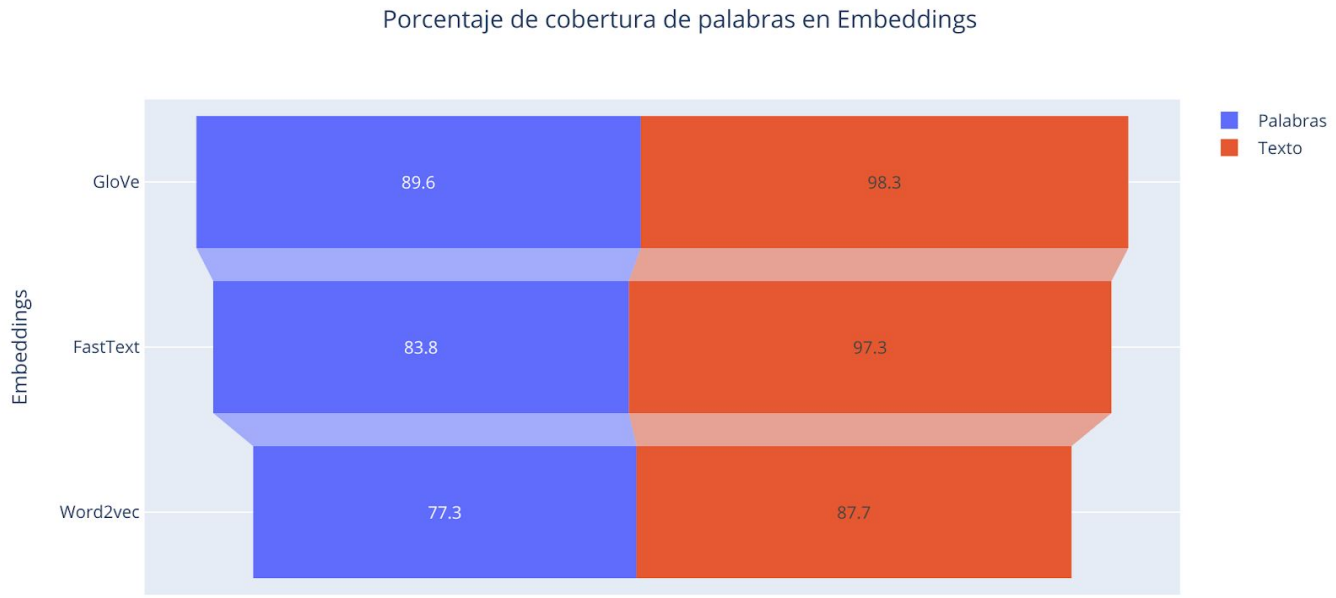
Para los algoritmos tradicionales y los basados en árboles representamos los documentos en forma de vectores(BOW) y TF-IDF. En todos los casos usamos herramientas provistas por la librería Sklearn tanto para tokenizar como para calcular los valores de TF e IDF.

Para los algoritmos basados en Deep Learning usamos incrustaciones de palabras(Embeddings) dado que las palabras que tienen similaridad tiene una representación similar. Trabajamos con incrustaciones como Word2Vec, FastText, GloVe previamente entrenadas con léxicos provistos de GoogleNews, Wikipedia, Twitter, etc.

Como ejemplo, en el siguiente gráfico se puede apreciar el beneficio de usar vectores pre entrenados ya que vemos que para el caso de la palabra '**terrorist**'(muy frecuente en los datos de entrenamiento) nos muestra el contexto o palabras relacionadas con esta, también muy frecuentes como 'police', 'bombing', etc.

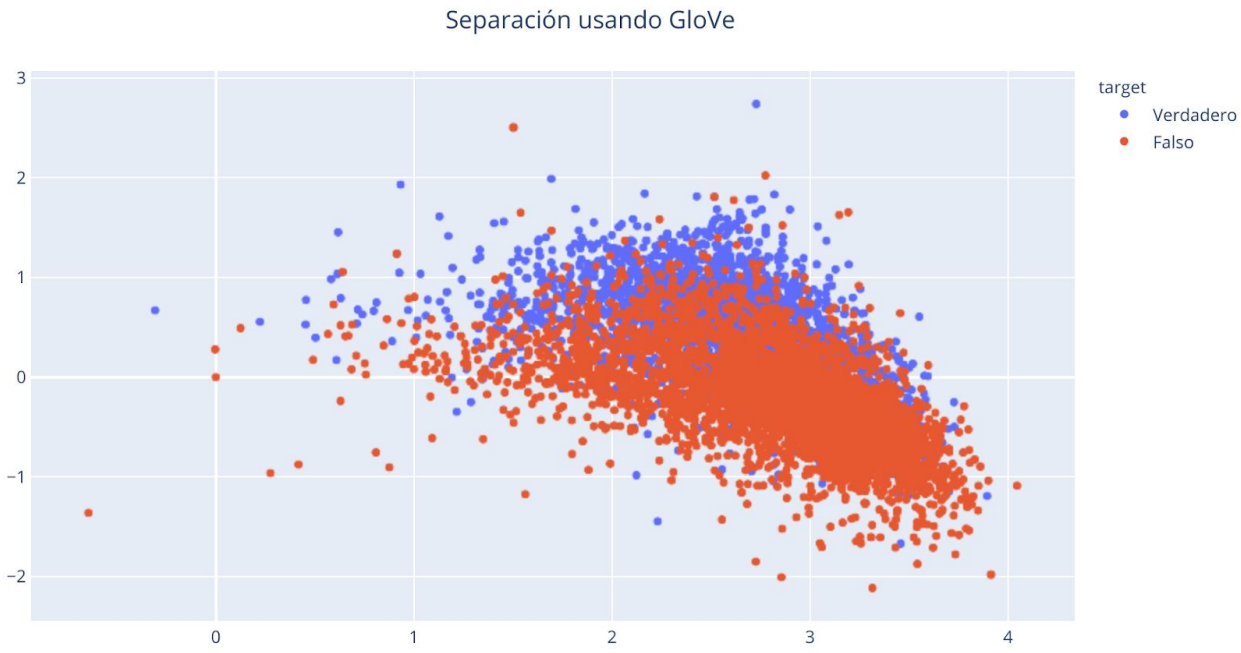


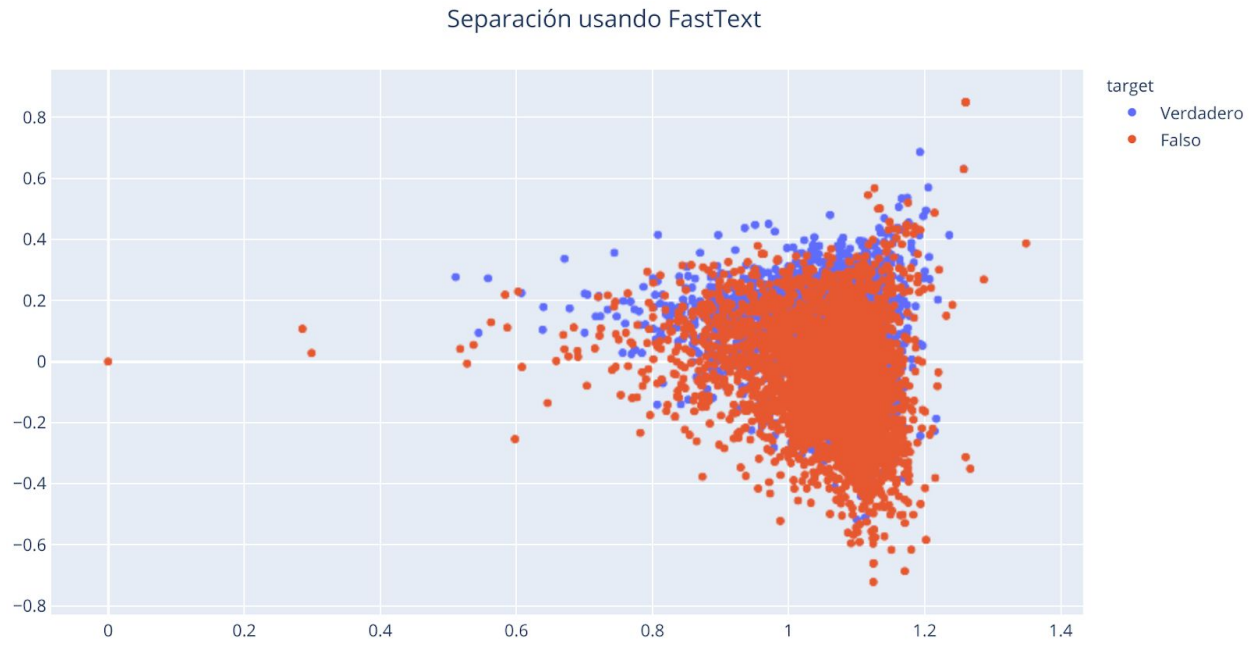
De todos los vectores pre entrenados que probamos, obtuvimos mejores resultados con GloVe de 300 dimensiones , 840B tokens. En parte debido a que estos cubrían la mayor parte de las palabras y el texto como podemos apreciar en el siguiente gráfico:

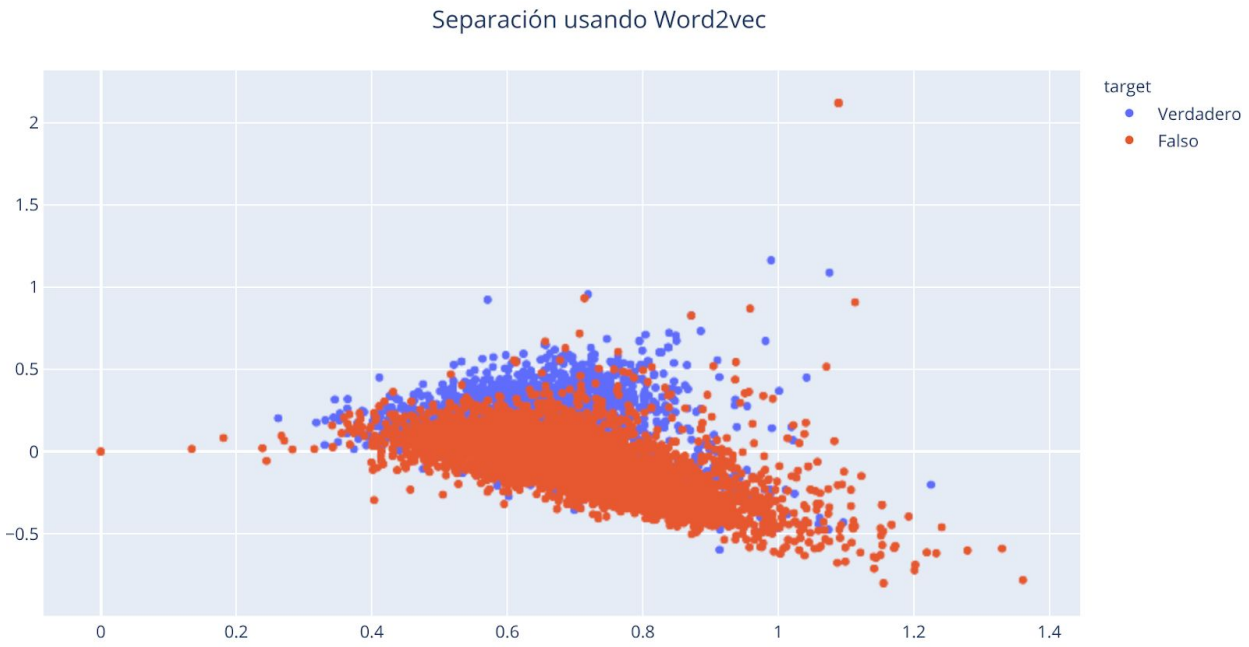


Usando incrustaciones Glove cubrimos el 98% del texto y el 89% de las palabras presentes en el set de entrenamiento.

Usando SVD proyectamos a 2 dimensiones y graficamos la separaciones para los tres tipos de embeddings:







Se aprecia que en el primer caso(usando GloVe) hay una mejor separación de los las clases. Esto nos confirma la el supuesto de que con Glove tendríamos mejores resultados por contener vectorizados gran parte de las palabras utilizadas en los Tweets.

Más adelante mencionaremos las incrustaciones BERT. Lo que hace distinto BERT es que procesa las palabras en el contexto de una oración, en lugar de palabra por palabra.

3.5 Location y otras ubicaciones

Otro factor a tener en cuenta durante el desarrollo fue que varios tweets contenían ubicaciones mal escritas o escritas en forma abreviada. Ej. uno de los más comunes es escribir “US” o “USA” al que transformamos en “United States” entre otros. Al no haber una librería o archivo que contuviera esas abreviaciones, tuvimos que buscar todas las apariciones y agregarlas manualmente al diccionario para ser reemplazadas.

Un detalle curioso que se observa es que en la mayoría de los casos el “Location” puede tener un lugar y pero el tweet puede tener otro diferente o un lugar más preciso. Ej. el location dice “Usa” pero el tweet habla sobre la ciudad de New York, por esta razón el location no fue considerado para las predicciones ya que puede ser vago o ambiguo

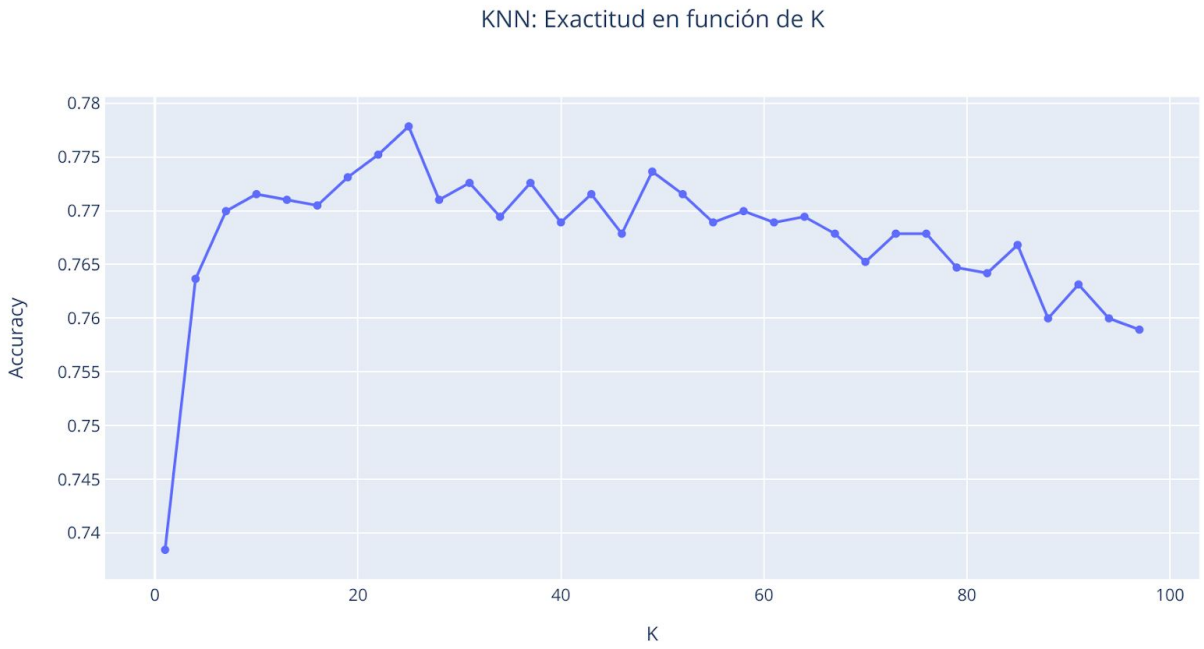
4. Algoritmos utilizados

En todos los algoritmos utilizamos el 25% del set de entrenamiento como datos de prueba. En los algoritmos tradicionales y basados en árboles usamos TF-IDF dado que con los mismos obtuvimos mejores resultados.

4.1 Algoritmos tradicionales

KNN

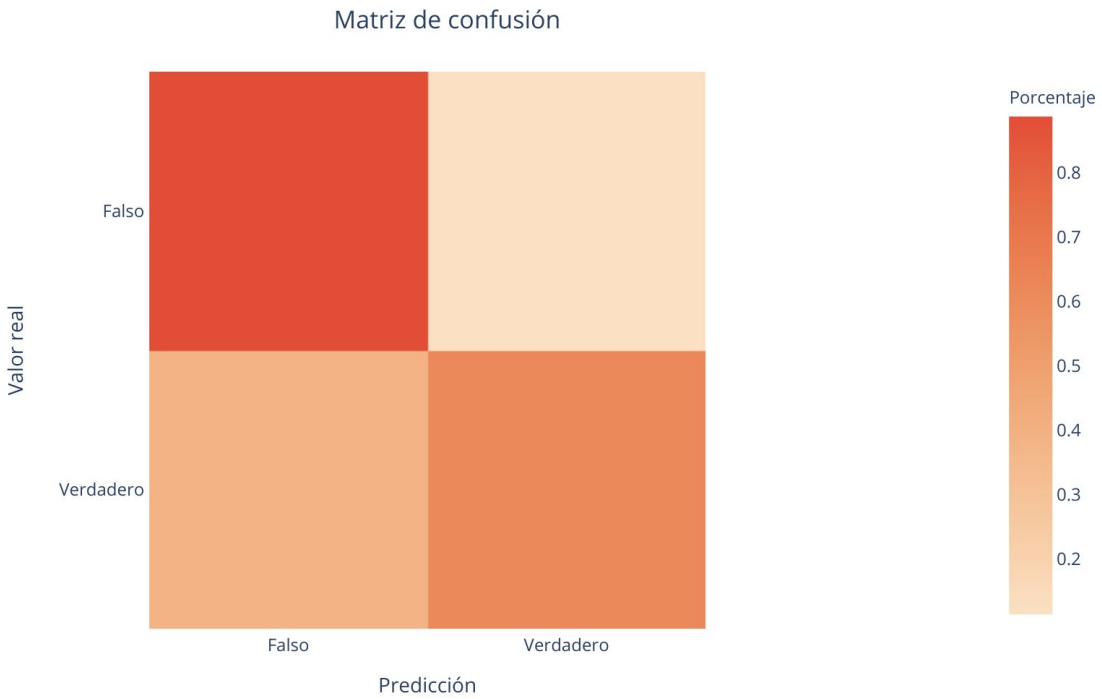
Para el cálculo de la distancia usamos Minkowski(se probaron con otras pero no logramos buenos resultados). Entrenamos nuestros datos en función del valor K(vecinos más próximos):



Con K=25 obtenemos mejores resultados. A continuación mostramos algunas métricas y la matriz de confusión para poder apreciar las dificultades para predecir cierta clase:

Predicción	Precisión	Recall	F1
Falso	0.74	0.89	0.81
Verdadero	0.81	0.61	0.70

ROC: 0.7490



Podemos apreciar que a nuestro algoritmo le está costando predecir los tweets reales, cerca del 40% de los tweets reales los predice como falsos.

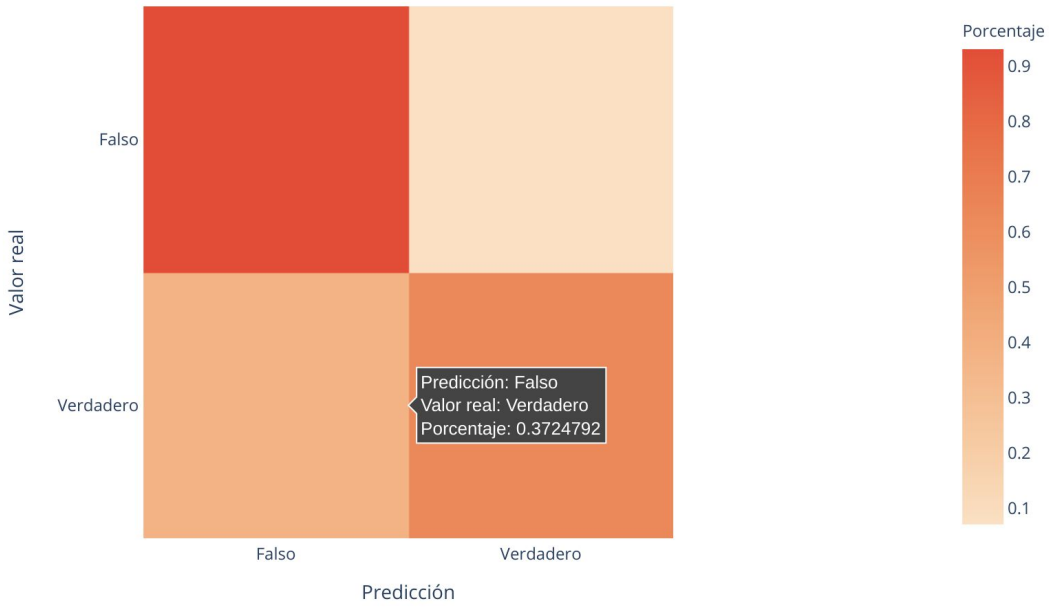
El score promedio obtenido con el set de prueba fue de **0.7757**.

Multinomial NB

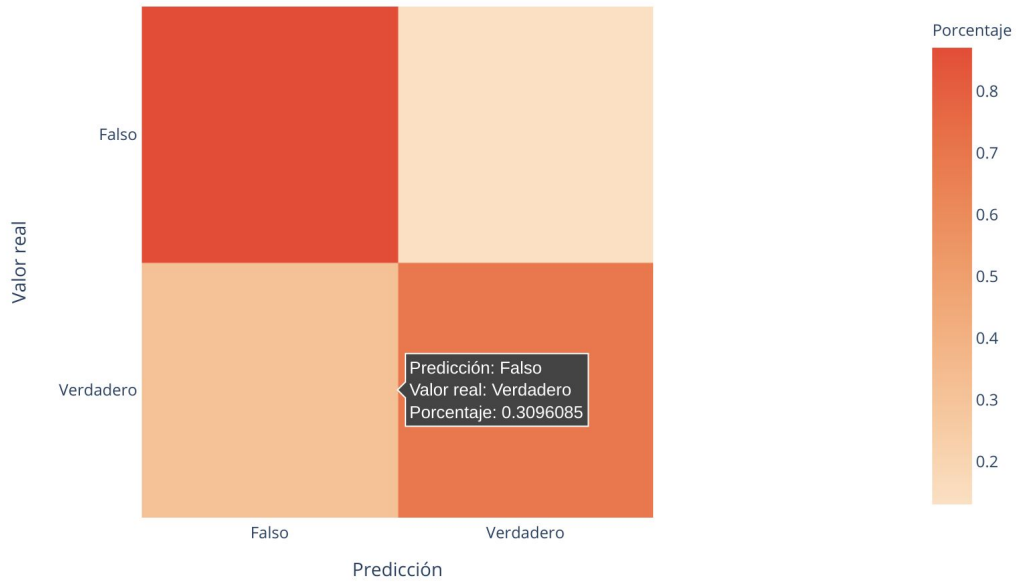
El rendimiento de este algoritmos es ligeramente superior a KNN. A continuación mostramos algunas métricas para el set de entrenamiento:

Predicción	Precisión	Recall	F1
Falso	0.76	0.93	0.84
Verdadero	0.88	0.63	0.63

ROC: 0.7784



Como se puede apreciar en el gráfico, a nuestro modelo le costó predecir los tweets verdaderos aunque en menor proporción que KNN. También realizamos pruebas con otro modelo basado en Naive Bayes que habitualmente se usa para datos pocos equilibrados: ComplementNB. Los resultados obtenidos fueron similares tomando como métrica ROC, el punto a favor es que predice mejor los tweets verdaderos como se aprecia en el siguiente gráfico:



Los scores obtenidos con el set de test:

MultinomialNB: 0.7946

ComplementNB: 0.7953

Logistic Regression

Buscando hiper parámetros , la cantidad de iteraciones no resultaba en un mayor exactitud del algoritmo por lo que se mantuvo en 100. Se buscó un valor apropiado para el parámetro C(regularización):

Luego afinamos la búsqueda entre 0 y 2:

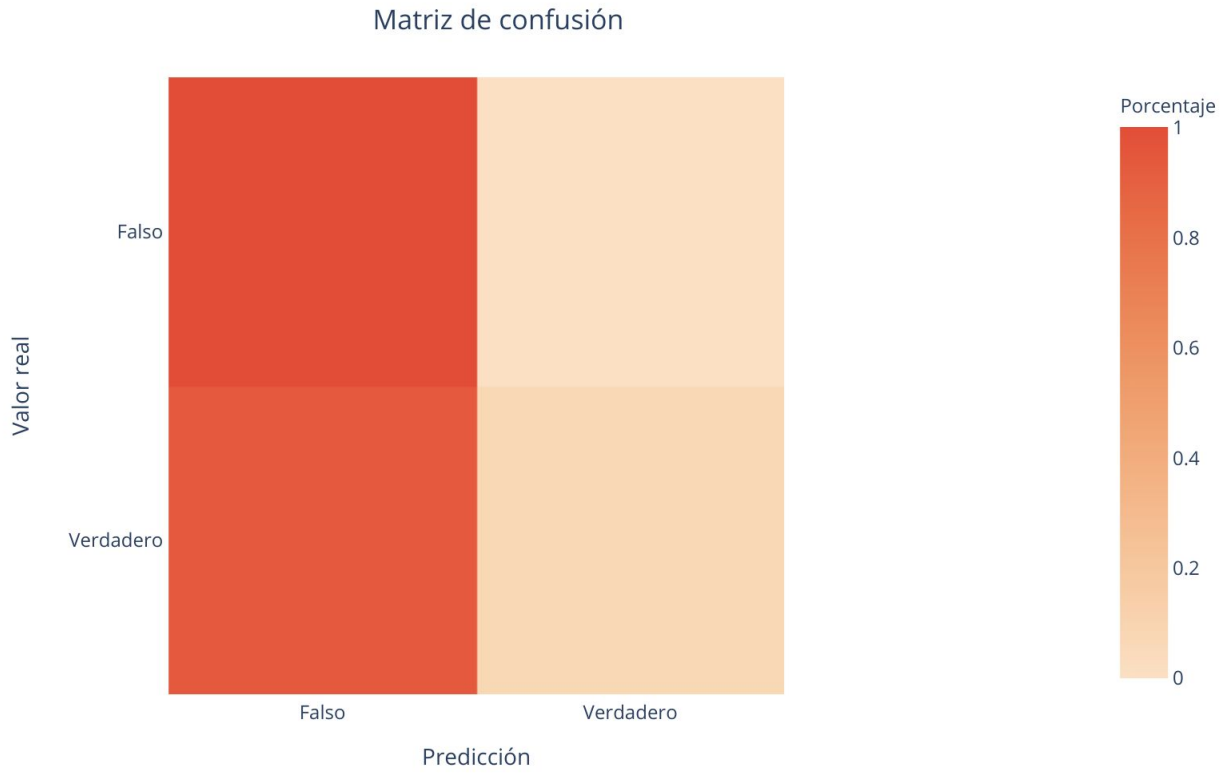
Obtenemos el valor **1.8** para **C** por lo que entrenamos con el set de entrenamiento completo y tratamos de predecir el set de test.

Obtenemos un score promedio de: **0.8029**

4.2 Algoritmos basados en árboles

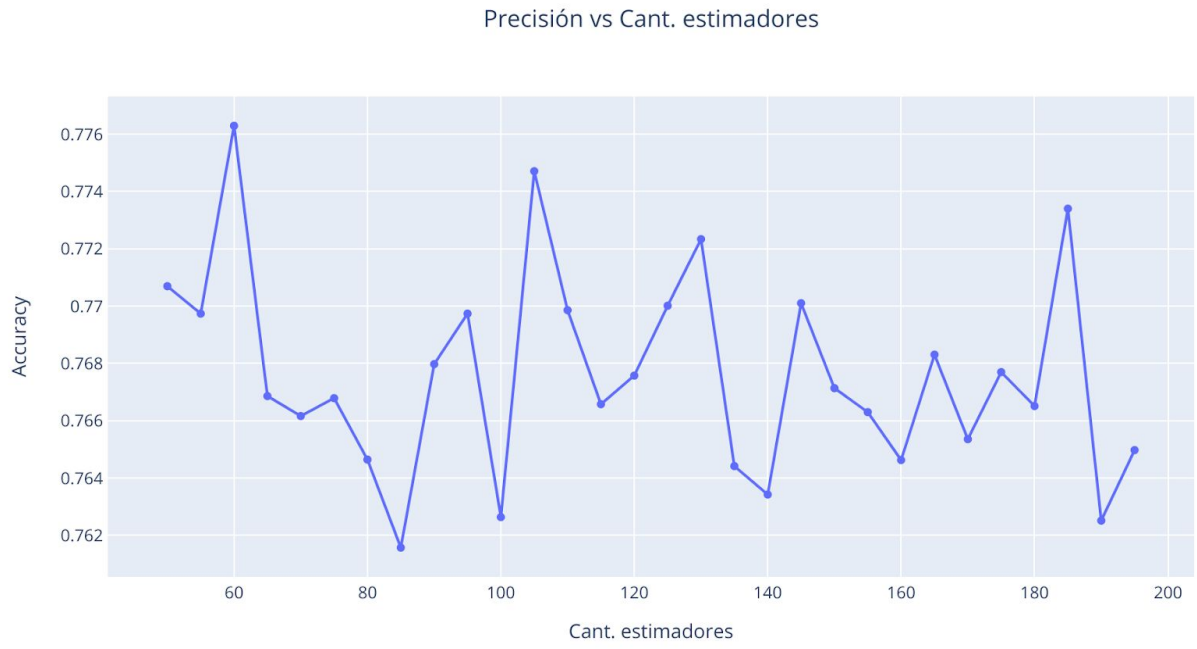
Random Forest

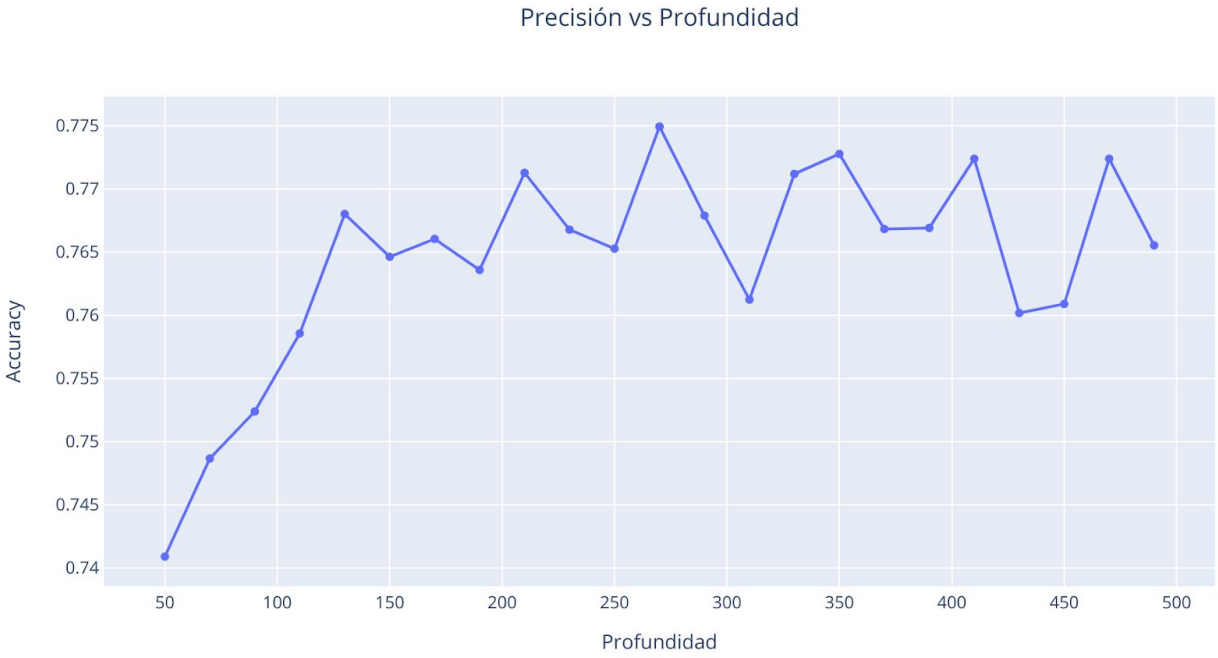
La búsqueda de hiper parámetros fué una tarea compleja, revisando documentación vimos que habitualmente se usan como profundidad máxima valores no muy grandes, debido a que estaríamos sobre-ajustando el modelo. Con valores entre 5 y 50 el algoritmo no podía predecir los tweets verdaderos, el 93% de los tweets verdaderos los predecía como falsos como se aprecia en el siguiente gráfico:



Usamos **GridSearchCV** para buscar algunos parámetros. Con estos parámetros obtuvimos un score(ROC) de **0.6635** con el set de entrenamiento y una precisión **0.6512** con el set de test.

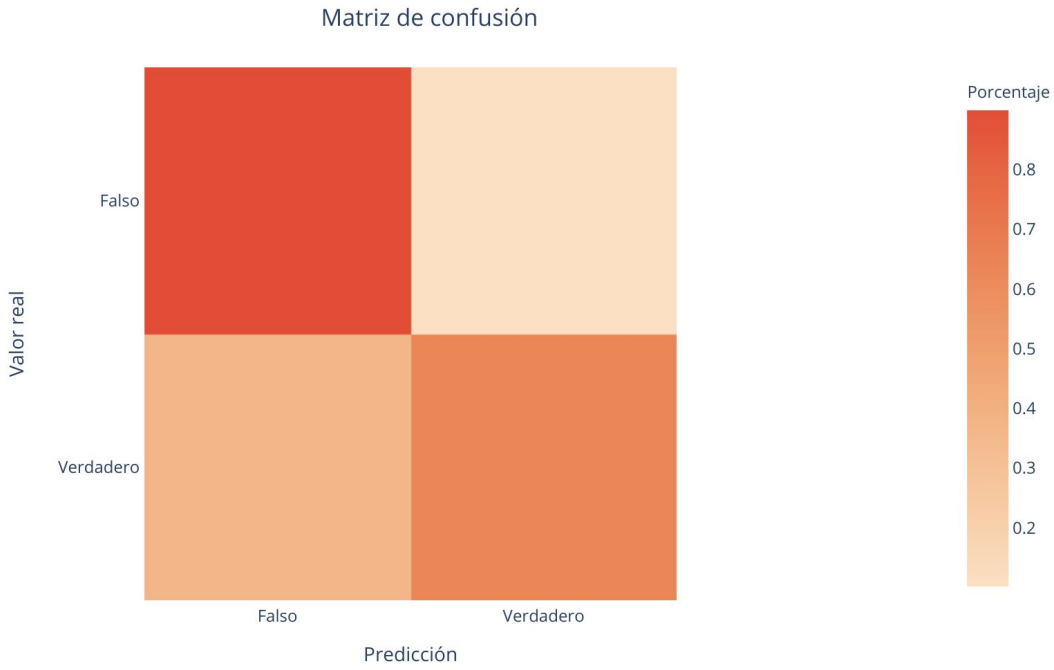
La cantidad de estimadores y la profundidad del árbol los buscamos manualmente:





Con estos nuevos valores obtuvimos un métrica de: **ROC: 0.7664**

En la siguiente imagen se aprecia que hubo una mejora en cuanto a la predicción de los tweets verdaderos como tal:



Luego de estos entrenamos el set de entrenamiento con estos valores y probamos el set de test con el que obtuvimos un score de **0.7863**, bastante mejor que el valor obtenido con GridSearchCV.

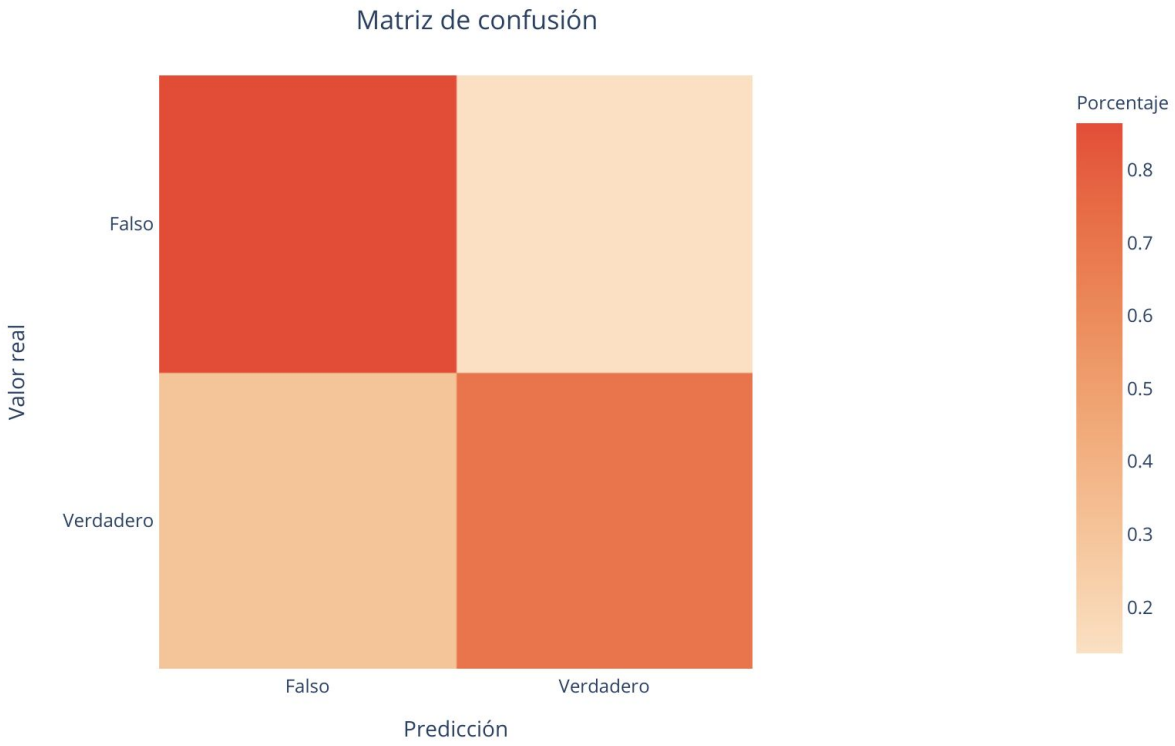
XGBoost

Usando GridSearchCV encontramos rápidamente hiper parámetros con los que obtuvimos buenos rendimientos. Con un árbol poco profundo(23) y 200 estimadores logramos los mismos resultados que Random Forest. Algunas métricas:

Predicción	Precisión	Recall	F1
Falso	0.78	0.86	0.82
Verdadero	0.80	0.70	0.75

ROC: 0.7827

No ajustamos más los valores de profundidad y estimadores ya que el modelo empezó a sobre-ajustar. Mostramos la generalización del algoritmo con la matriz de confusión:



Con estos valores obtuvimos un score de 0.7814 con el set de prueba.

Light GBM

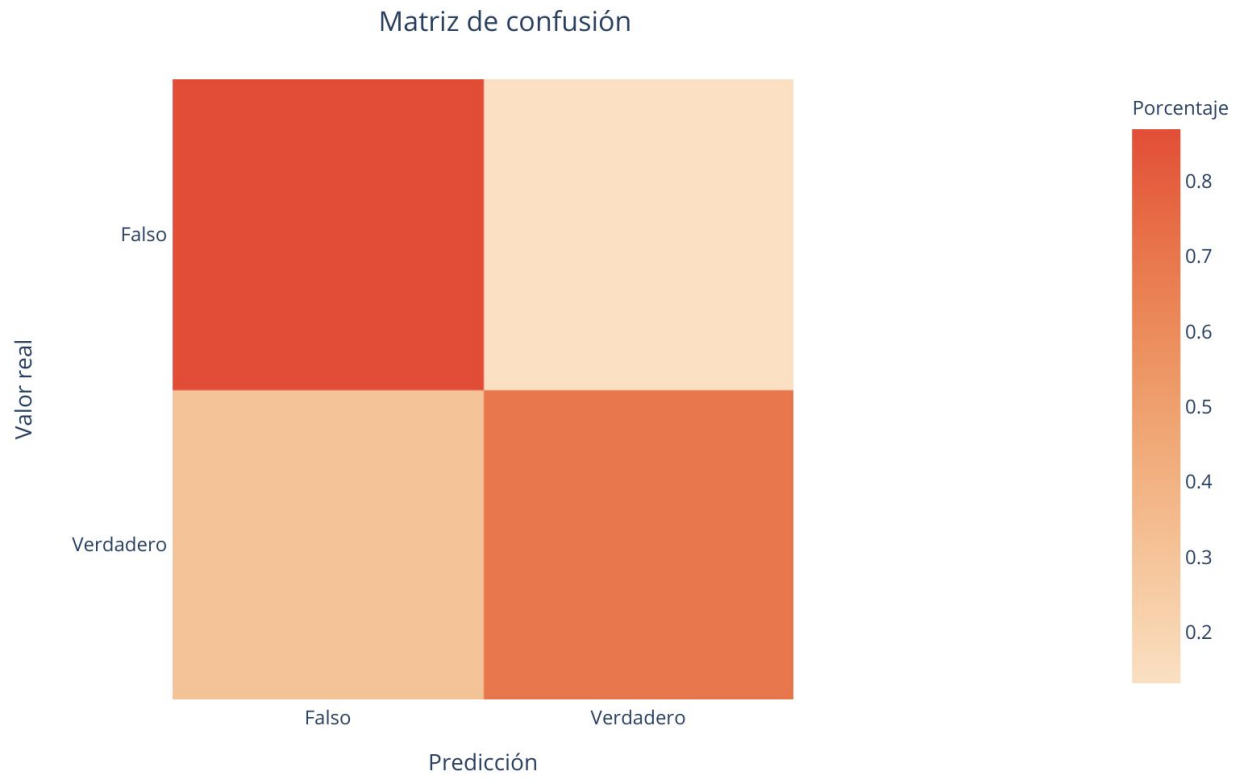
El comportamiento de este algoritmos es similar a los anteriores, no logramos grandes diferencias en cuanto a resultados. Fue dificil encontrar 'buenos' hiper parámetros usando GridSearch ó RandomGridSearch, ya que es un algoritmo son bastante sensible a la profundidad del árbol y otros parámetros.

Aumentando la profundidad del árbol mejoró la performance pero con el riesgo de que nuestro modelo sobreajuste.

Mostramos las métricas obtenidas con el set de entrenamiento:

Predicción	Precisión	Recall	F1
Falso	0.78	0.87	0.82
Verdadero	0.81	0.69	0.75

ROC: 0.7809



El resultados obtenido en promedio con el set de test fue: **0.7897**

4.3 Algoritmos basados en Redes Neuronales

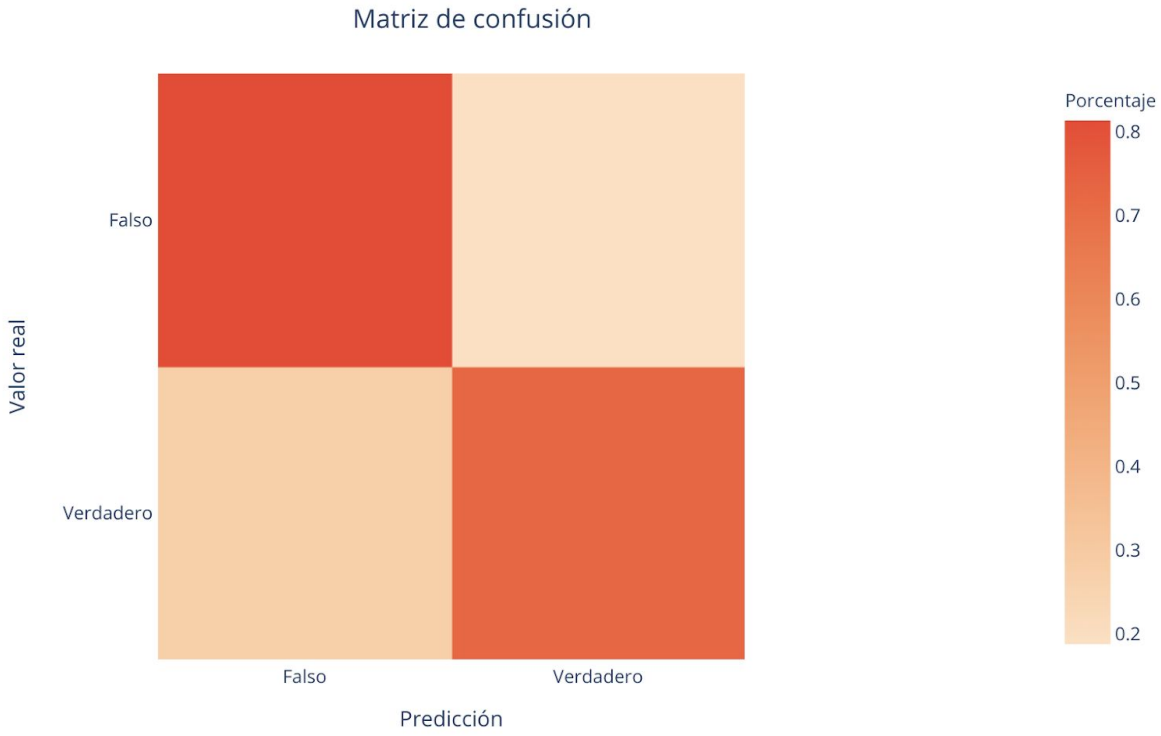
MLP Classifier

Dentro de los algoritmos basados en redes neuronales, únicamente con este algoritmo usamos TF-IDF. El inconveniente de este algoritmo es que tiene un costo computacional bastante alto con resultados menores que con los algoritmos estudiados posteriormente.

Para la búsqueda de mejores hiper parámetros usamos GridSearchCV. A continuación algunas métricas obtenidas:

Predicción	Precisión	Recall	F1
Falso	0.79	0.81	0.80
Verdadero	0.75	0.72	0.77

ROC: 0.7680



El modelo en general no tiene tantos falsos negativos o positivos pero la performance en cuanto a resultados es menor que otros algoritmos basados en redes neuronales.

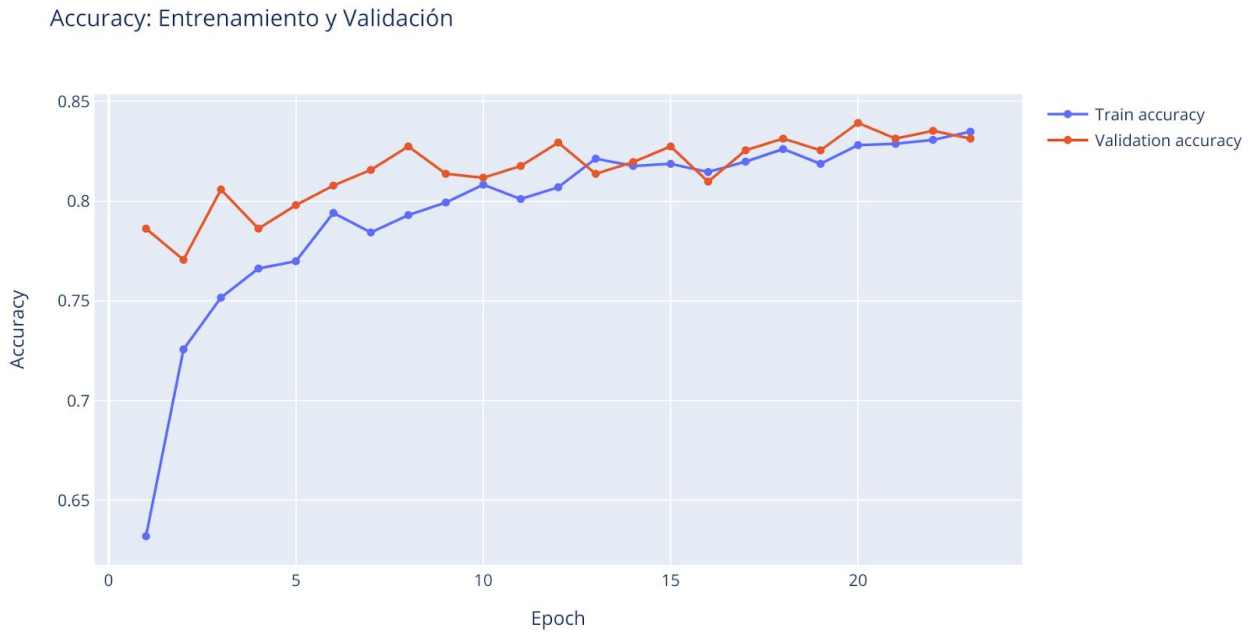
El resultado obtenido con el set de test fué: **0.7686**

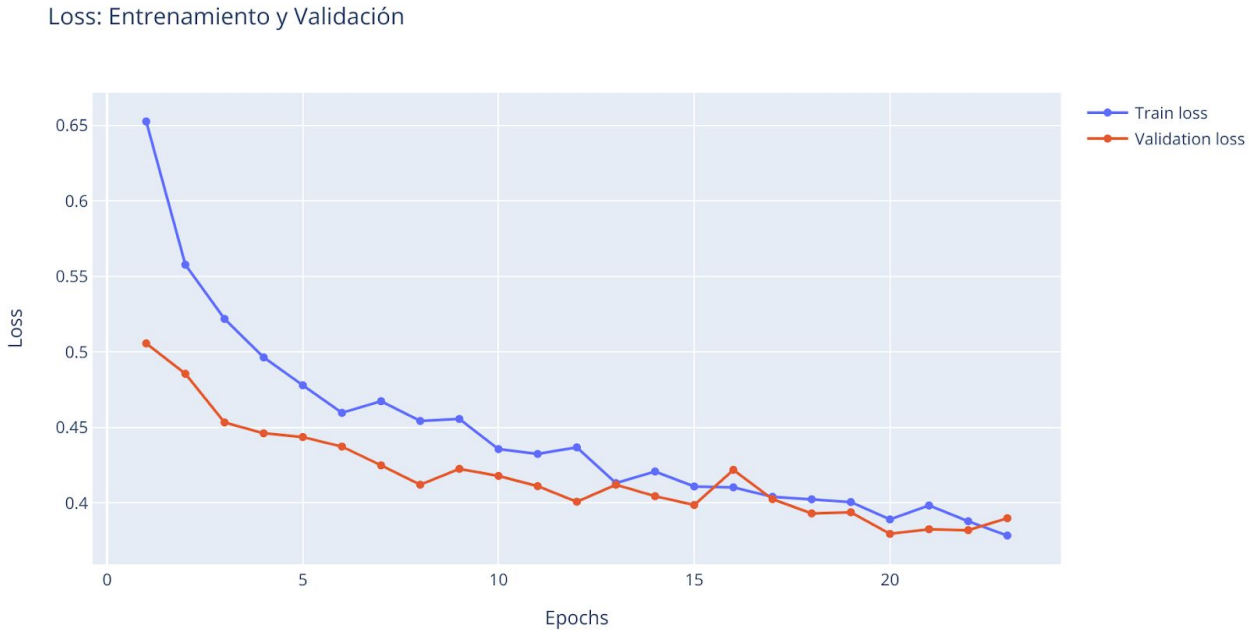
Convolutional 1D

Para el entrenamiento con el modelo se tuvieron en cuenta varios parámetros:

- Cantidad de filtros. Sus tamaños.
- Funciones de activación.
- Cantidad de neuronas.
- Tamaño de lote.
- Epochs.
- Tasa de aprendizaje.

A su vez se monitorearon los valores valores de exactitud y pérdida para evitar el sobreajuste. También configuramos los callbacks para que el modelo cambie la tasa de aprendizaje o se detenga en caso de que el valor de pérdida vuelva a aumentar.

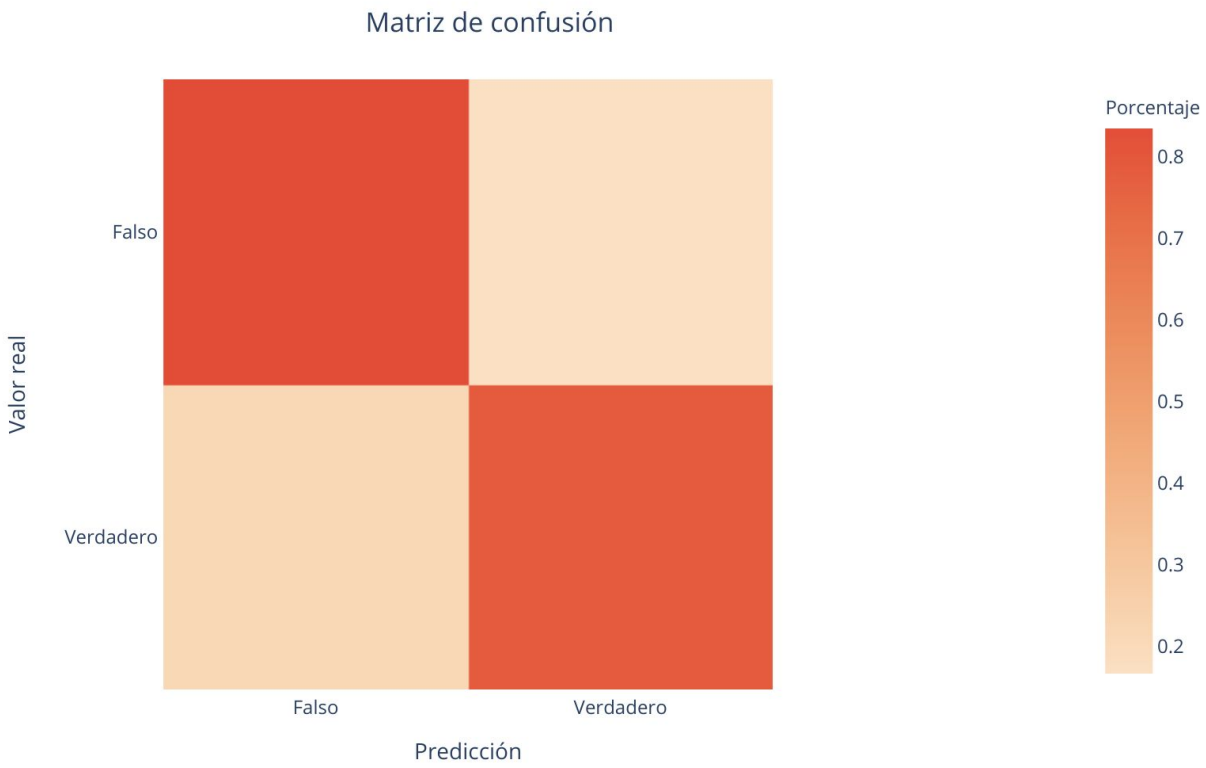




Cuando el valor 'validation loss' vuelve a aumentar(epoch=23) detenemos la ejecución. Con estos valores obtuvimos **0.8197** en promedio con el set de entrenamiento. Mostramos algunas métricas:

Predicción	Precisión	Recall	F1
Falso	0.84	0.83	0.84
Verdadero	0.78	0.78	0.78

ROC: 0.8090



Con el set de test obtuvimos **0.81489** lo cual es un valor cercano al obtenido al valor de entrenamiento con lo que podemos afirmar que el modelo no está sobre ajustado.

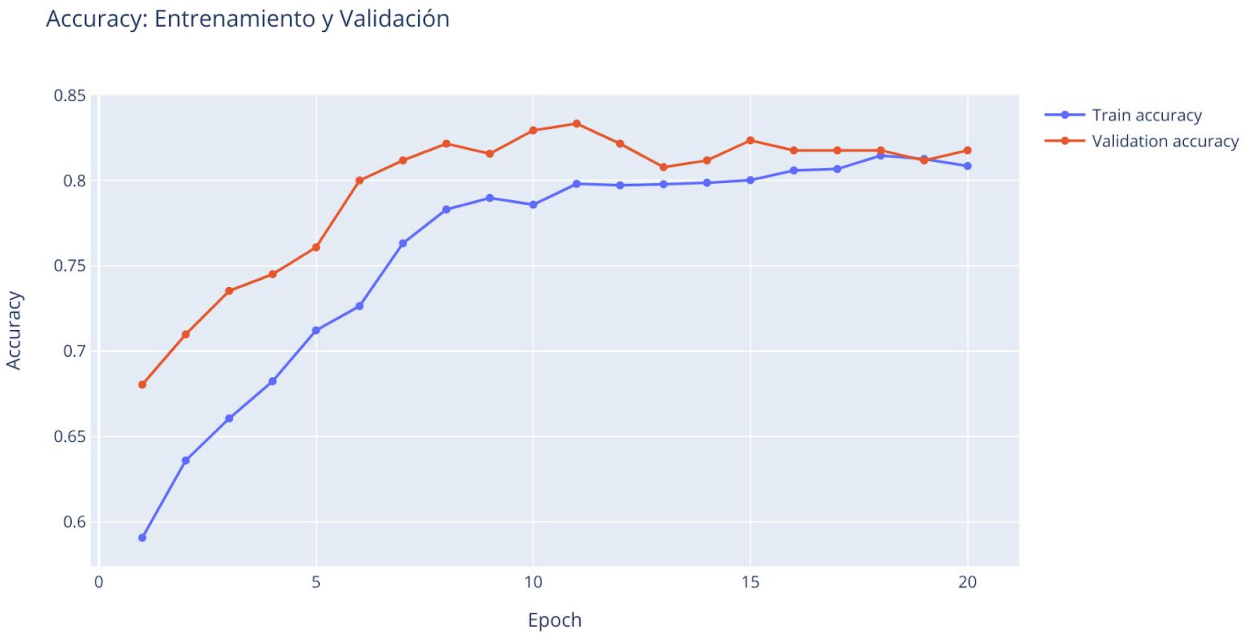
Bidirectional GRU(Gated Recurrent Unit)

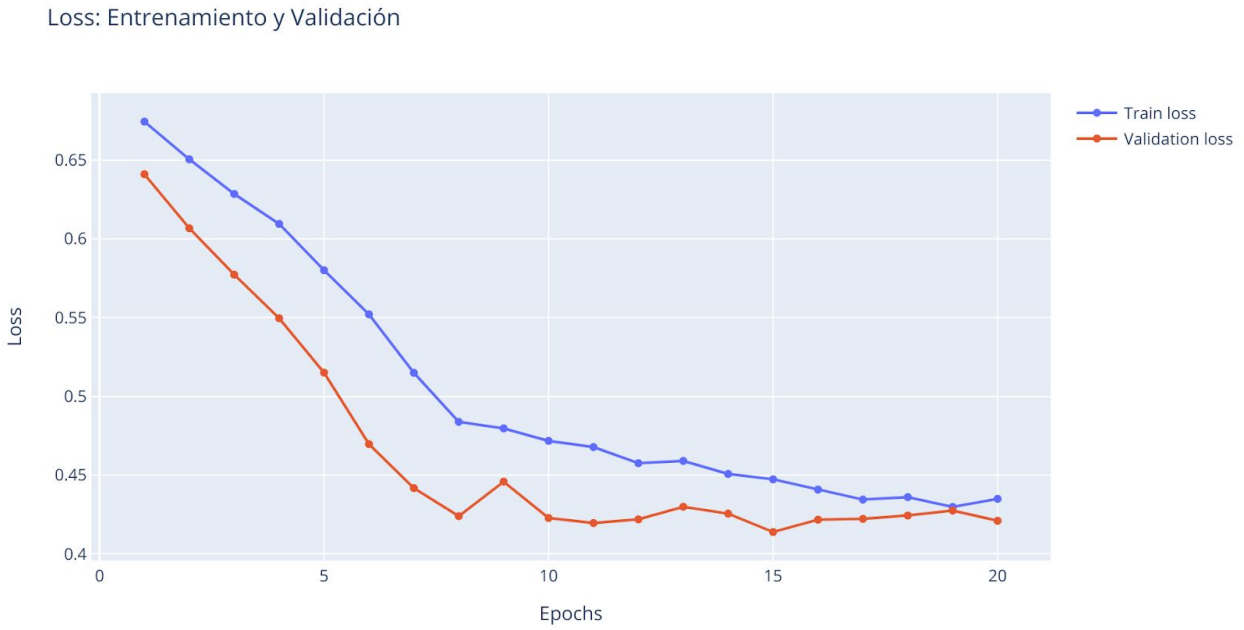
A diferencia de la red convolucional, en este algoritmos tuvimos que bajar la tasa de aprendizaje(desde 1e-3 hasta 1e-4) debido a que converge rápidamente(con 4 epochs) y el modelo empieza a sobre-ajustar.

Cambiamos la función de activación 'tanh'(default) por 'relu' con la que se obtuvieron mejores prestaciones. Algunas métricas:

Predicción	Precisión	Recall	F1
Falso	0.82	0.88	0.85
Verdadero	0.82	0.74	0.78

ROC: 0.8097





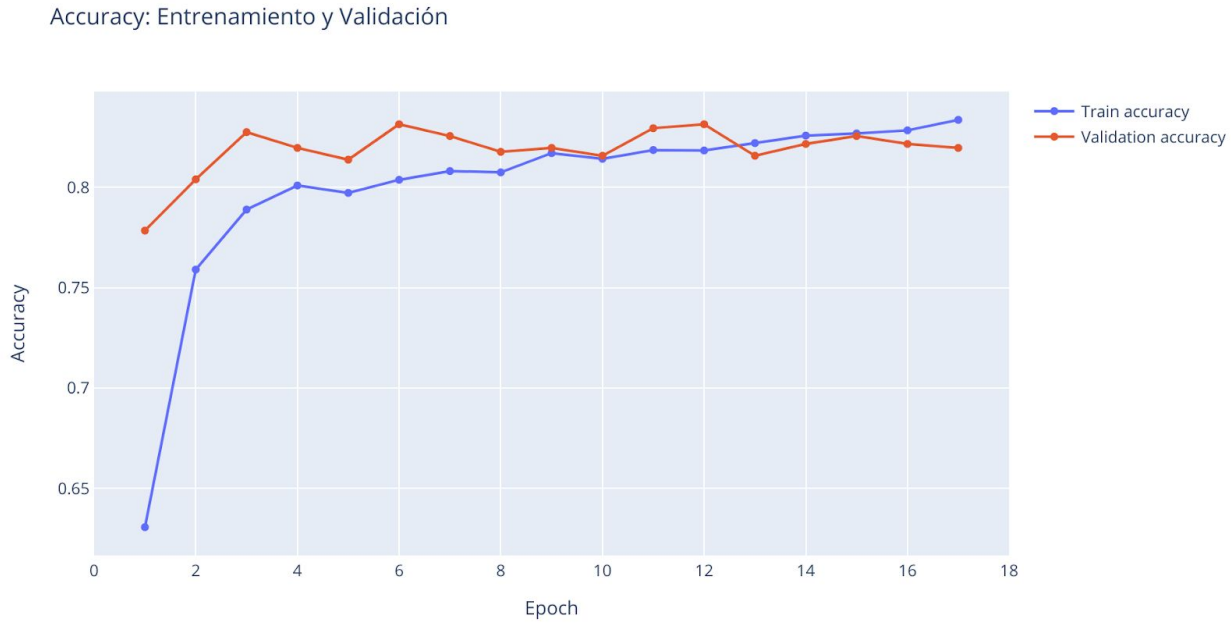
En este modelo particularmente requirió un mayor trabajo buscando parámetros ya que pensamos que es bastante sensible a los hiper parámetros.

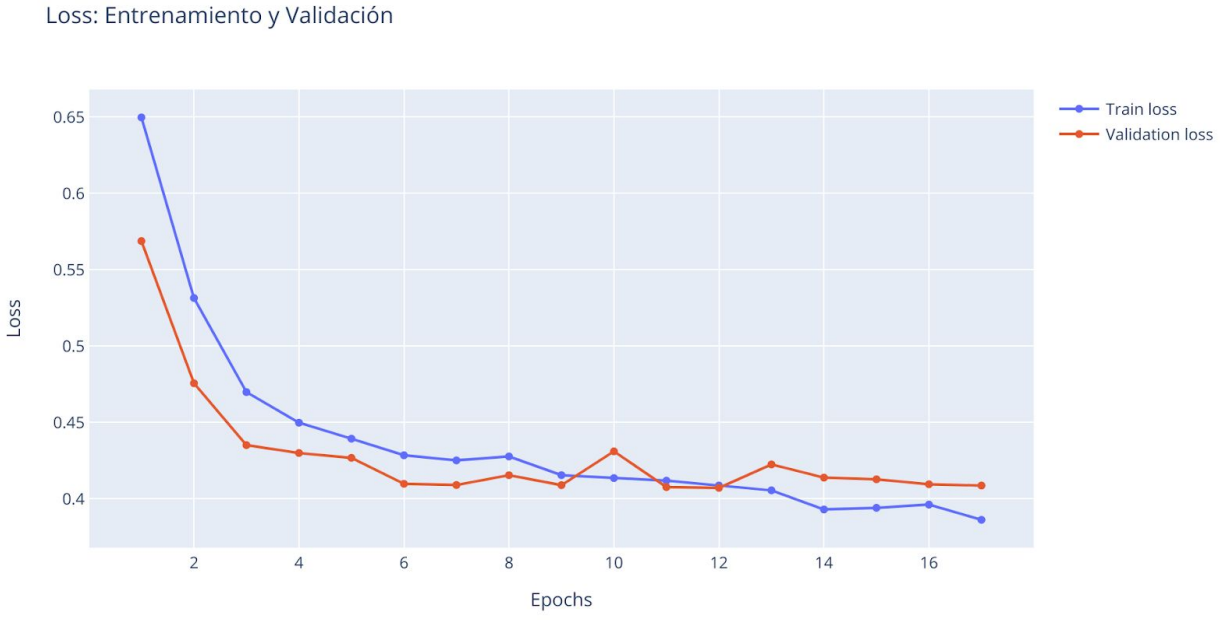
El valor promedio con el set de entrenamiento fue: **0.8213**

El score obtenido con el set de prueba fue: **0.8106**

Bidirectional LSTM(Long Short Term Memory)

Este algoritmos trajo mejores resultados con una tasa de aprendizaje del orden de (1e-5) con un tamaño de lote chico(32). En general converge rápidamente con pocos epochs, por lo que hay que configurar los callbacks para bajar la tasa de aprendizaje y en caso de que la pérdida vuelva a aumentar detenemos la ejecución.





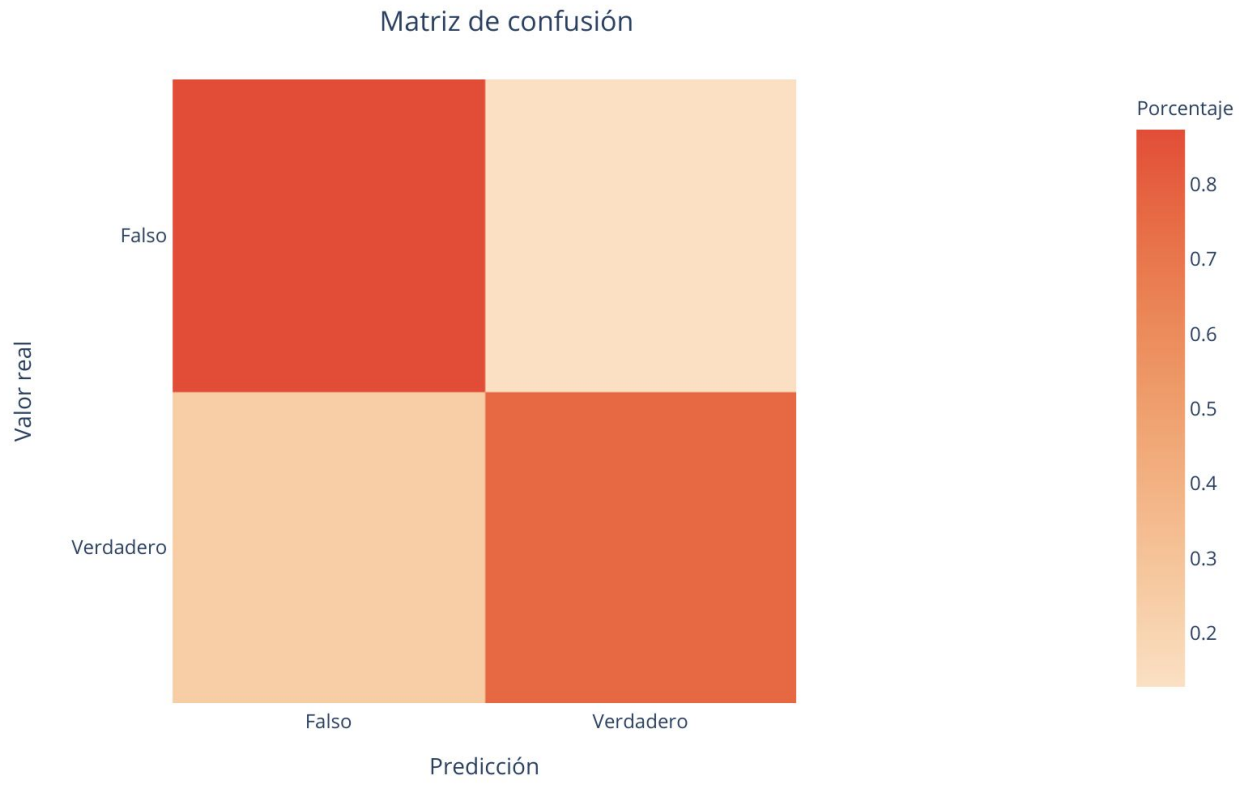
En el epoch 7 aproximadamente deja de caer la pérdida y vuelve a subir por lo que detenemos el algoritmos y entrenamos con esos valores el set de entrenamiento completo.

A continuación algunas métricas:

Predicción	Precisión	Recall	F1
Falso	0.83	0.87	0.85
Verdadero	0.82	0.76	0.79

ROC: 0.8150

La matriz de confusión:



El valor obtenido con el set de test fue: **0.8158**

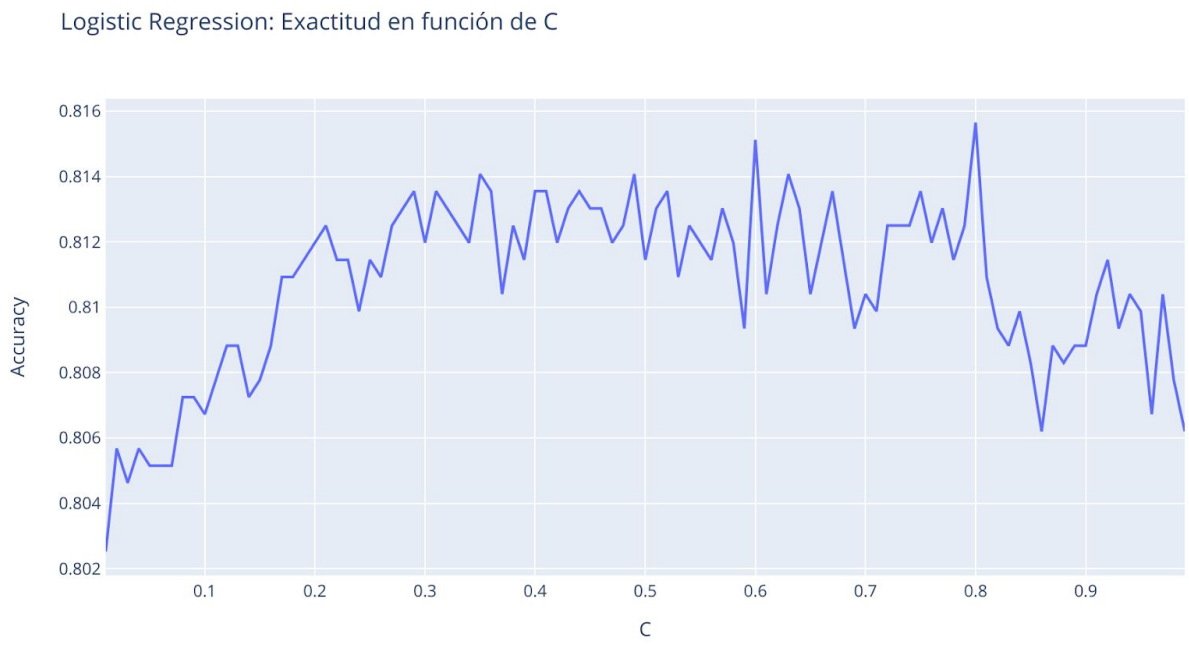
DistilBERT

En este caso vamos a usar una versión ligera del modelo BERT que es el próximo a analizar. La salida es un vector de 768 dimensiones por cada valor del campo 'text', un vector de incrustaciones de un modelo previamente entrenado(BERT):

'our deeds are the reason' → DistilBERT → Logistic Regression → Predicción(1/0)

Una dificultad que tuvimos es que con más de 2000 mil registros teníamos dificultades por el uso de memoria RAM. Para evitar ejecutar en un plataforma como Kaggle particionamos

nuestro set en 3 bloques y luego apilamos las salidas. Buscamos el mejor valor para el hiper parámetro C.



Los scores obtenidos usando 5-fold cross validation:

[0.81260946, 0.82486865, 0.81698774, 0.81873905, 0.80718668]

Luego del entrenamiento con los mejores hiperparametros, con el set de test obtuvimos un score: **0.8121**

BERT(Bidirectional Encoder Representations from Transformers)

La característica importante de esta tecnología es que se tiene en cuenta la palabra en el contexto de una oración, es decir, importa la palabra que está adelante o detrás de esta. Esto permite comprender la semántica de la oración teniendo en cuenta toda la oración y no las palabras únicamente.

Para ayudar al modelo a distinguir entre dos oraciones se agregan separadores entre dos oraciones. [CLS] al inicio de la oración y [SEP] para separar oraciones:

[CLS] Sentence A [SEP] Sentence B [SEP]

En una oración, el modelo enmascara al azar el 15% de las palabras en la entrada con [MASK], luego tiene que predecir las palabras enmascaradas. Esto es diferente de las redes neuronales recurrentes tradicionales que generalmente ven las palabras en forma secuencial. Esto permite que el modelo aprenda una representación bidireccional de la oración.

Como ejemplo mostramos una oración extraída del set de entrenamiento en la cual enmascaramos la palabra 'dead' , la cual intentaremos predecir:

'police officer wounded suspect dead after exchanging shots'

Reemplazamos la palabra **'dead'** por **[MASK]** :

'police officer wounded suspect [MASK] after exchanging shots'

La API que simula el modelo devuelve las posibles palabras que dado el contexto de la oración podrían ocupar esa posición, cada uno con sus probabilidades respectivas.

https://huggingface.co/bert-large-uncased

Increíblemente el modelo predijo que la palabra es **'dead',** la misma que la oración original.

Para el entrenamiento probamos con 3 versiones de los vectores pre entrenados:

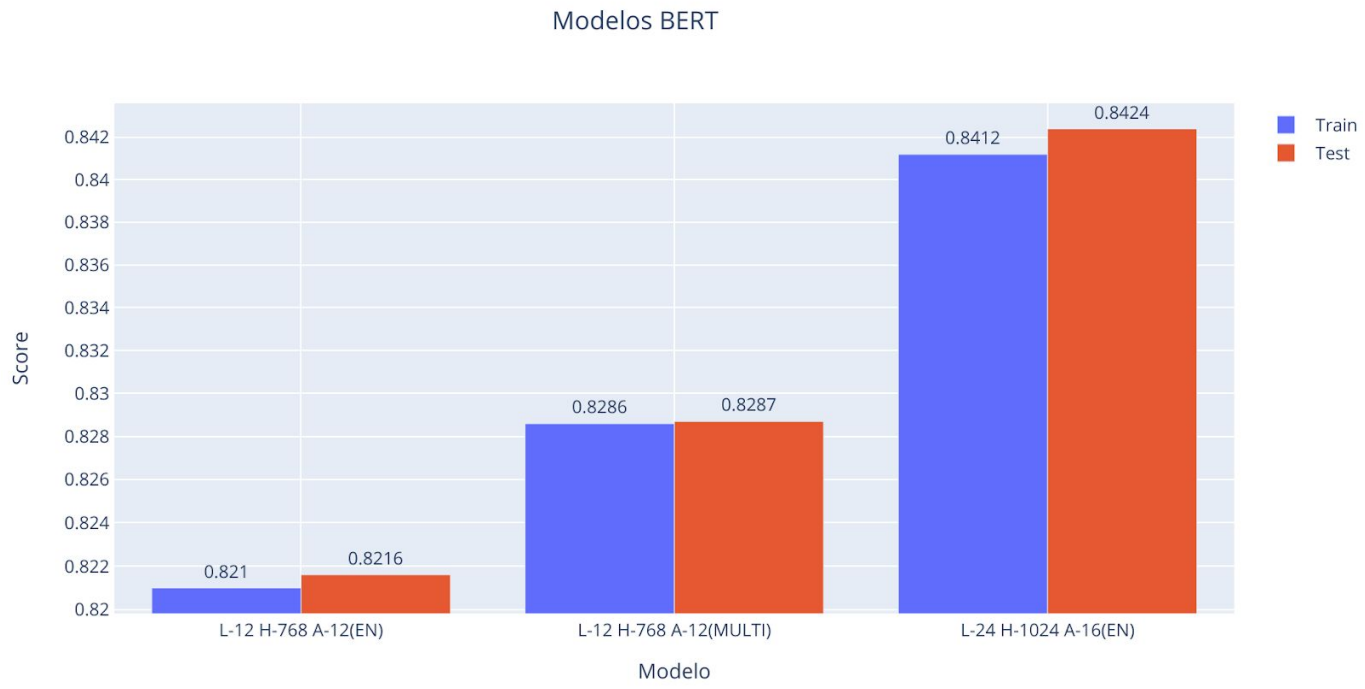
- Versión base con 12 head-attention y 768 unidades ocultas. En inglés.
- Versión large con 16 head-attention y 1024 unidades ocultas. En inglés.
- Versión multilingüe con 12 head-attention y 768 unidades ocultas.

La plataforma Kaggle nos provee 16GB de memoria RAM, más 16GB usando la GPU con lo que pudimos entrenar solamente con algunos hiper parámetros.

En la salida del modelo de entrenamiento probamos con varias capas e incluso con una red convolucional 1D logrando buenos resultados en comparación a los algoritmos ya estudiados.

Con una capa densa con función de activación **'sigmoid'** logramos un score de **0.8412** para el set de entrenamiento y **0.8424** para el set de test siendo hasta ahora nuestro mejor resultado.

Con 3 epochs y 16 de tamaño de bloque el modelo generaliza bien y computacionalmente no es muy costoso. Aumentando la cantidad de epochs el modelo tiene a sobre-ajustar. No podemos aumentar el batch_size debido a que tenemos una limitación en cuanto al uso de memoria. El resultado de entrenar con los tres modelos:



5. Ensambles

5.1 Blending

Para la implementación de este ensamble seguimos las indicaciones del apunte de la cátedra.Combinamos 3 modelos usando GloVe como embeddings:

Primer modelo:

- Dos capas bidireccionales LSTM
- Una capa en la salida con función de activación sigmoide.

Segundo modelo:

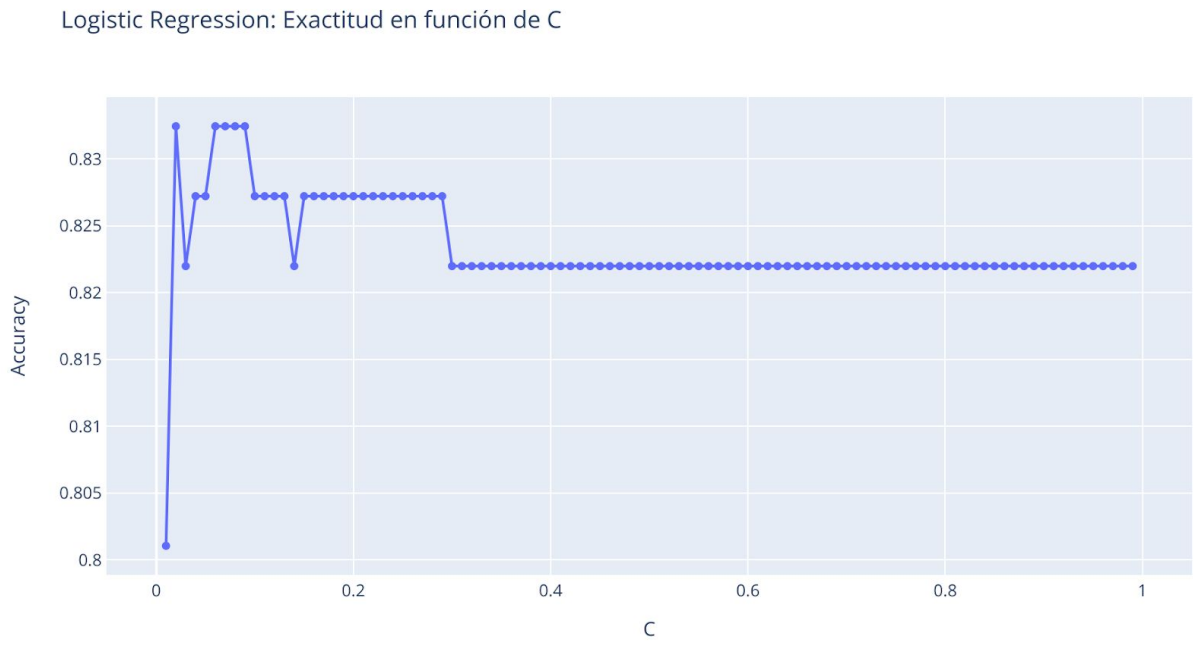
- Una capa bidireccional GRU

- Una capa densa de activación relu.
- Una capa de salida con función de activación sigmoid.

Tercer modelo:

- Una red convolucional 1D
- Una GlobalMaxPooling 1D
- Una capa densa con función de activación relu.
- Una capa en la salida con función de activación sigmoide.

Para el entrenamiento de las predicciones de estos 3 algoritmos usamos Logistic Regression.



Usamos el valor C=0.08 para entrenar el algoritmo Logistic Regression con el que obtenemos un score de **0.8324**.

Luego de esto entrenamos los 3 modelos con el set de entrenamiento completo como lo hacemos habitualmente y realizamos las predicciones con el set de test.

Hacemos las predicciones con Logistic Regression(previamente entrenado) con las salidas de estos 3 algoritmos:

Obtenemos 0.81795 con el set de test, un valor ligeramente más alto que los algoritmos evaluados individualmente.

5.2 Majority Voting

Este ensamble lo usamos combinando el resultados del set de test de 5 algoritmos. En algunos casos repetimos las salidas del mismo algoritmos con otros hiper parámetros.

Las mejores combinaciones las obtuvimos con:

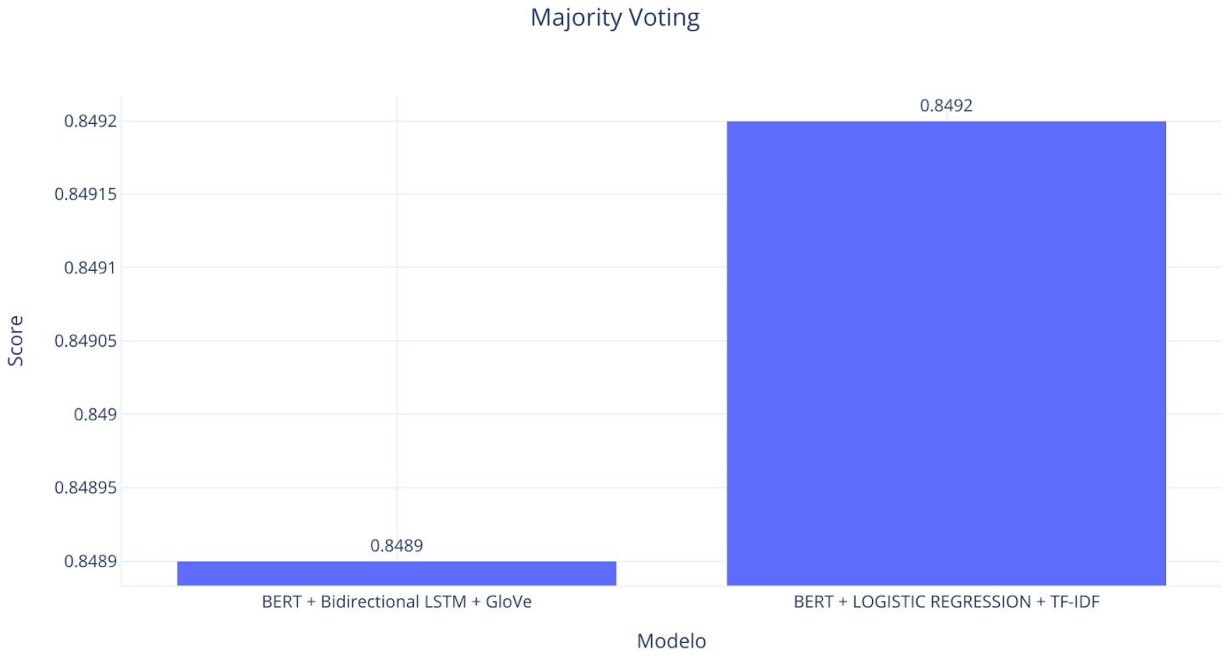
- BERT con optimizador SGD
- BERT con optimizador Adamax
- BERT con optimizador Adam
- BERT con optimizador Adam
- Bidirectional LSTM con embeddings GloVe

Con estas combinaciones obtuvimos **0.8489.**

La segunda combinación, mejor que la anterior fue con:

- BERT con optimizador SGD
- BERT con optimizador Adam
- BERT con optimizador Adam
- BERT con optimizador Adam
- Logistic Regression con TF-IDF

Obteniendo el resultado **0.8492,** el mejor hasta el momento.



6. Conclusiones

Los resultados de las predicciones son combinaciones de la naturaleza del set de datos, preprocesamiento y algoritmos con sus respectivos parámetros.

Una de las tareas que más tiempo nos consumió fue el preprocesamiento de los tweets ya que contenían mucho ruido para los algoritmos utilizados.

Dado que los tweets tienen pocas palabras, los algoritmos tradicionales tienen un rendimiento menor debido a que requieren mayores características. Los algoritmos basados en árboles si bien tienen un rendimiento superior, son sensibles a datos desbalanceados, si bien nuestros datos estaban bastante balanceados, los algoritmos tenían una tendencia a predecir tweets verdaderos como falsos dado que los verdaderos eran la clase minoritaria. Al hacer el uso de weighting hubo una leve mejora.

Los algoritmos basados en redes convolucionales(CNN) aprenden mejor las características locales de las oraciones mientras que las RNN como LSTM pueden aprender de las secuencias temporales debido a tienen la capacidad de usar estados previos para decidir sobre el siguiente.

Los algoritmos basados en transformers como BERT/ELMO devuelven resultados superiores ya que contextualizan las palabras dentro de la frase u oración. BERT usa head-attention y ELMO una RNN(LTSM) en sus capas. Estos modelos a su vez han sido pre entrenados con dos fuentes: Wikipedia y BookCorpus(miles de libros).

Finalmente, para la competencia organizada por la cátedra usamos ensambles entre varios algoritmos logrando buenos resultados.

7. Referencias

- **Pandas Cookbook** - Theodore Petrou
- **English Letter Frequency Counts** - http://norvig.com/mayzner.html
- **TensorFlow** - https://www.tensorflow.org/hub/tutorials/tf2_text_classification
- **Diccionario** - https://dictionary.cambridge.org/es/
- **Urban Diccionary** - https://www.urbandictionary.com/
- **Google Research** - https://github.com/google-research/bert
- **Contextualized Word Representations** - http://ai.stanford.edu/blog/contextual/
- **Yoon Kim - Convolutional Neural Networks for Sentence Classification** - https://www.aclweb.org/anthology/D14-1181.pdf
- **Embedding Projector** - http://projector.tensorflow.org/
- **Transformers** - https://huggingface.co/