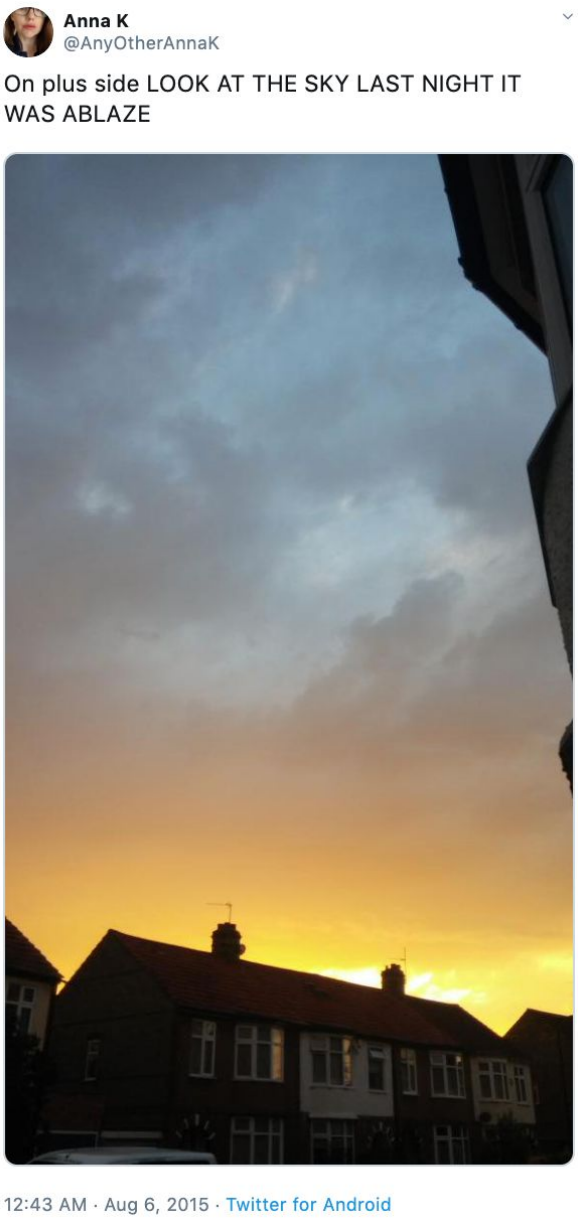
¿Real o No? NLP con Tweets de Desastres

Predecir cuáles Tweets son sobre desastres reales y cuáles no

Daniel Fernando Acosta González
Maria Fernanda Florez Jaimes
Johan David Marin Benjumea

Descripción del problema



Twitter se ha convertido en un importante canal de comunicación en situaciones de emergencia.

La omnipresencia de los smartphones permite a las personas anunciar una emergencia que están observando en tiempo real. Por este motivo, más actores como organizaciones de atención a desastres y agencias de noticias, están interesadas en monitorear periódicamente esta red social.

Sin embargo, no siempre es claro si las palabras de una persona realmente anuncian un desastre. Como en el ejemplo de la imagen.

El autor puede usar de manera explícita la expresión “ABLAZE” que puede traducirse como “en llamas” pero referirse a ella metafóricamente. Esto es claro para una persona, especialmente con ayudas visuales pero, es menos claro para una máquina.

El objetivo de este proyecto es construir un modelo de Machine Learning que prediga cuáles tweets son sobre desastres reales y cuáles no lo son.

El proyecto parte de una competición de Kaggle que puede encontrarse en este link.

Se espera predecir si un tweet dado es sobre un desastre real o no. Si lo es, se predice un 1 y si no lo es, se predice un cero.

Descripción del dataset

Se tiene acceso a un conjunto de datos (dataset) de 10000 tweets que fueron clasificados a mano. Para realizar el ajuste de un modelo se dividen los datos en dos archivos. uno para entrenamiento con 7613 registros y los 2387 restantes para test. Organizados en un fichero aparte cada uno. Cada uno de los registros posee varias columnas explicadas más adelante.

Ficheros

- **train.csv** - El conjunto de entrenamiento
- **test.csv** - El conjunto de test

Columnas

- id - un identificador único del tweet
- text - el texto del tweet
- location - la ubicación desde la que fue enviado el tweet (podría estar en blanco)
- keyword - una particular palabra clave dentro del tweet (podría estar en blanco)
- target - solo en **train.csv**, denota si el tweet es sobre un desastre real (1) o no (0)

Métricas de desempeño

Las presentaciones se evalúan utilizando F1 entre las respuestas previstas y esperadas.
F1 se calcula a través de la siguiente expresión:

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

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

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

True Positive $[TP] =$ La predicción es 1, y en realidad también es 1 - se predijo un positivo y eso es verdad.

False Positive $[FP] =$ La predicción es 1, y en la realidad es 0 - se predijo un *positivo*, y eso es *falso*.
False Negative $[FN] =$ La predicción es 0, y en realidad es 1 - se predijo un negativo, y eso es falso.

Desempeño deseable en producción

Se puede esperar que el modelo sirva como una herramienta que provea alertas tempranas a organismos de socorro y agencias de noticias.

Por otro lado, en cuanto al desempeño del modelo, el 54.5% de los modelos previamente ajustados por otros participantes han superado 0.8 en F1. Se considera entonces que este modelo puede tener un desempeño adecuado y posiblemente brinda buenas oportunidades a organismos de socorro y agencias de noticias si se supera el 0.8 en F1. Una herramienta de alertas tempranas para una agencia de noticias podría significarle un considerable aumento en primicias y para un organismo de socorro podría significar reducir sus tiempos de atención. Alcanzado el desempeño propuesto en el modelo, se puede aumentar considerablemente el desempeño operacional de estas organizaciones.

Exploración y limpieza de datos

Inicialmente se realizó una exploración de los datos en el archivo de train con la instrucción *describe,* que realiza un análisis estadístico básico del conjunto de datos, observando primero si existían tuplas (text, target) duplicadas, encontrando un total de 620 registros duplicados, los cuales se removieron dejando solamente un registro de cada tipo. Luego se realizó la misma búsqueda pero esta vez basados solo en la variable text, en este caso se encontró un total de 146 tweets duplicados y clasificados en ambas categorías, como se muestra en la figura 2. Al hacerse imposible determinar si son desastres o no, se eliminan, quedando con un total de 6847 de los 7613 totales, 2801 de los cuales son tweets de desastres reales y 4046 que no lo son.

	text	target
	U.S National Park Services Tonto National Fore...	0
	World Annihilation vs Self Transformation http...	0
	U.S National Park Services Tonto National Fore...	1
	World Annihilation vs Self Transformation http...	1

Limpieza 1

Para realizar todo el proceso de limpieza de los datos y evaluación de modelos se desarrolló un módulo: `NPLtweet`. Este contiene dos funciones de limpieza diferentes, 5 funciones para crear corpus, listas de palabras, remover palabras, concatenar strings y comparar conjuntos de palabras, una función para dibujar word clouds, una función para guardar la información de los

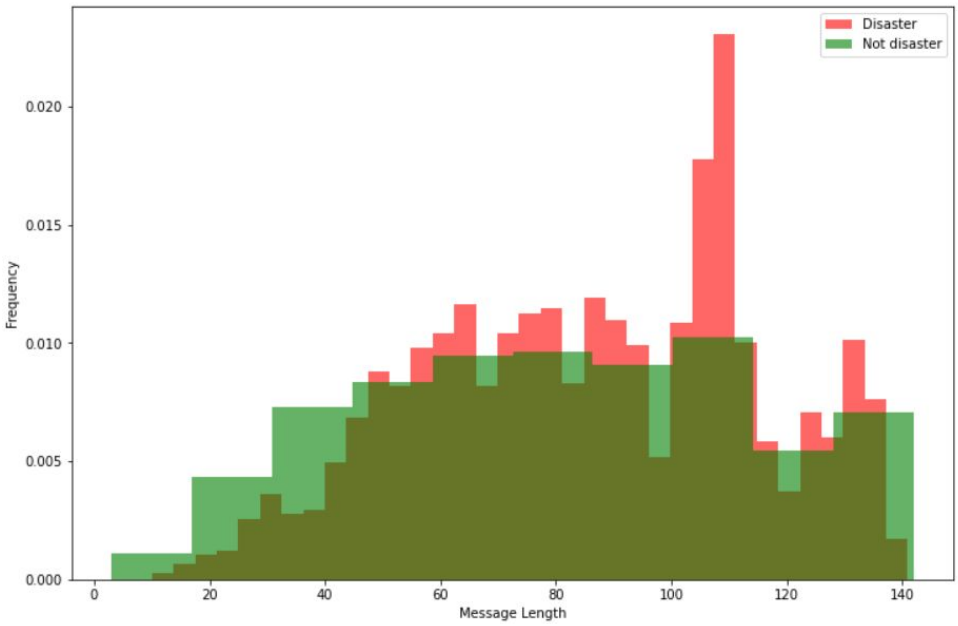
modelos y una para graficar el desempeño de los mismos con distintas condiciones y valores de los paŕametros.

La primera limpieza se realiza utilizando la función `NPLtweet.clean()` que se apoya en el uso de *Regular expression operations* (paquete `re`) y, principalmente, las funciones `re.sub(), re.compile(), y re.findall()`, para reemplazar cadenas de caracteres, crear expresiones regulares y buscar coincidencias en el texto respectivamente. La función de limpieza `NPLtweet.clean()`, recibe un tweet y elimina URLs, Hashtags, Tags y emoticones. Además, reemplaza contracciones por palabras completas, elimina caracteres especiales generados por la sintaxis html y markdown y elimina la puntuación, retornando un texto limpio. Se aplica esta función al conjunto de train, utilizando las funciones propias de python, `apply() y lambda,` agregando la variable *sms* al data frame como se muestra en la siguiente figura.

	id	keyword	location	text	target	sms
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	our deeds are the reason of this earthquake ma...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada

Se realiza una exploración de los datos con el fin de comparar la longitud de los mensajes que hacen referencia a un desastre y aquellos que no. Se asigna el color rojo a los tuits que reflejan desastre y color verde para el caso contrario. Las longitudes de los mensajes no parecen reflejar alguna tendencia ligada al target, por lo que dicha “variable” no sería relevante en el análisis.

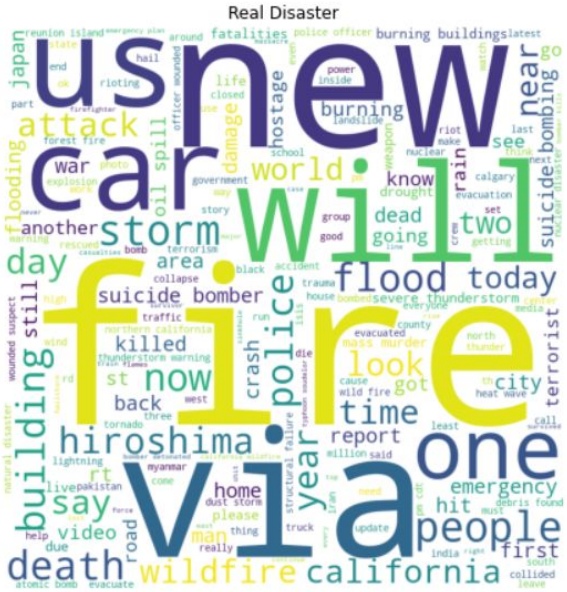
Mediante la función `split` y un filtro, se crea un listado de palabras, con todos los tweets para cada una de las etiquetas, es decir, los que se refieren a un desastre (1) y los que no (0). Luego, utilizando la función `Counter()` del paquete `collections` se crea la función `NPLtweet.compare_most_commom(),` cuyo objetivo es comparar las palabras más frecuentes en ambas etiquetas, implicando que estas, son principalmente palabras que se cree no son relevantes para clasificar el tweet como desastre o no. Estos casos son evidentes en los artículos, conjunciones, preposiciones, pronombres, verbos modales, adverbios, entre otras.



Sin embargo, no se remueven debido a que esto se hace en la segunda limpieza y de esta forma se hace posible comparar el desempeño en ambos tipos de tratamiento.

```
['the', 'in', 'a', 'of', 'to', 'is', 'and', 'on', 'for', 'i', 'at', 'it', 'from', 'that', 'not', '
```

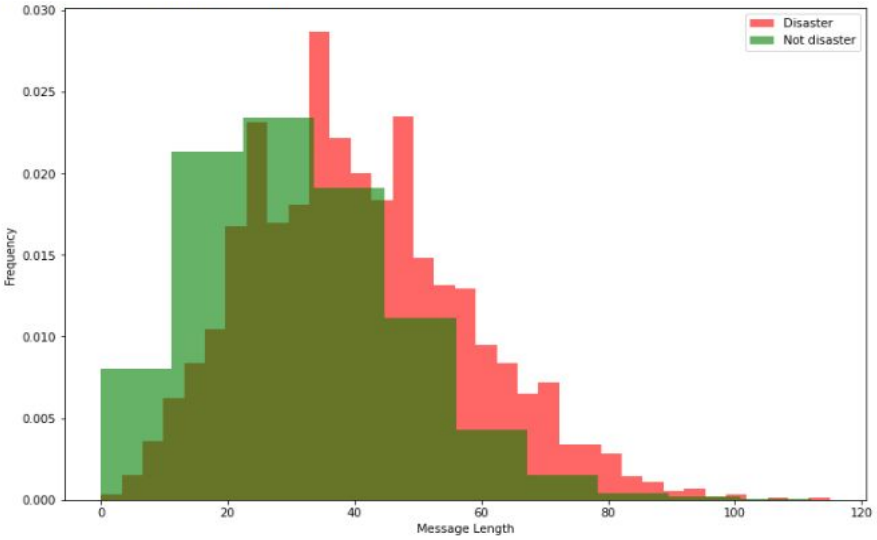
Se generan nubes de palabras para cada grupo de tweets, observando así las 200 palabras más frecuentes en cada categoría.



Limpieza 2

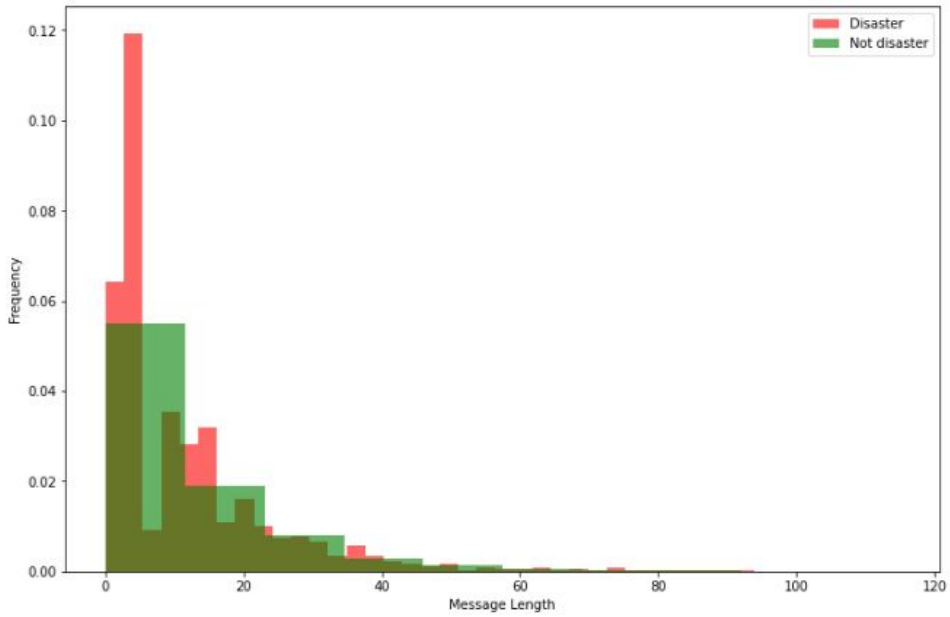
La segunda limpieza se realiza utilizando la función `NPLtweet.clean2`. De manera similar a la función anterior, elimina los componentes web (etiquetas, urls, emoticones y caracteres especiales de html y markdown) utilizando el paquete `re`. Luego, utilizando listas de construcción propia y la función join, se eliminan las stop words (pronombres, verbos modales, verbos comunes, auxiliares, adverbios, conjunciones, wh-questions, entre otras), meses, días de la semana, la puntuación, los números y números ordinales; estos dos últimos tanto en palabras como en digitos. También se eliminan palabras que se cree no aportan valor, y que simplemente pueden complicar al modelo al momento de hacer predicciones. Además, como se tiene el objetivo de que el modelo pueda predecir si un tweet en inglés que está ligado a un desastre o no, sin importar el lugar de origen del tweet, se eliminaron las expresiones relacionadas con la localización del mensaje, añadiendo que estas características, no presentan relevancia aparente en la clasificación y, podrían sesgar la identificación de desastres con lugares específicos. Se obtiene de este modo un total de 13818. Teniendo en cuenta la recomendación del profesor de que reducir el número de palabras puede disminuir el overfitting, se eliminaron las palabras con una frecuencia de aparición inferior a 5. Finalmente

se realiza la misma exploración con el histograma para la longitud del tweet y los word clouds para las 200 palabras más frecuentes en cada categoría. El gráfico de frecuencia de la longitud de los tweets de desastre no presenta un comportamiento significativamente superior al de no desastre, por lo que dicha longitud no aporta valor al modelo predictivo.



En la nueva nube de palabras, se observa variación con respecto a la realizada anteriormente, puesto que, por ejemplo, no se presenta la palabra 'will', cuya alta frecuencia de aparición en ambas clases de tweet (disaster, not disaster), la convierte en una palabra poco aportante al análisis y posible fuente de sesgo en las predicciones.

Limpieza 3



Como último tratamiento, se extraen todos los componentes web de cada tweet: hashtags, etiquetas, emoticones y un http si incluye una url.

Finalmente, se tiene en cuenta que para la mayoría de los tweets las variables *keyword* y *location* presentaban datos nulos y, por lo tanto, se evitó su uso en modelos.

Pre-procesamiento y análisis de datos

El texto se preproceso de 3 formas diferentes, para cada uno de los tratamientos de limpieza. En primera instancia, se usó un bag of words (bow), utilizando la función `CountVectorizer` del módulo `feature_extraction.text` de la librería `sklearn` obteniendo un corpus con 16757

palabras para el tratamiento 1 y 4282 para los tratamientos 2 y 3. En segunda instancia, se realiza un análisis de componentes principales sobre los bow, reduciendo lo componentes a 100 para de cada una de las limpiezas, finalmente se utilizó un text embedding con ayuda de la librería `tensorflow_hub` y el corpus "universal-sentence-encoder/4", obteniendo un vector de 535 dimensiones para cada tratamiento.

Modelos

Se consideraron dos tipos de modelos predictivos de un solo parámetro, los cuales son: random forest classifier (`sklearn.ensemble.RandomForestClassifier`) y support vector classifier (`sklearn.svm.SVC`). Para cada modelo, se usaron tres valores diferentes de los parámetros: una profundidad máxima = {3, 6, 12} para el bosque de clasificación y un valor gamma = {0.3, 0.5 y 0.7} para el SVC. Cada uno de los modelos se prueba con diferentes porcentajes de los datos para el train p = {0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9}. Finalmente cada uno de los experimentos se corre un total de 5 veces como se observa en las siguiente tabla; con un total de 2160 corridas. Para medir el desempeño, se implementa la función `metric` para calcular la métrica explicada previamente en este documento, además de utilizar el BIAS entre el desempeño en las predicciones de entrenamiento y test como método para medir el overfitting.

<table>
<tr><th colspan="2">Factores</th><th>Niveles</th><th>Réplicas</th></tr>
<tr><td rowspan="2">Modelo</td><td>Random Forest Classifier (3)</td><td rowspan="2">6</td><td rowspan="5">5</td></tr>
<tr><td>SVC(3)</td></tr>
<tr><td colspan="2">Train proportion</td><td>8</td></tr>
<tr><td colspan="2">Limpieza datos</td><td>3</td></tr>
<tr><td colspan="2">Preprocesamiento datos</td><td>3</td></tr>
</table>

Debido a que el tiempo que tarda cada una de las corridas es considerable, se crea la función `NPLtweet.learning_data`, la cual permite almacenar el valor medio y la desviación estándar de las 5 réplicas de cada tratamiento, además de las características y el estimador utilizado.

Después de realizar el análisis de los algoritmos, se propone efectuar la prueba realizando una clusterización de los tweets utilizando un algoritmo k mean, con diferentes números de clusters: {2, ..., 5}. Se entrena un modelo, encontrando el mejor desempeño en el análisis anterior para cada uno de los clusters y se concatenan las predicciones Posteriormente, se calcula el desempeño y el overfitting.

Resultados

Para comparar los modelos, en las diferentes condiciones planteadas previamente, se implementa la función `NPLtweet.plot_size_efect` que permite observar el efecto del porcentaje de datos en train y test sobre el desempeño del modelo en la predicción de estos dos subconjuntos (para un valor de `max_dp` y gamma determinado) y, la función `NPLtweet.plot_parameter_efect`, que permite ver el desempeño del modelo para diferentes valores de max_dp y `gamma`, con un porcentaje para train y test determinado.
Primero se analizan los tres métodos de limpieza, concluyendo que el rendimiento medio de los modelos es mejor para la limpieza 2 que para las demás, sin embargo con ambos estimadores se llega al mejor desempeño con la limpieza 1.

Modelo	**Limpieza**	**Desempeño medio**	**BIAS promedio**	**Mejor desempeño**	**BIAS mejor desempeño**
Random forest Classifier	**1**	0.473	0.123	0.746	0.407
	2	0.498	0.114	0.734	0.304
	3	0.283	0.080	0.576	0.241
SVC	**1**	0.546	0.358	0.778	0.775
	2	0.629	0.216	0.756	0.829
	3	0.262	0.073	0.588	0.262

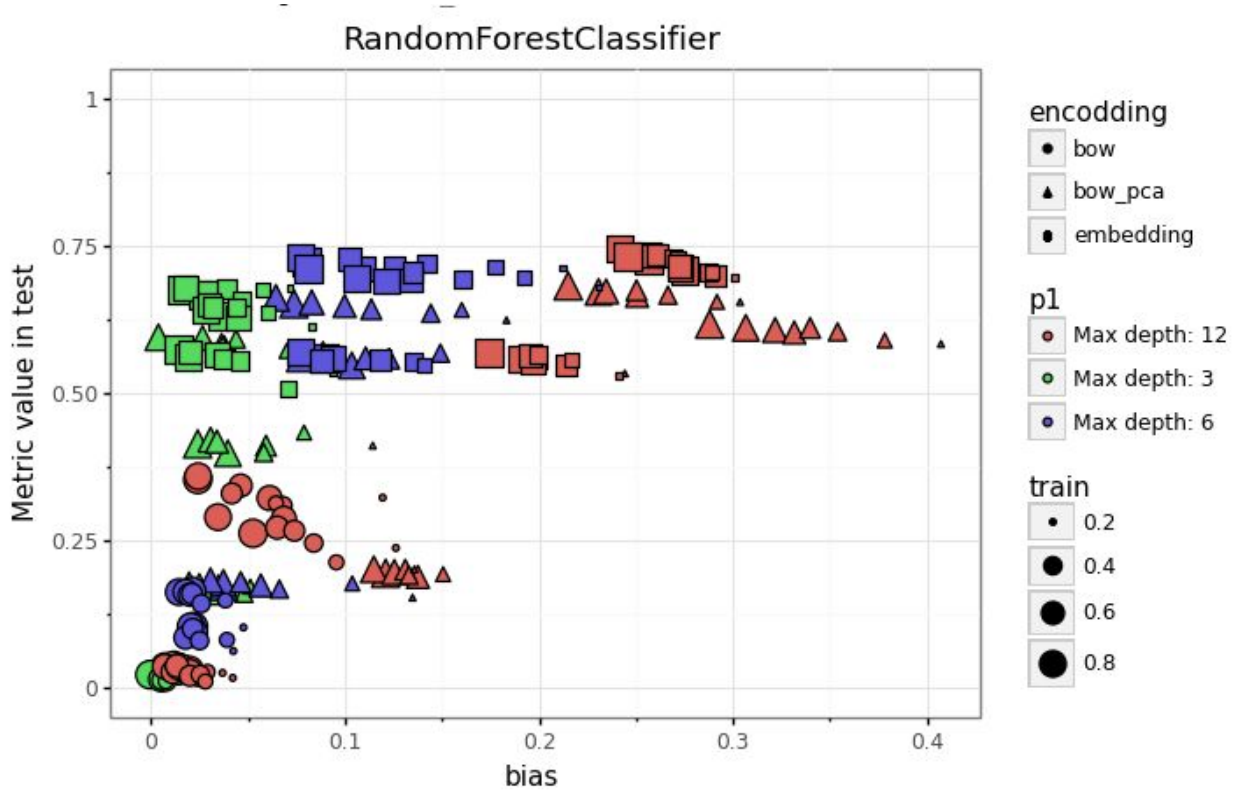
En cuanto a los métodos de procesamiento se encontró que a medida que se realiza un mejor procesamiento de las variables predictoras aumenta el desempeño medio en la predicción y disminuye el overfitting, tanto el promedio como el mejor alcanzado.

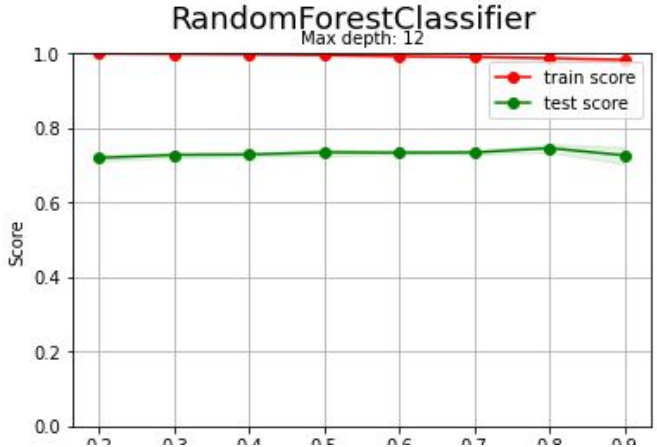
Limpieza	**Tratamiento**	**Desempeño medio**	**BIAS promedio**	**Mejor desempeño**	**BIAS mejor desempeño**
1	**bow**	0.262	0.364	0.507	0.775
	bow y PCA	0.529	0.524	0.620	0.524
	embedding	0.734	0.113	0.780	0.279
2	**bow**	0.328	0.275	0.701	0.829
	bow y PCA	0.643	0.110	0.683	0.304
	embedding	0.709	0.121	0.756	0.301
3	**bow**	0.114	0.094	0.185	0.209
	bow y PCA	0.162	0.052	0.202	0.150

	embedding	0.563	0.080	0.588	0.241

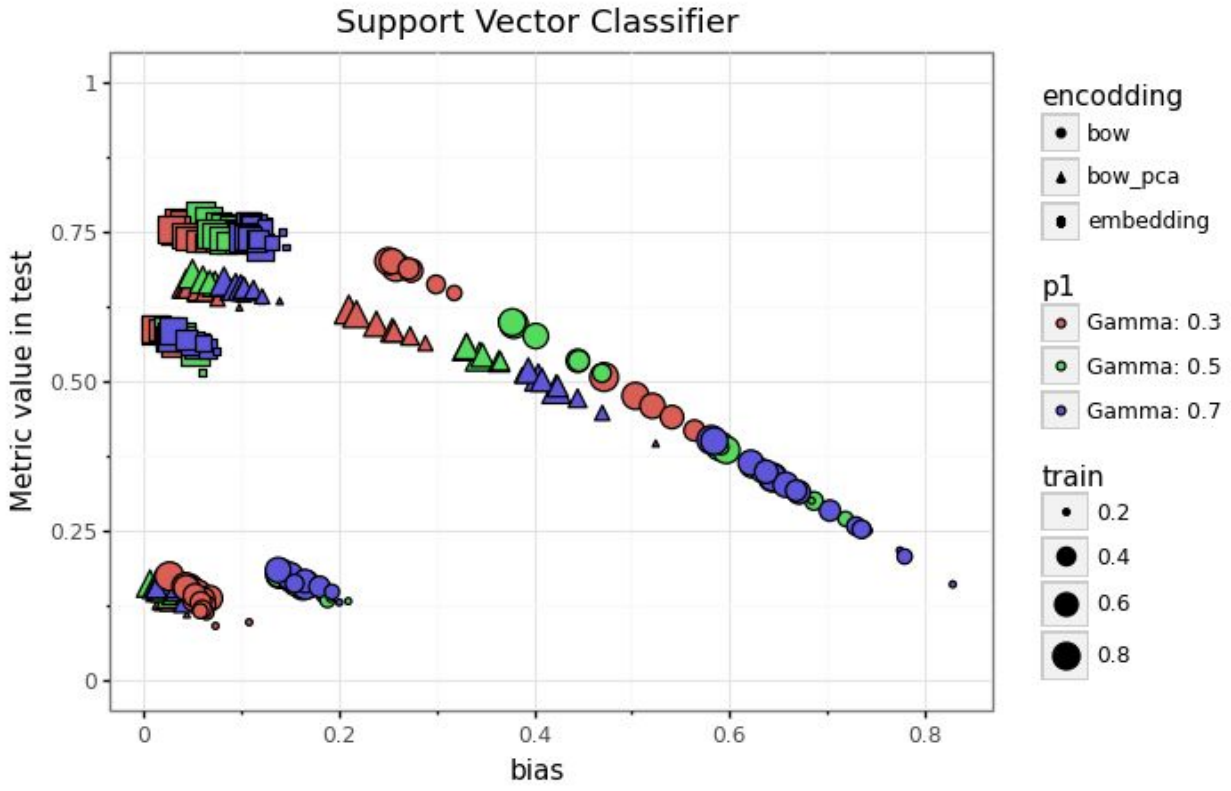
En cuanto a los porcentajes de train y test se tiene que los mejores desempeños se dan cuando el conjunto de train está entre el 70% y el 90% de los datos.

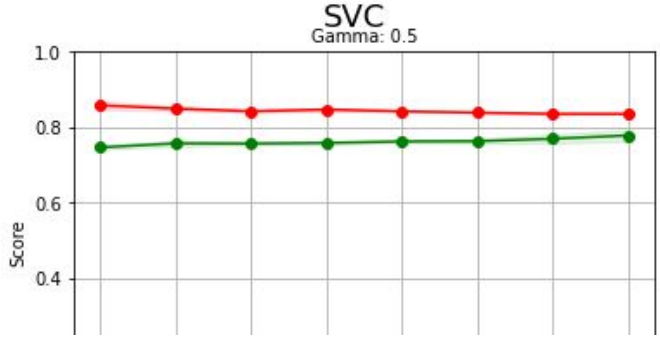
Para el random forest, se observó un mejor performance a mayor complejidad del modelo y a un mayor procesamiento de las variables predictoras, es decir, que en general tiene un mejor desempeño en test para una profundidad máxima de 12 que cuando esta es 6 y mejor aún que cuando esta es 3. Por otra parte, con el overfitting ocurre lo contrario con respecto a las variables de entrada, ya que este es menor en el caso del bow que en el PCA y a su vez en este que sobre el embedding. Teniendo como mejor estimador para este tipo de modelo un random forest classifier con profundidad 12, usando el clean 1, aplicando el embedding y un 80% de los datos en train.



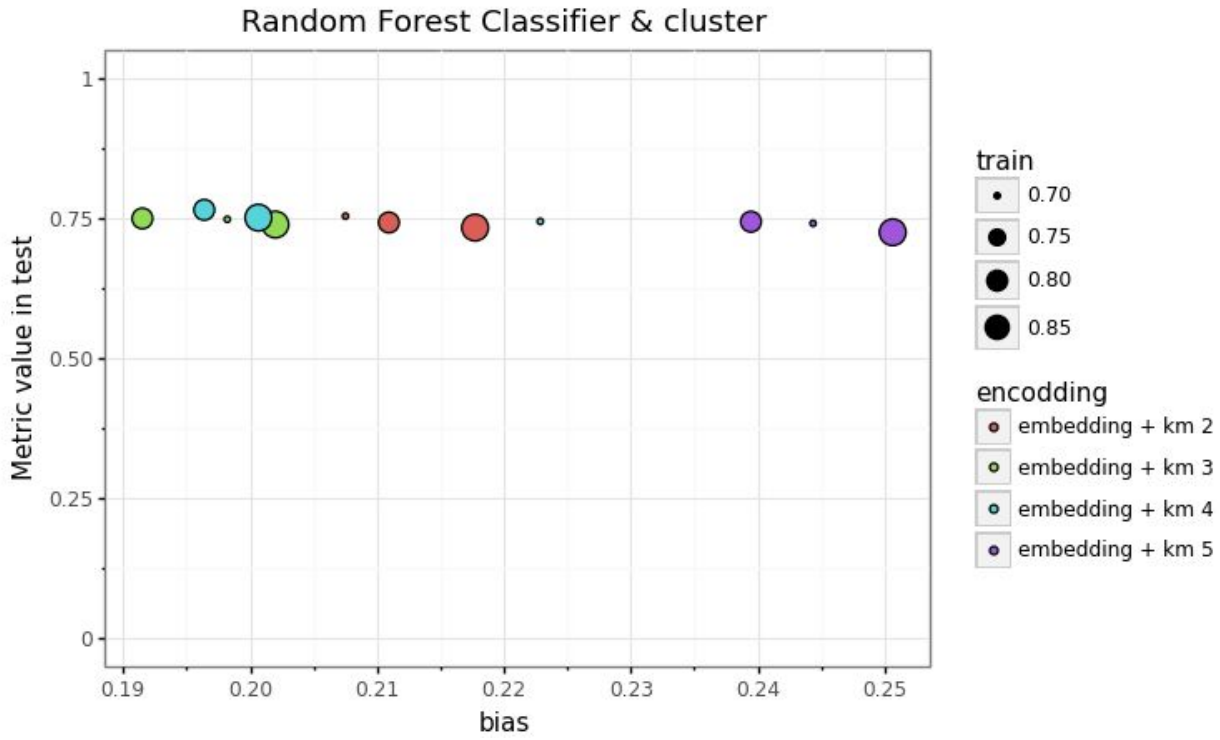


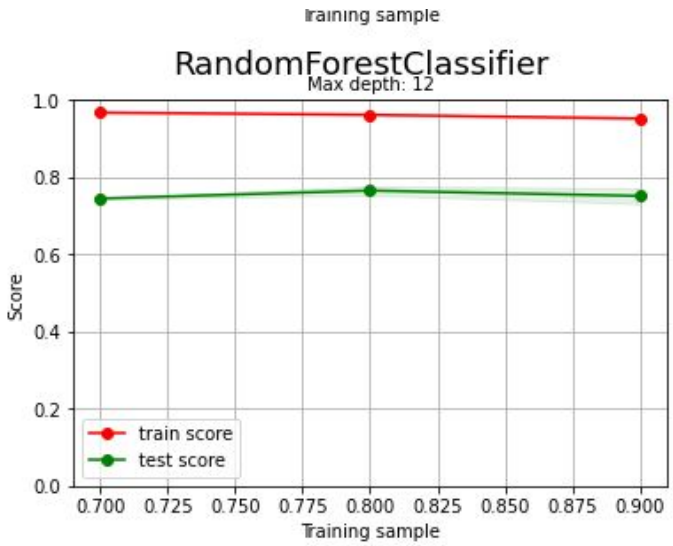
En cuanto al Support Vector Classifier, encontramos que a un mayor procesamiento de las variables de entrada aumenta la tasa de las predicciones y disminuye el overfitting, es decir, que hay un mejor funcionamiento con el embedding y que el bow sin PCA tiene el desempeño inferior. En cuanto al valor del gamma se observa que el modelo con bow, presenta un mejor desempeño para un gamma de 0.3 en dos ocasiones y para 0.7 en una ocasión. Para el bow con pca, el mejor desempeño se da dos veces para gamma igual a 0.5 y una para gamma de 0.3. En el embedding, el mejor desempeño se da dos veces para el valor de 0.3 y una el de 0.5. Lo anterior impide encontrar un patrón claro. La mejor estimación obtenida se logra usando el clean, aplicando el embedding, utilizando el 90% de los datos en train y usando un SVC con un gamma de 0.5.

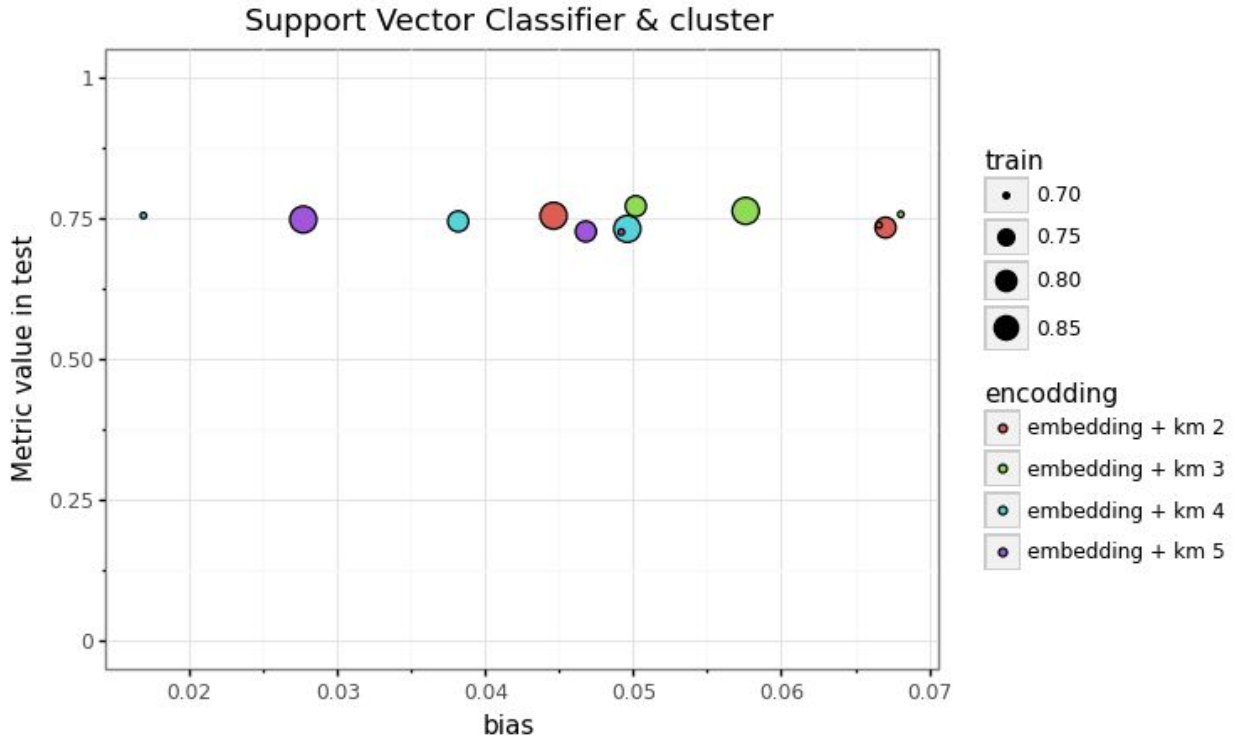


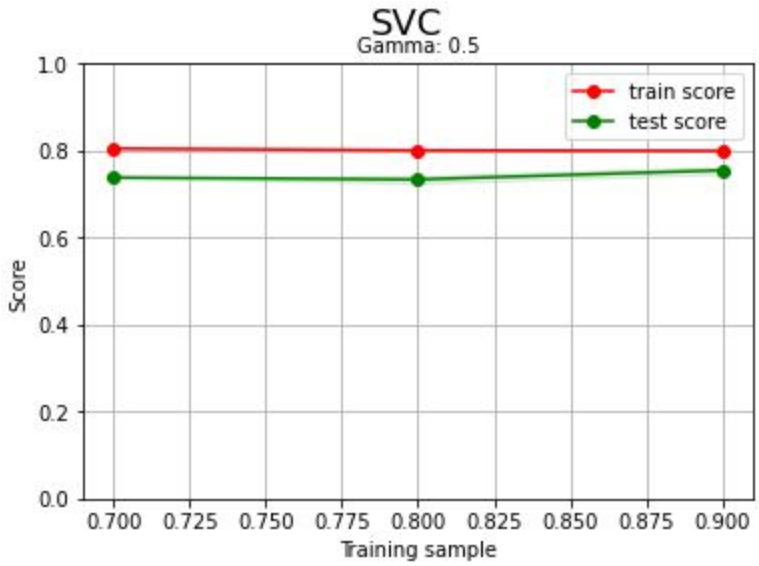


Se encontró que el número de cluster afecta el overfitting, sin mostrar un patrón. Además, se observa que en ninguno de los casos se logra mejorar el desempeño de los modelos anteriores. El mejor desempeño utilizando el random forest obtiene con un 80% de datos en entrenamiento y cuatro cluster, manteniéndose en el mismo valor máximo de 0.746. El SVC en combinación con el kmeans, presenta el mejor desempeño con tres clusters y el 90% de los datos en entrenamiento, sin embargo el desempeño baja de 0.78 a 0.77.









Finalmente de todos los modelos explorados, se selecciona el support vector classifier con un gamma de 0.5, utilizando un 80% de los datos en train, realizando la limpieza 1 y usando un embedding para procesar los datos.

Conclusiones

Para mejorar el desempeño del modelo se sugiere explorar otras limpiezas y otros modelos más específicos para texto, otra opción sería recopilar más datos, sin embargo al explorar el reto en kaggle se encontraron competidores que alcanzaron un mejor desempeño utilizando los mismos datos.

Considerando la naturaleza de los mensajes habituales en twitter y la velocidad de cambio en la red, se espera una complejidad importante o al menos la necesidad de una revisión y retroalimentación constante en los algoritmos relativos a la limpieza y adaptación del texto a las necesidades del modelo. Las alternativas de procesamiento, en especial Embedding, presentan beneficios considerables en cuestiones de costo computacional y eficiencia del modelo. En primera instancia, se recomienda entonces, la constante adaptación de los códigos de limpieza y un análisis profundo de caracteres y elementos que puedan contaminar los mensajes limpios.

El despliegue en producción del modelo requiere en primer nivel, el acceso a suficiente información que amplíe la posibilidad de captar palabras adecuadas en los corpus creados a partir de las mismas y una constante actualización de términos, además de consideraciones especiales ligadas al tipo de lenguaje usado en ciertas regiones.

El desempeño alcanzado por algunos de los modelos planteados, se presenta como una buena alternativa y herramienta para los actores interesados en alertas tempranas ligadas a distintos desastres y la obtención de una herramienta de este tipo, brindaría beneficios económicos y sociales, aumentando el nivel de respuesta de organismos de socorro y propiciando un seguimiento eficaz de mensajes de alerta. Se sugiere que el nivel de desempeño mejore para asegurar la eficiencia de la herramienta, pero en el contexto académico y de aprendizaje, aporta valiosas experiencias y es un excelente resultado.

Referencias

Abadi, Mart'in, Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., … others. (2016). Tensorflow: A system for large-scale machine learning. In *12th $USENIX$ Symposium on Operating Systems Design and Implementation ($OSDI$ 16)* (pp. 265–283).

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

https://github.com/rramosp/20192.ai4eng

Anexos

Se exploraron alternativas de modelos de redes neuronales, en los que se aplicaron distintas combinaciones de parámetros.

Se analizó la incidencia del parámetro de optimización en el resultado de un modelo de igual estructura: Una capa densa de 8 neuronas con activación RELU, una capa densa de 4 neuronas con activación RELU y una capa con una neurona y activación SIGMOID. Se consideraron 128 épocas.

En primer lugar, se revisó el desempeño del modelo haciendo una partición del 30% de los datos para entrenamiento y utilizando SGD como optimizador.

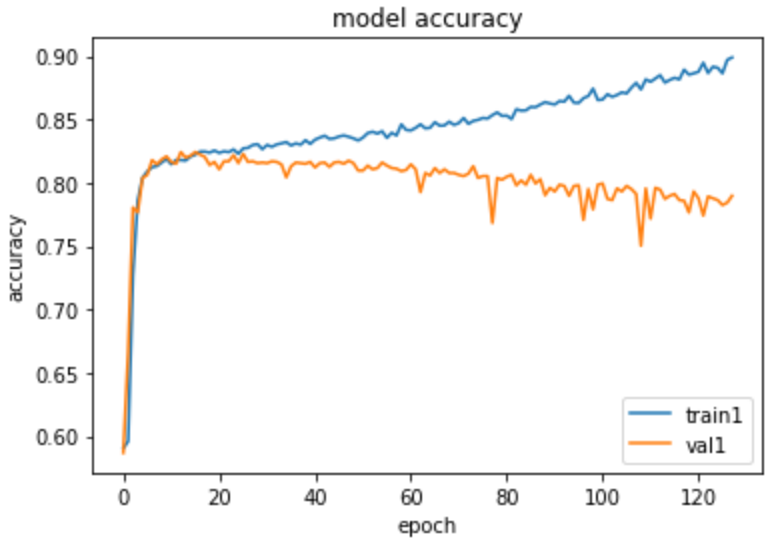


Gráfico Anexo 1: Accuracy del modelo con un SGD como optimizador

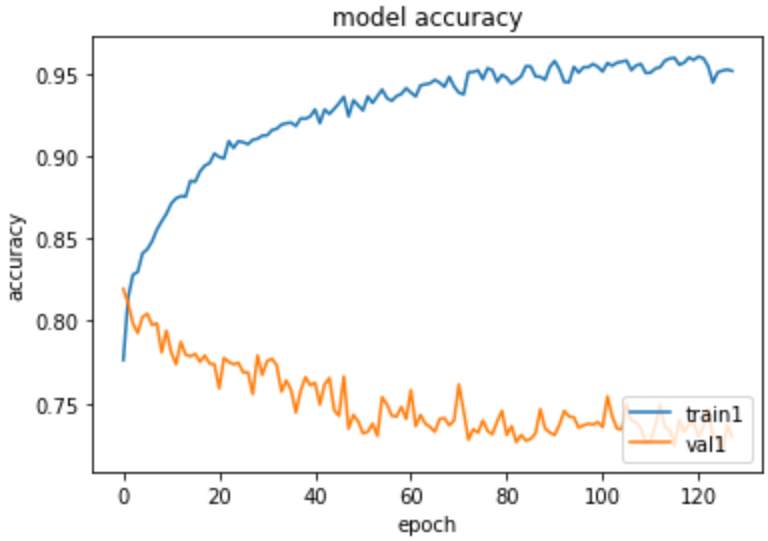


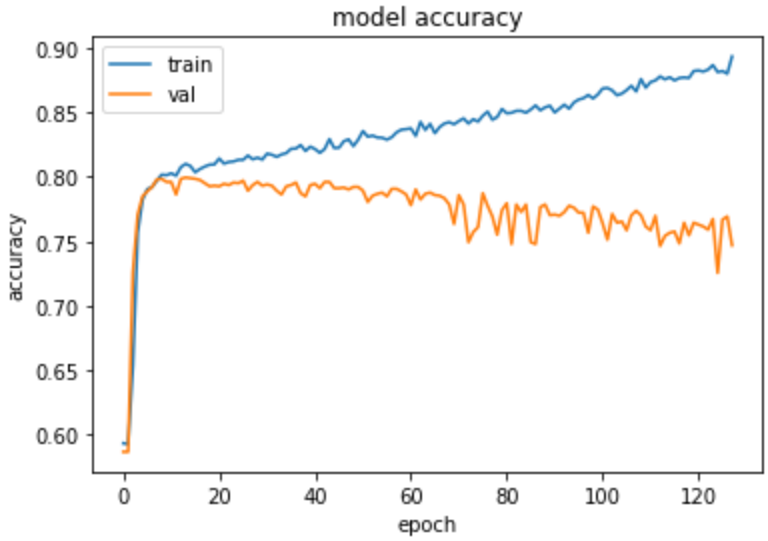
Gráfico anexo 2: Accuracy del modelo con un optimizador ADAM

Se hace evidente el caso de sobreajuste rápido y bajo desempeño que muestra el modelo con el optimizador ADAM (Gráfico anexo 2). Esto podría mejorarse reduciendo la complejidad del modelo o recurriendo a otro optimizador. Por parte del modelo con SGD (Gráfico anexo 1), se observa un desempeño interesante hasta la época 15 aproximadamente, lo que hace este optimizador un buen candidato para el planteamiento del modelo predictivo.

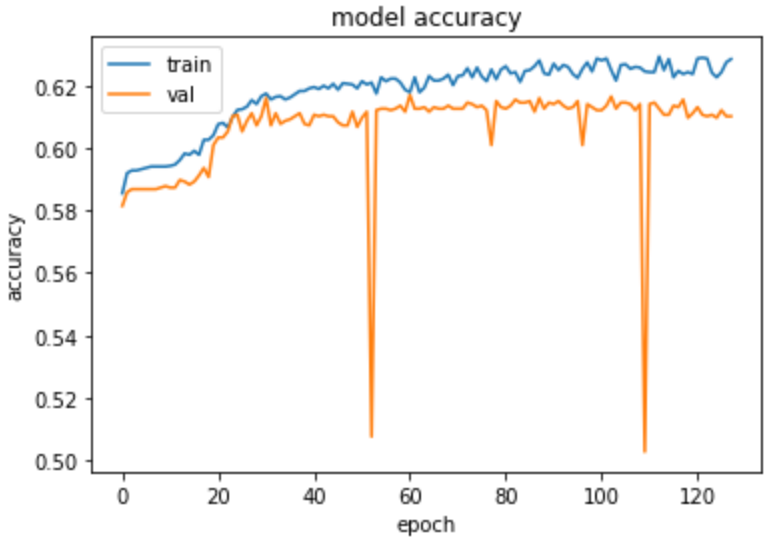
Se realizaron pruebas adicionales reduciendo la complejidad del modelo con el fin de reducir el overfitting y, sin embargo, resultó mejor la medida de accuracy (0.8224) para el test en el caso del primer modelo reportado.

A partir de este análisis, que surgió como un interés de profundización en modelos más complejos, se puede hablar de una aproximación con un nivel interesante de aproximación. Cabe resaltar que la medida de precisión usada no es igual a la propuesta en el problema.

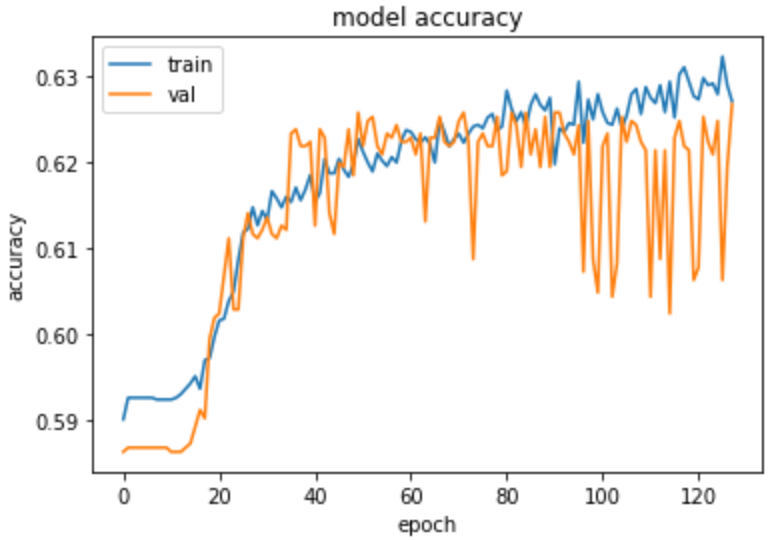
Se aplicó el mismo modelo al conjunto que parte de la limpieza 2, pero los resultados no fueron tan prometedores como el anterior pero siguen siendo una buena alternativa.



Posteriormente, se aplicó el modelo a los casos en los que se aplicó PCA al BOW y los resultados fueron considerablemente inferiores y, pese a que el overfitting se dió desde épocas elevadas, el comportamiento de la predicción se notó bastante inestable, lo que podría mejorarse si se optimiza la complejidad del modelo.



Desempeño del modelo con PCA aplicado al BOW del clean 1



Desempeño del modelo aplicado al PCA del clean 2