FACULTAD DE INGENIERIA DE LA UNIVERSIDAD DE BUENOS AIRES

Organización de Datos

Trabajo Práctico Nº2

Martín Monzó

08/10/2020

Repositorio disponible en: https://github.com/martinmonzo1/Datos_TP2

Introducción

El objetivo de este trabajo es desarrollar un algoritmo de Machine Learning que sea capaz de predecir si un tweet está haciendo (o no) referencia a una catástrofe real. El desafío fue planteado en la competencia de Kaggle titulada “Real or not? Disaster tweets”, que provee tanto el train set como el test set. El primero de ellos fue el objeto del análisis exploratorio de datos que quedó enmarcado en el Trabajo Práctico N°1 de la materia.

En este informe se podrán notar dos partes bien diferenciadas: Por un lado, aquella en donde, efectivamente, se explicita el camino que llevó al algoritmo de mejor performance; por otro lado, conformando una especie de antesala, aquella en donde se expone todo el preprocesamiento de la información y la creación de los atributos que serán el alimento para los algoritmos.

Primera parte

Repaso del Trabajo Práctico N°1

El TP1 a través de todos los insights que ha llegado a recoger, representa la fuente de información de la que se nutre el TP2 y no puede entenderse el uno sin el otro. Es por esto que, como primer medida para comenzar este informe, se hará un breve repaso de algunas conclusiones que hayan podido obtenerse y se desarrollarán rápidamente algunos puntos que no han sido abordados con el énfasis suficiente.
Entre las características del set de datos que fueron mencionadas y que se revisarán en las secciones siguientes, estaban:

- El interrogante de cuál debía ser el tratamiento de los NaN en *keyword* y en *location*; el desbalance en target;
- la distribución de características particulares de los tweets (hashtags, menciones, hipervínculos, etc);
- la hipótesis de que los tweets que se correspondieran a catástrofes verdaderas incluyerían una estructura que que los otros no

Luego, entre aquellas que no se examinaron con la intensidad suficiente pero que fueron tenidas en cuenta para este informe:

- Vocabulario, frecuencias de términos, stopwords, etc.;
- relación ente las variables.

Pre procesamiento del texto

El objetivo aquí es llevar al texto a una forma que, en lo que a contener información respecta, sea mucho más densa. Esto implica deshacerse de cualidades que no aportan demasiada información y solo logran entorpecer el funcionamiento de los algoritmos. Sin embargo, no todo lo que de limpia del texto va a la basura y se verá que el producto de la poda puede desembocar en un nuevo feature.

* Contracciones: En el idioma inglés, el uso de contracciones en el texto es algo particularmente recurrente y aún más en el texto informal de los tweets. Por ejemplo: Escribir “there’s” en lugar de “there is” o “I’m” en lugar de “I am”.
Permitir la presencia de este tipo de palabras extiende el vocabulario innecesariamente, haciendo que la vectorización sea menos eficiente y requiera más procesamiento o provocando que la palabra quede catalogada como desconocida en el contexto de embeddings pre-entrenados a pesar de que, estando separadas, sean bien comunes. Además, es común que (por lo menos) alguna de las palabras contenidas en la contracción sean stopwords y, al romper con la contracción, será más sencillo sacarlas a partir de una lista de stopwords más concisa.
Quitarlas es sencillo y solo implica recorrer el texto hasta encontrar aquella que pertenezca al espacio de direcciones de claves de un mapa de contracciones para luego reemplazarla por su versión extendida.

* Características particulares de los tweets: No todas son tratadas de la misma manera. Si la característica en cuestión es una mención o un hipervínculo, se eliminan del texto sin más. Esto es así porque se puede suponer que ni los nombres contenidos en las menciones ni el conjunto de símbolos representativos de un URL son material de vocabulario. A diferencia de ellos, los hashtags contienen palabras pertenecientes al lenguaje (en su gran mayoría, al menos) y se perdería información quitándolos. Entonces, en este caso solo se retira el numeral (#).
El modus operandi para estas operaciones no incluye más que el uso de Regular Expressions.

* Casing: Se pasa todo el texto a minúsculas. Esto no sólo es útil para reducir el tamaño del vocabulario sino que es imperativo para el uso de embeddings, ya que no todos ofrecen vectores pre entrenados con minúsculas y mayúsculas. Además, los que lo hacen son mucho más extensos y requieren de mucha memoria para manejarlos.
Hay que tener cuidado con llevar el texto a minúsculas antes de tokenizar. Lo que sucede es que le estaríamos quitando un recurso útil a los algoritmos de tokenización que se valen de las mayúsculas para evitar hacer separaciones equivocadas. El ejemplo típico es alguna frase que contenga algo como “Mr. Smith”. Normalmente, el tokenizador puede reconocer que este punto no se corresponde con el final de una oración, pero con minúsculas ya no es así.

* Caracteres numéricos: Este tipo de caracteres se limpian del texto sin más.

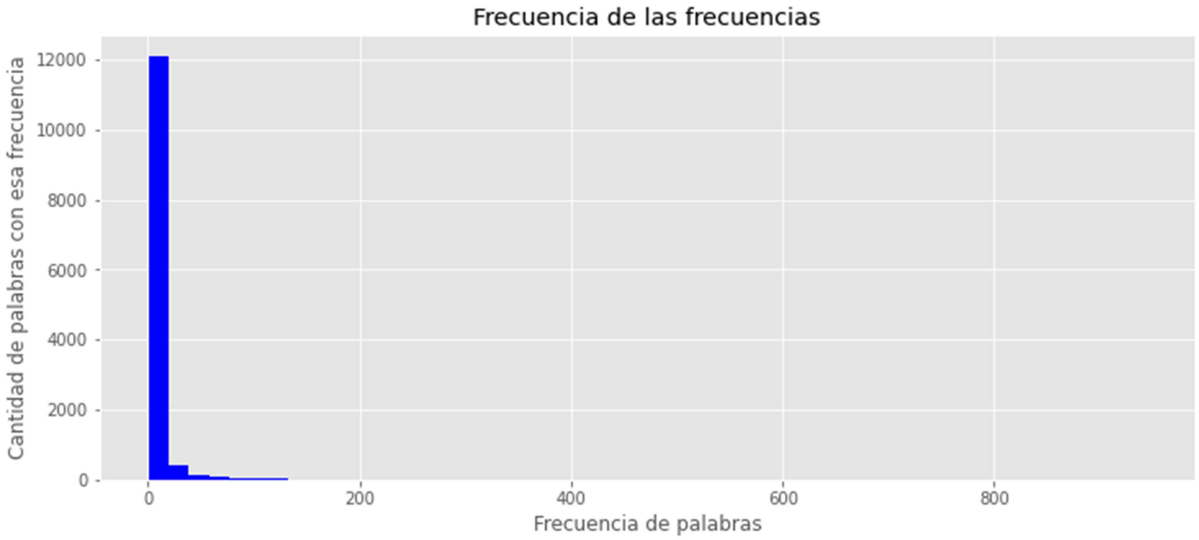
* Stemming: La operación de stemming es muy sencilla: La idea es quedarse únicamente con la raíz de las palabras. El procedimiento mediante el cual esto se realiza consiste en descartar todas las letras que aparezcan en posteriori a la raíz. Así, por ejemplo, tanto el stem de ”painting” como el de “painted” resultan en “paint”. Se puede ver como esto permite llevar el vocabulario a una forma más concisa. Sin embargo, hay que mencionar que no todos los ejemplos son así de coherentes.
La conocida librería NLTK contiene las herramientas para esta operativa que se realiza luego del tokenizado previo correspondiente.

* Stopwords: Las stopwords son palabras sumamente comunes en un lenguaje, de alta frecuencia de aparición y que cumplen la función de rellenar espacios que terminan de darle color a una frase. Sin ellas, las cosas suenan raro, es así de sencillo. Pero también es cierto que, más allá de eso, nunca la información de una frase está contenida en ellas y resultan prescindibles para el entrenamiento de un algoritmo. Entonces, y en pos de mejorar la performance, se quitan. En el idioma inglés, son stopwords palabras como “a”, “the”, “is”, etc.

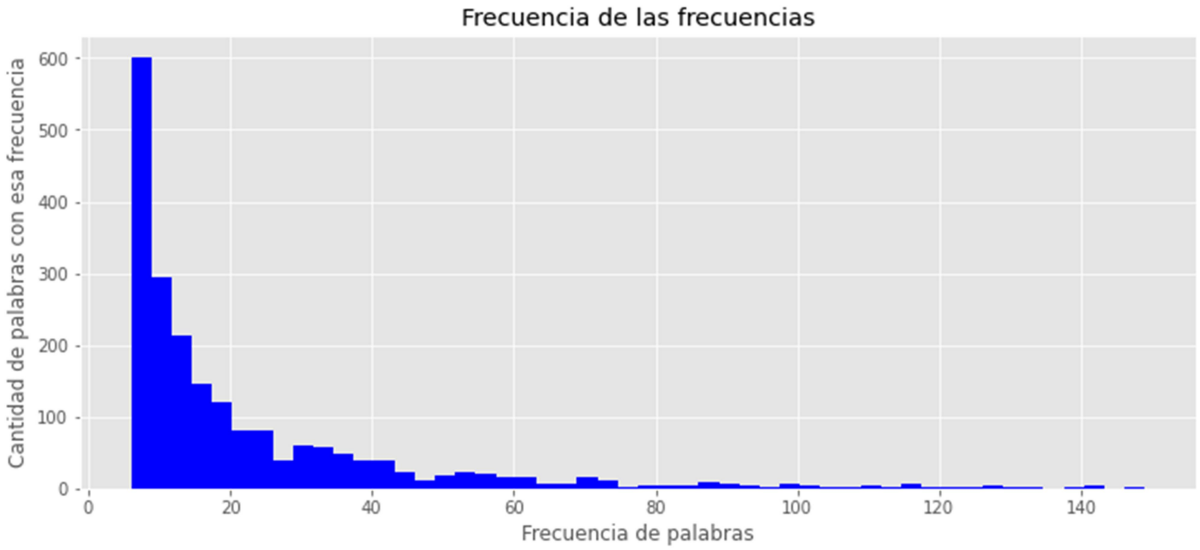
El proceso de quitarlas es muy similar a aquel de las contracciones. Pero mientras que es fácil decir si una contracción lo es o no, lo mismo no ocurre con las stopwords (sin contar casos obvios como los ejemplos anteriores. Para este trabajo se decidió incluir en la lista de stopwords a aquellas que aporta NLTK, reforzando este conjunto con distintos signos de puntación, que son algo que suelen abundar en los tweets.

* Conformación del vocabulario: Las operaciones de vectorización del texto necesarias para generar la entrada a los algoritmos trabajan en base a un vocabulario. Es importante que el mismo sea representativo del texto contenido en el set de datos y que pueda generarse cada uno de los datos en base a las palabras que contiene. Lamentablemente, el caso extremo de conservar todas las palabras habidas y por haber no es posible porque la pérdida en performance de los algoritmos y la consecuente lentitud del entrenamiento no se vería recompensada en un aumento de la calidad del resultado final. Es necesario un balance y de eso se trata el preprocesamiento del texto hasta aquí descrito.
Podríamos decir que la mayor parte del trabajo ya está hecho y que los datos preprocesados tal como están ya pueden generar el tan preciado vocabulario. Pero no. Como se puede observar en el siguiente gráfico, un vocabulario así conformado tendría dos particularidades que lo hacen menos apto:

- La mayoría de las palabras solo aparecen unas pocas veces o, incluso, una única vez. Casos tan fuera de lo común no permiten sacar conclusiones y, en consecuencia, no son útiles para entrenar un algoritmo. En definitiva, es tan raro que aparezcan en tweets de catástrofes reales como de catástrofes no reales.
- Hay un puñado de palabras que aparecen muy seguido y se repiten a lo largo y ancho del set de datos. Estas palabras son ni más ni menos que… ¡Stopwords!



Para atacar estas particularidades, establecemos un intervalo de frecuencia y se descartan del vocabulario todas aquellas que queden por debajo o por arriba de los límites.
El resultado es un vocabulario de alrededor de 2000 palabras, con la siguiente distribución:



Un último detalle a tener en cuenta son los N-gramas con N>1, especialmente por lo señalado en el segundo de los ítems expuestos recién respecto a la frecuencia de palabras. Existe la posibilidad de que términos muy repetidos, aparentemente stopwords, cobren importancia cuando se los considera contenidos es un contexto, i.e., prestando atención a las palabras que tiene alrededor. La prueba de incluir bigramas en el vocabulario fue hecha y los resultados se mantuvieron sin marcados cambios respecto al uso exclusivo de unigramas; también se probó incluir trigramas, que directamente empeoraron los resultados.

Feature engineering: Extracción de características del texto

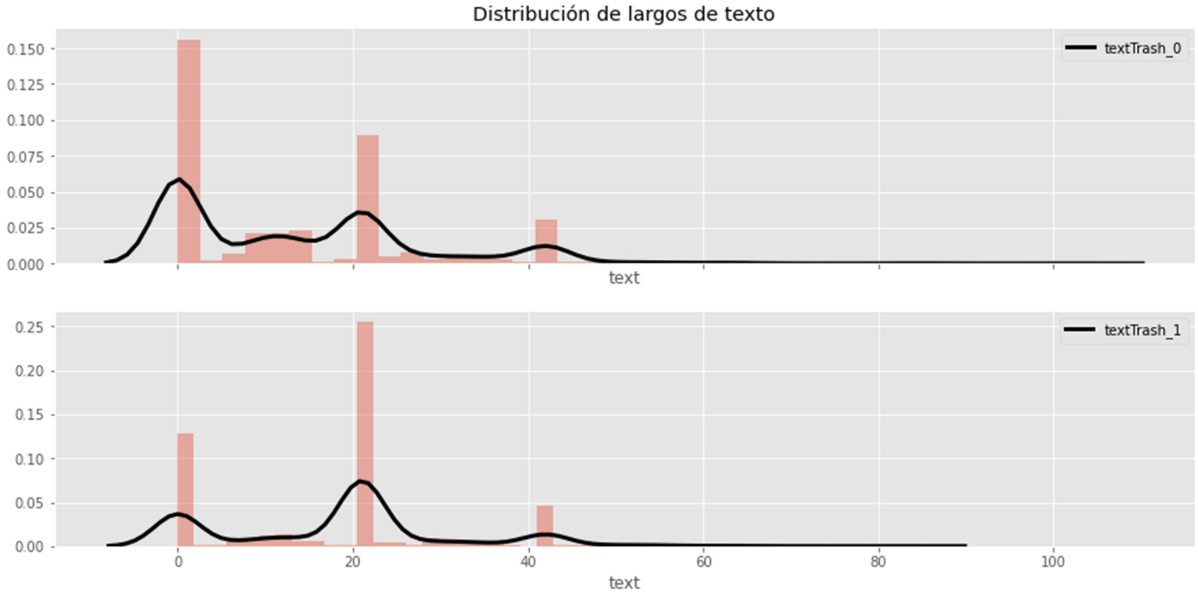
Cada uno de los features que se extrajeron del texto es del tipo numérico. Sin embargo, la naturaleza de cada uno de estos features suele ser completamente distinta porque así lo es su origen. Mientras algunos solo toman valores binarios, otros solo toman enteros positivos, otros valores que están entre -1 y 1, etc. Algunos se desprenden del preprocesamiento del texto, otros no. En fin… A continuación se hará un repaso de todos ellos.
Algo que hay que aclarar es que lo features a continuación listados no necesariamente pasarán a formar parte de las entradas que se usen para entrenar los algoritmos.

* Location in Text y Keyword in Text: Ya se ha mencionado que la relación entre los atributos originales es una temática que no fue explorada exhaustivamente en el TP1. Este es un intento de reivindicación.
El título lo explica por sí solo. La idea es generar, para cada uno de los atributos originales *location* y *keyword,* un feature que toma valores binarios para cada dato de acuerdo a si el término se encuentra (1) o no (0) en el texto.

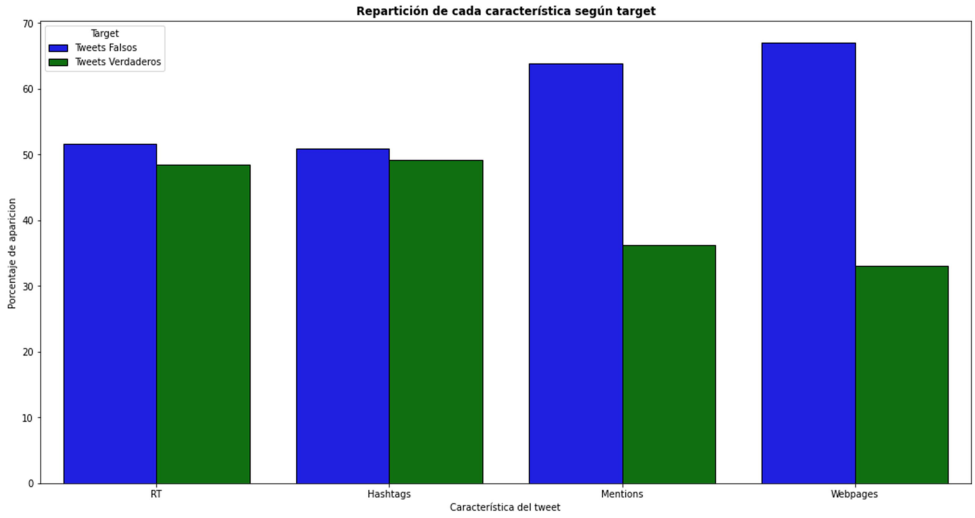
* Presencia de contracciones: De forma similar, se tiene en cuenta la presencia de contracciones en el texto, pero yendo un poco más lejos y usando un contador en lugar de un valor binario.

* Largo del texto: Ni más ni menos que la longitud del string que representa al texto del dato. Se ha intentado darle una vuelta de tuerca al registrar este valor en distintas etapas del preprocesamiento. Primero, luego de que se hayan reemplazado las contracciones; posteriormente, una vez que se han quitado menciones y URLs; por último, al quitar las stopwords

y dar por finalizado el preprocesamiento. Estos valores no son tan interesantes por sí solos como lo son cuando se toman las diferencias. Es ahí cuando se ven diferencias en las distribuciones del atributo para uno y otro target. Para ilustrar esto, en el siguiente gráfico se muestra la distribución de la diferencia de largo entre el texto original y aquel posterior al descarte de menciones y URLs, para cada uno de los target:



* Hashtags, menciones y URLs: El caso de las menciones y las URL es un calco de aquel de las contracciones. Es decir que el nuevo feature viene a dejar registros de cuántos había antes de que el preprocesamiento los borrara de la faz de la Tierra. El caso de los hashtags es similar pero con la salvedad de que es todo ganancia porque la palabra correspondiente al hashtag permanece en el texto.
El hecho de haber usado un contador en lugar de un valor binario no es arbitrario en este caso y responde a un análisis hecho en el TP1, en donde se analizó en qué proporción estas características se encontraban en el texto de cada uno de los targets. El resultado fue que, para cada una de las tres características y cada uno de los *target,* el valor porcentual siempre rondaba el 50%. La mayor diferencia ocurría en el caso de las menciones y las URL, que eran contenidas en un mayor porcentaje por los tweets del *target 0*. Dicho gráfico se reproduce a continuación:



* Tagging: Este es, en realidad, un conjunto de features. La idea fue capturar el tipo de palabras que puebla el texto de los tweets, denominado en inglés como Part of Speech (PoS).
Existen distintos conjuntos de categorías para llevar a cabo esta tarea y algunos son más específicos (y, por lo tanto, complejos) que otros. Por ejemplo, mientras una opción es hacer una distinción más básica entre sustantivos, adjetivos, verbos y demás, otras opciones van más allá y agregan subcategorías. En este caso, se hizo uso de una diferenciación bastante básica, como la descrita en el primer ejemplo. La razón radica en que se consideró más apropiado tener un vector con algunas categorías bien marcadas a otro en donde los valores quedaran más repartidos entre las distintas subcategorías. Las categorías son:

Tag	Meaning	English Examples
ADJ	adjective	*new, good, high, special, big, local*
ADP	adposition	*on, of, at, with, by, into, under*
ADV	adverb	*really, already, still, early, now*
CONJ	conjunction	*and, or, but, if, while, although*
DET	determiner, article	*the, a, some, most, every, no, which*
NOUN	noun	*year, home, costs, time, Africa*
NUM	numeral	*twenty-four, fourth, 1991, 14:24*
PRT	particle	*at, on, out, over per, that, up, with*
PRON	pronoun	*he, their, her, its, my, I, us*
VERB	verb	*is, say, told, given, playing, would*
.	punctuation marks	*. , ; !*
X	other	*ersatz, esprit, dunno, gr8, univeristy*

El resultado, para cada dato, es un vector en el que cada posición se corresponde con una PoS y su valor es la cantidad de apariciones de la misma. Este vector se separa en tantas features como componentes tiene.

* Tagging ratios: Para este grupo de features se buscó capturar distintas relaciones que, de alguna manera, resumieran y fueran representativas de la distribución de PoS en el texto de cada dato que se acaba de explicar. La idea no es reemplazar sino exponer.
Se proponen tres ratios:

- Noun Ratio: Cociente entre la cantidad de sustantivos contenida sobre la cantidad de palabras del texto de ese dato.
- Most Common PoS Ratio: Cocientre entre cuántas categorías alcanzan el valor máximo para ese dato sobre la cantidad de categorías que contiene. Da una idea de preponderancia y si hay una PoS dominante o la cantidad de palabras se distribuye de manera pareja entre las categorías que ocupa. En otras palabras: De todas las categorías que contiene, ¿Cuántas alcanzan el valor máximo?
- PoS Ratio: Cantidad de categorías ocupadas sobre el total de categorías posibles.

Nótese que el PoS Ratio siempre tiene el mismo denominador y su numerador es un valor que va entre cero y el valor del denominador, sea cual sea el texto. Sin embargo, tanto el Noun Ratio como el Most Common PoS Ratio requieren otro tipo de cuidados que conciernen al hecho de que el texto de cada dato tiene un largo distinto y, por lo tanto, distintas cantidades de palabras para llenar las categorías de PoS. A pesar de eso, puede ocurrir que valores de estos ratios sean el mismo para textos de largo 2 o textos de largo 100, por poner un ejemplo numérico. Esto está equivocado desde el punto de vista y de la estadística y es por eso que el valor que se toma para cada uno de esos ratios es el que manda el límite inferior del intervalo de confianza de Wilson.

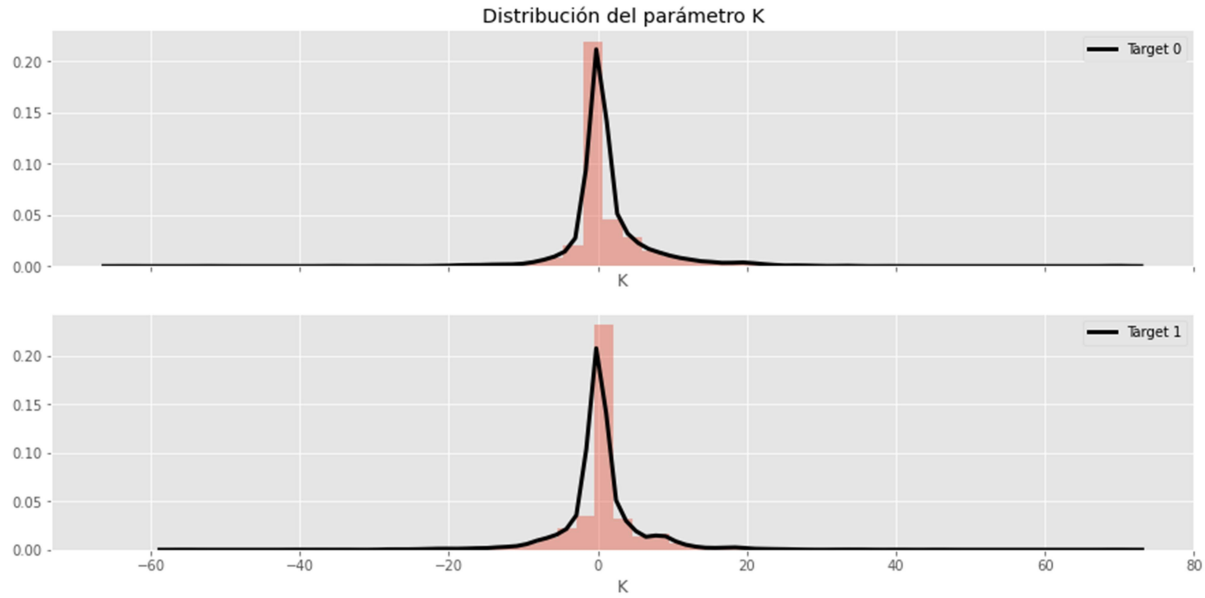
* Polarity y Subjectivity: La librería TextBlob creada para NLP contiene un algoritmo de Naive Bayes ya entrenado que permite determinar estas características en el texto. Para obtener estos features simplemente se aplicó el método que es parte de la librería sobre el texto.

* Headlines Distance: Quizás la hipótesis más fuerte planteada en el TP1 fue aquella que postulaba posibles similitudes entre la estructura del texto de los tweets que referían a catástrofes reales y los titulares de noticias, diferenciándose así del carácter desestructurado que suelen presentar los tweets en el caso general. En ese momento se había buscado probar esta hipótesis a través del análisis de la longitud del texto antes y después del preprocesamiento. Este feature busca, de alguna manera, reiterar el análisis incluyendo todos los features hasta este punto extraídos.
La pregunta entonces es: ¿Si este grupo de features se extrajeran del titular de una noticia, tomarían valores similares que aquellos que toma para los tweets de catástrofes verdaderas?
El set de datos en '/abcnews-date-txt.csv' contiene un millón de titulares de noticias. De todos esos se tomó una muestra de 10000 datos a los que se le aplicó la extracción de features para, finalmente, promediar los resultados de todos ellos y obtener así un valor de referencia para cada uno de los features en forma de vector. El paso posterior se trata de medir qué tan cerca de esta referencia caen los vectores de features correspondientes a los tweets. Se utilizó la distancia coseno para realizar esta comparación y el resultado es el valor del nuevo feature.

* Parámetro K: Este es un feature pensado casi exclusivamente para los algoritmos basados en árboles de decisión, que no son capaces de establecer relaciones entre los features por sí solos.

```
DF['K'] = DF['textLenght']*DF['NOUN/TOT']*DF['polarity']
```

Esto tiene todo el aspecto de ser algo completamente arbitrario y lo es. La ecuación es el resultado de haber probado operaciones entre parámetros hasta que surgiera algo cuyo resultado llevara a distribuciones de la variable para uno y otro target que tuvieran una diferencia marcada. En este caso sucede que la distribución de K para el target 0 y el 1 ocurre de manera muy marcada en las adyacencias del cero, pero una está del lado de los positivos y otra del lado de los negativos.



Feature engineering: Encodings

Dos de los tres atributos originales del set de datos corresponden a variables categóricas: *location* y *keyword*. En este apartado será tratado el encoding de ambas.
Se usa One-Hot Encoding. El método generará tantas nuevas features como categorías haya, que podrán tomar los valores cero o uno según la categoría no esté o esté en el dato. Otra manera de verlo, teniendo en cuenta que los valores para estos dos atributos toman un único valor para cada dato es que estamos generando un vector fila cuyas componentes son todas cero a excepción de aquella categoría a la que adhiere el dato.
De los dos, el que más categorías contiene es *location* con 3341 categorías distintas, quizás demasiado para un One-Hot Encoding; *keyword* lo sigue con 221. Eso sin contar los NaN, con ocurrencias en ambos.
Uno de los cuestionamientos que había quedado pendiente del análisis exploratorio de datos del TP1 era qué tratamiento darles a estos NaN. Ahora que se sabe la operación que se quiere efectuar sobre estos atributos, conocemos la respuesta: Nada. Sucede que el algoritmo de encoding entrenará a partir de las categorías reales, sin incluir los NaN. Por eso, una vez que se quiera transformar el atributo y aplicarle el encoding, el algoritmo tomará a los NaN como categoría desconocida a los que les asignará el vector nulo.
Sin embargo, el hecho de que un tercio de los datos de location sean nulos y produzcan entonces vectores nulos trae el interrogante de qué tan útil puede llegar a ser e, incluso, si no será contraproducente. Se sabrá próximamente.

Feature engeneering: Vectorización del texto

Ya se ha hecho mención en este informe de la importancia del vocabulario para vectorizar el texto pero nada se ha dicho de esto último. Vectorizar el texto no es más que dotar al texto de una representación numérica capaz de ser tomada por los algoritmos de machine learning. En búsqueda de la mejor solución, se han probado los siguientes métodos:

- Bag of Words y TF-IDF
- Hashing Vector
- Embeddings pre-entrenados

Tanto la operación de Bag of Words (BoW) como la de TF-IDF dan como resultado, para cada dato, un vector de longitud igual a aquella del vocabulario en el que cada componente se corresponde (de manera invariable para cualquier dato sobre el que se aplique la operación) a uno y solo uno elemento del vocabulario. Se hace más claro ahora el porqué de insistir tanto en disminuir el tamaño del vocabulario y por qué, a pesar de ser opcional el alimentar el algoritmo con un vocabulario propio, es sumamente recomendable. La diferencia entre ambos es que las componentes del vector de BoW toman un valor binario de acuerdo a presencia de la palabra; mientras que las componentes del vector de TF-IDF modifican el valor binario para tener en cuenta la frecuencia de la palabra en el texto y la importancia de la palabra para el vocabulario.
Una variante que se consideró fue aplicarle una reducción por valores singulares al resultado de TF-IDF. La implementación mediante Scikit-Learn es muy sencilla porque basta incluir el algoritmo de SVD en un Pipeline junto con el clasificador elegido para poder buscar, mediante algún algoritmo de tuneo de hiperparámetros, el mejor valor de cantidad de valores singulares a considerar junto con el resto de los hiperparámetros del modelo.
Hashing Vectors realiza un procesamiento similar en el sentido de que entrega vectores de una longitud definida por el usuario que es ni más ni menos que el espacio de direcciones al que hashea cada uno de los tokens del texto de un dato cuando el algoritmo les aplica su función de hashing.
Entre estas opciones, fue la combinación de TF-IDF con SVD la que generó los mejores resultados sobre varios algoritmos basados en árboles de decisión. Sin embargo, nada pueden hacer frente a los embeddings.
Los embeddings son representaciones vectoriales de palabras previamente entrenados, de manera que los valores de sus componentes no son arbitrarios sino que son tales que palabras similares tendrán vectores similares, teniendo en cuenta como palabras similares no solo a aquellas que lo son letra por letra sino también a aquellas que se usan en contextos análogos. La manera de utilizarlos varía según el modelo de Machine Learning que se utilice: Para el caso de las redes neuronales, se construye una matriz de embeddings a partir de los vectores y el vocabulario y se usa esta matriz para configurar la capa correspondiente a la operación de embedding; en el resto de los algoritmos, la opción utilizada fue tomar el embedding del texto de un dato como aquel que se corresponde con el promedio componente a componente de los vectores de cada una de las palabras que lo componen.
Existe un número de conjuntos de embedding preentrenados que se pueden descargar de manera gratuita. Para este trabajo se utilizó un conjunto de embeddings desarrollado y entrenado por GloVe a partir de léxico utilizado en Twitter que resultó sumamente conveniente, pues fue capaz de cubrir más del 99% del vocabulario.

Data augmentation:

A pesar de que este apartado forma parte de la Primer Parte del informe, la realidad es que esta práctica no fue tenida en cuenta hasta bien entrado el período de pruebas sobre algoritmos basados en redes neuronales. Como se verá más adelante en el apartado correspondiente, un síntoma que se dejaba ver en los gráficos de loss y accuracy para el entrenamiento y la validación a lo largo de las epochs, era una curva de validation loss que, luego de alcanzar un mínimo a las pocas epochs, comenzaba a incrementar su valor. Esto es una clara señal de overfitting que se veía

refleja luego al evaluar los resultados de las predicciones sobre el test set, que arrojaba valores de accuracy aún peores que aquellos de la validación.
A raíz de estos acontecimientos es que se decidió extender el set de datos con el que se entrenaban los modelos. Tres posibilidades distintas fueron evaluadas, a saber:

- Extender el set de datos manteniendo la desproporción en el target
- Extender el set de datos solo lo suficiente como para balancear las clases (es decir, solo se agregan datos de la clase minoritaria hasta alcanzar equilibrio)
- Extender el set de datos de manera tal que, además, se balanceen las clases.

La forma en que se fue materializada cualquiera de las tres opciones es la misma y se basa en agregar ruido. La operativa radica en clonar registros del set de datos original e introducir variaciones en el texto que luego, al extraer los respectivos features, resultarán también en valores que presentan una variación frente a aquellos que toman los del registro original. El resultado, entonces, es un set ruidoso que se parece bastante al original sin ser igual y que queda a disposición para cumplir con cualquiera de las tres opciones que hemos propuesto.
El interrogante que está quedando por contestar es cuál fue el criterio bajo el cual se intrujeron las variaciones. Se pueden considerar dos partes: Primero, se reemplaza cada una de las palabras en el texto por otra que, perteneciendo al vocabulario conformado, tenga el vector de embedding más cercano, midiendo esta cercanía con la distancia coseno; como segunda medida, se inserta una palabra al azar (también perteneciente al vocabulario) en una posición aleatoria del texto. De esta manera, quedan cosas del estilo del siguiente ejemplo:

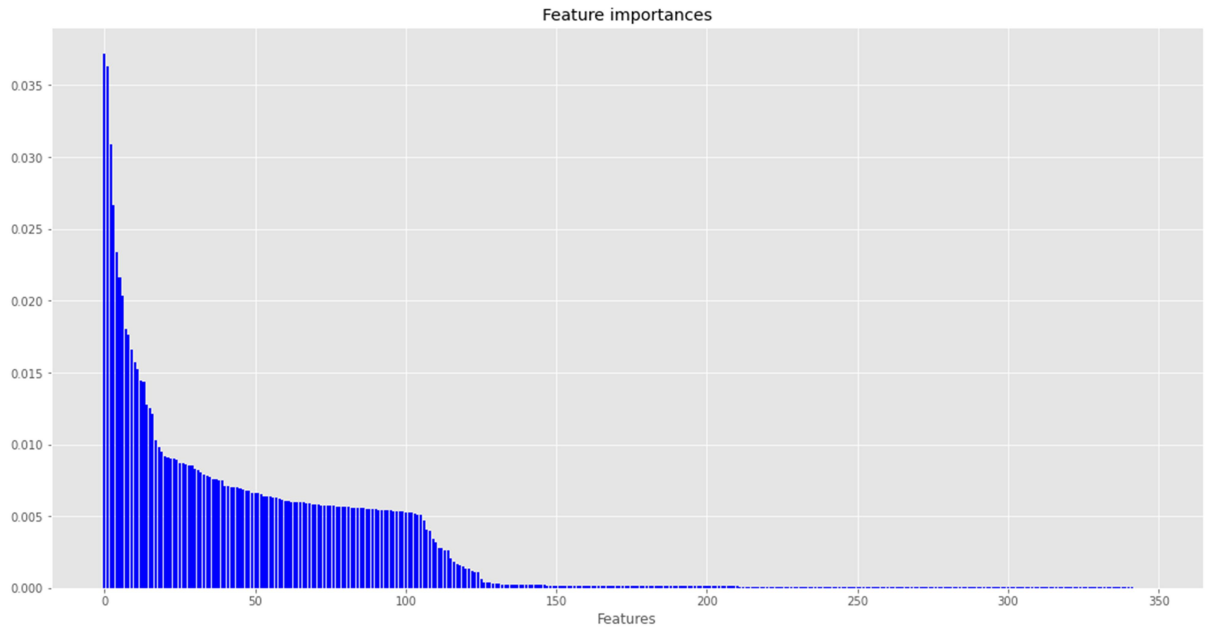
Versión original: might killed airplane accident night car
Versión adulterada: could murdered plane crash tonight truck inch

De las tres opciones, aquella del set de datos extendido y balanceado fue la que dio los mejores resultados, mejorando los resultados de accuracy y f1-score en los algoritmos basados en árboles y mejorando la convergencia de las curvas de loss en el caso de los algoritmos basados en redes neuronales. El corolario es que resulta más fácil confiar en la validación y suponer que, luego, los valores de las métricas tomadas a partir de las predicciones del set de test serán similares.
Más de esto en apartado siguiente.

Feature importance

Anteriormente se han enumerado y desarrollado las características de los features que llegaron a estar en consideración para alimentar a los algoritmos. Pero el hecho de que hayan sido creados no significa que efectivamente se los vaya a utilizar. Todo depende los resultados que puedan llegar a dar y, para eso, es necesario tener alguna manera de evaluar y llevar registro de los cambios en dichos resultados.
Gran parte de este apartado podría estar en la segunda parte de este trabajo y contar como uno más de los modelos que fueron probados. A pesar de eso, se lo ubica aquí porque, a lo largo del trabajo, esta algoritmo fue considerado más para la tarea de diagnóstico que para la de predicción. Random Forest fue el primer algoritmo que se probó y, si bien es cierto que en ese momento se le dio el trato adecuado y necesario para llegar a tener un buen clasificador binario (optimizando sus hiperparámetros y demás), rápidamente se lo descartó para esto al considerar que sus resultados solo podían servir para sentar un piso para el resto del algoritmos por probar. Quedó entonces relegado a la función de evaluar cualquier cambio que se quisiera introducir al set de datos o al

conjunto de features a utilizar. Esto no es casualidad: La capacidad de Random Forest para evaluar la importancia de los features es conocida y, además, el hecho de que sea un algoritmo basado en árboles que permita plantar una semilla y repetir resultados hace que sea muy útil para comparar entre antes y después de los cambios.
Si bien es cierto que lo que suceda en este algoritmo no necesariamente deberá cumplirse para el resto, también es cierto que es una manera de dar la pauta. La idea, más que venerar aquello que ha funcionado para Ramdom Forest y adoptarlo como regla de oro para el resto de las cosas, es descartar aquello que muestra en Random Forest un rendimiento claramente peor al resto. Es así como se fue llegando a la conclusión de utilizar un set de datos extendido y balanceado, como ya se ha explicado.
Pero su rol principal, sin dudas, está en diagnosticar la importancia de los features. Lo que se hizo, entonces, fue tomar un set de datos que incluyera todos los features mencionados anteriormente y usarlo para entrenar un modelo de Random Forest de parámetros por defecto. El resultado de la importancia de cada feature se muestra a continuación:



Puede verse que existe un 60% de features que resultan poco significativas. A ese porcentaje lo ocupan:

- Encodings
- Location in text
- Keyword in text
- X

Estos resultados son sumamente lógicos. Por un lado, quedan catalogados como poco importantes los features correspondientes al encoding de un atributo como *location* que está plagado de NaNs y de cuyas categorías ya habíamos hablado en el TP1, diciendo que no se notaba una correlación entre los valores que podía tomar y aquellos del target; a pesar de que no tiene tantos NaN, lo mismo aplica para *keyword*. Siguiendo con el análisis, *location in text* y *keyword in text* son atributos que suelen valer cero, independientemente del target. Por último, X es una categoría de PoS que en realidad no es una categoría, porque solo existe para englobar cualquier cosa que no haya podido ser catalogada como ninguno de los otros tipos.

Pero hay que tener cuidado porque esta evaluación solo deja planteada la duda de si los features detectados como poco importantes pueden incluso perjudicar la performance de los algoritmos de machine learning, pero no asegura nada. Al fin y al cabo, los que mandan son los resultados y, hasta no comprobar la hipótesis, hay que tomarla con pinzas.
Las comparaciones en Random Forest y XGBoost dejaron ver que el rendimiento aumenta al quitar los features. En los algoritmos de redes neuronales parece no surtir demasiado efecto pero las comparaciones aquí son más dificultosas, puesto que el resultado del entrenamiento de un algoritmo de este tipo nunca entrega valores iguales.
Algo interesante para señalar es que, usando TF-IDF como método de vectorización en lugar de embeddings, el top 20 de los features más importantes estaba ocupado, casi en su totalidad, por todos aquellos resultantes de la extracción de features del texto. Luego, al cambiar TF-IDF por embeddings, el poderío de estos últimos se hizo evidente, desplazando la mayoría de estos features del top en un duro golpe al orgullo de quien escribe estas líneas. Sin embargo, hay que decir que hubo algunas que lograron mantener su lugar entre las más relevantes, a saber:

- Headlines distance
- Noun Ratio
- Text Trash

Segunda parte

En esta sección se hará un repaso a lo largo de los algoritmos que fueron utilizados en la búsqueda de realizar las mejores predicciones.
Aclaraciones previas:

- Se da por visto el algoritmo de Random Forest a partir del testimonio en “Feature Importance”.
- Es imposible dejar registro en un informe como este de todas las alternativas que se han propuesto, probado, verificado o descartado para cada uno de los algoritmos que se verán a continuación. Sin embargo, sí se repasaran las variables sobre las que se puso el foco y se repasará el resultado general.

CatBoost

Este algoritmo de boosting es reconocido por entregar buenos resultados sin tener que tunear sus hiperparámetros, es decir, usando sus parámetros por defecto. Por lo tanto, es normal que se lo use para fijar un límite inferior que perita detectar a un algoritmo malo (o mal tuneado) si es que sus predicciones están en un nivel por debajo de este límite.
Correr este algoritmo arroja niveles de accuracy similar a Random Forest, lo que implica que ronda los 0.77 puntos, con la particularidad de que es bastante malo prediciendo aquellos datos cuyo target vale 1. Esto es algo que vale para todos los algoritmos probados, pero en este caso era muy notorio y el F1-Score de la clase 1 apenas superaba el 0.70.

XGBoost

Este fue el primer algoritmo que se probó esperando buenos resultados, en gran medida por su buena fama como clasificador. Todos los set de datos, variantes en el uso de features y métodos de vectorización del texto descritos en este informe fueron probados en este algoritmo, en primer medida como una especie de comprobación de los diagnósticos de Random Forest y luego con mucho énfasis en las configuraciones "exitosas" del set de datos.
Si bien los resultados mostraron mejoras notables al introducir embeddings, fue invertida una considerable cantidad de tiempo en optimizar mediante Grid Search los hiperparámetros de un Pipeline que incluía TF-IDF + SVD + XGBClassifier. Los hiperparámetros sobre los que se puso el foco fueron:

- n_components para SVD
- n_estimators, max_depth, learning_rate, colsample_bytree, scale_pos_weight para XGBClassifier

Los mejores resultados para este tipo de vectorización se alcanzaron con parámetros como n_components=150, que resulta una reducción importante considerando que el vocabulario original tenía unas 2000 componentes, y scale_pos_weight=1.5, que también resulta interesante considerando que se usó un set de datos balanceado y que, aun así, al algoritmo le cuesta más clasificar correctamente los *target 1* que los *target 0*. Como era de esperar, el set extendido permite el uso de más cantidad de estimadores, una mayor profundidad y un mayor porcentaje de samples por árbol. Los valores de precisión resultantes rondan los 0.76 puntos.
Recién al introducir el uso de embeddings es que XGBoost se despega de Random Forest y lo supera, llegando a superar los 0.80 en accuracy y 0.79 en Macro Averaged F1-Score. La diferencia entre las métricas es porque el F1-Score refleja la dificultad de clasificación de una clase respecto a la otra. Y cabe destacar que estos valores se encontraban bastante más alejados antes de usar un set balanceado.

Redes neuronales

En este caso se realizaron un sinnúmero de pruebas, puesto que cada combinación de capas es un modelo nuevo. Las capas que se utilizaron a lo largo de estos modelos fueron:

- Capa densa
- Capa de Conv1D
- Capa de LSTM
- Capa de Bi-LSTM
- Distintas capas de pooling: Max Pooling, Average Pooling, Global Max Pooling, Global Average Pooling
- Capa de embedding
- Capa de BatchNormalization

Algo que tienen en común todos estos modelos es que ambos requieren de dos entradas: Una para el texto pre-procesado y otra para los features numéricos. Esto tiene su razón de ser en el hecho de que se use una capa de embedding que solo procesa datos que tengan la forma de un vector representativo del texto en relación con el vocabulario que puedan ser transformados en una secuencia de embeddings por dicha capa. La secuencia queda formada entonces por un

arreglo de embeddings en donde cada embedding es una palabra de del texto que se ha transformado. Esta capa permanece inmutable durante el entrenamiento y sus pesos deben ser configurados a priori en base a una matriz cuyas filas corresponden al embedding de cada una de las palabras del diccionario.
Otro punto que hay que mencionar es la diferencia en el procesamiento de los features numéricos respecto a lo que ocurría cuando se hacía uso de algoritmos basados en árboles. En ese caso, no era necesario normalizar los features pero ahora sí, porque escalas más grandes en el rango de valores que puede tomar un feature eleva la preponderancia con que las redes neuronales lo consideran. Por lo tanto, es necesario centrar los features alrededor de cero y mantener su variación entre [-1,1].
Una tendencia que se observó en todos los modelos de redes neuronales y que los diferencian del tratamiento que se hizo con algoritmos abordados previamente es que modificar el peso por defecto de las clases fue absolutamente contraproducente. En lugar de mejorar las predicciones de la clase a la que se le aumentaba el peso, empeoraron aquellas de la otra clase.
Hecha estas aclaraciones, pasemos ahora a ver las generalidades de los modelos que pueden ser construidos al combinar estas capas.

1) Perceptrón multicapa: De todos, el modelo más sencillo. La salida de la embedding layer pasa directamente a una capa de Global Average Pooling o Global Max Pooling que reduce la secuencia de embeddings a un único vector. Este vector se concatena con el segundo input (de los features numéricos) y pasan a un conjunto de capas densas que terminan en la salida del modelo.
Los parámetros a tunear, más allá de aquellos correspondientes al compilador, son la cantidad de capas a utilizar, la cantidad de neuronas de cada capa y la función de activación. El problema de clasificación binaria hace que, obligadamente, la última capa antes de la salida sea de una única neurona y es típico el uso de la sigmoide como función de activación. Sobre el resto de las capas no hay limitaciones aunque es común que la función de activación sea la ReLU.
Lo que se observa es que el modelo alcanza una clara meseta. En un principio, los resultados parecen mejorar con más capas y neuronas. Al ser las únicas capas del modelo, se pueden agregar varias en combinación con capas de dropout y evitar que el modelo overfittee. Pero viendo las curvas de accuracy, se nota un límite cada vez más marcado en forma de una clara asíntota que, en este caso, rondaba los 0.78 puntos.
Una diferencia interesante se notaba al jugar con los inputs del modelo. El valor de la asíntota era prácticamente el mismo usando únicamente el primer input o combinándolo con el segundo. Pero mientras la primera opción generaba curvas de accuracy y loss bastante erráticas y dentadas, la segunda opción mostraba curvas mucho más suaves.

2) Redes convolucionales: Las capas de Conv1D se usan para que el modelo aprenda features a partir de la salida de la capa de embedding. La idea entonces as agregar una combinación de estas con capas de pooling entre dicha capa y la concatenación con el segundo input. Las combinaciones son infinitas pero una sobre las que más se trabajó fue la de Conv1D > Conv1D > MaxPooling > Conv1D > GlobalMaxPooling. Lo interesante es que una búsqueda de hiperparámetros a partir de Bayesian Optimization llegó a que la mejor opción incluía tamaños de kernel escalonados entre las tres capaz de convolución y tamaños de filtro que eran inferiores a la dimensión de los embeddings.
Los valores de accuracy que entregó este modelo iban entre los 0.79 y 0.80.

3) LSTMs: Se las ubica de manera similar a las capas de convolución, para tomar features del texto. Son capas más complejas que van guardando registro de lo visto. Esto hace que se pueda jugar con su salida haciendo que devuelva el último registro o la secuencia completa. En base a esto, se

probó usar este último registro, la secuencia completa afectada de un GlobalMaxPooling, la secuencia completa afectada de un GlobalAveragePooling o una concatenación de un par de ellas. Si la única capa que afectaba al input1 era la LSTM, la opción de concatenar dos resultados producía una buena mejora a un resultado que, de por sí, era pobre, permitiendo pasar de unos 0.75 puntos de accuracy a unos 0.78. Sin embargo, el agregado de alguna otra capa que complejizara el modelo hacía que esta posibilidad ya no fuera posible y que se produjera lo que parecía ser el desvanecimiento del gradiente.
Otra variante sobre la que se trabajó fue al de anteponer una combinación de Conv1D+MaxPooling a la LSTM. Los resultados mejoran un poco en este caso pero lo mejor es el aumento en la velocidad de entrenamiento al reducir el input de la LSTM con el MaxPooling (¡Las LSTM son demasiado lentas!)

4) Bi-LSTM: En este caso no se probaron tantas variantes como con otras capas bajo el temor de que agregar más capas al sistema teniendo una tan compleja como la Bi-LSTM llevara rápidamente a overfitting. En un modelo conformado a partir de reemplazar todas las capas convolucionales y capas de pooling por una Bi-LSTM, se llegó a resultados similares que en el modelo original, con una inversión de tiempo mayor para el entrenamiento.

Attention models

Este apartado existe únicamente por creer que hay que saber aceptar la derrota.
Se hizo un intento por implementar un modelo con una BERT Layer a partir del desarrollo pre-entrenado que presenta TensorFlow Hub. La capa es tal que contiene más de 100.000.000 pesos distintos que sería imposible re-entrenar con una capacidad computacional normal en un tiempo aceptable.
Un modelo conformado a partir de esta capa freezada y algunas capas densas posteriores no sirve para mucho. Lo que se intentó, entonces, fue armar un modelo en donde se use esta capa con casi todos sus pesos freezados pero no todos. No hubo éxito.

El mejor modelo

El mejor modelo surgió a partir de una de las primeras pruebas con redes neuronales y, más específicamente hablando, con Percherón Multicapa.
En ese momento aún no se había desarrollado la embeddings matrix que fue explicada en el apartado anterior como algo que tenían en común todos los modelos de redes neuronales que se habían probado. Pero este no. En este caso los embeddings entran listos para caer a las capas densas, porque ya se encuentran pre-procesados en el sentido de que se ha transformado la frase en el promedio de los embeddings que la componen Entonces, la entrada al modelo no es más que la concatenación de este vector y aquel de los features numéricos correspondiente al mismo dato, tal cual el caso de las entradas de Random Forest o XGBoost.
Es de suponer que lo que se acaba de describir debería funcionar igual que la combinación de embedding layer y Global Average Pooling que se comentó en el apartado de redes neuronales, porque son esencialmente lo mismo. Pero la realidad es que este caso está varios puntos por arriba de los resultados de ese modelo, llegando a unos 0.81 de accuracy y F1-Score. Quizás sea porque la normalización ahora incluye ambas entradas antes de entrar al modelo, pero lo cierto es

que en cualquier de los modelos previamente mencionados, era infaltable la normalización de la salida de la embedding layer.