UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

75.06 Organización de Datos

Trabajo Práctico 2
Primer Cuatrimestre de 2020

Grupo 10: The Datalorian

Bobadilla Catalan, German	90123
Calvani, Sergio Alejandro	98588
Valdivia, Josue Giovanni	93075

Link de GitHub:
https://github.com/bobadillagerman/75.06-Datos-TP2-2020

Índice

1. Introducción

El objetivo del presente informe es la de explicar que criterios se utilizaron para la creación del modelo de datos a utilizar con Machine Learning y así mismo, también los algoritmos utilizados. Ahora bien, el mismo va a estar dividido en tres secciones:

- ***Procesamiento de Datos:*** Se anunciará todo lo que se hizo en el moldeado de los datos desde el pre-procesamiento, hasta los features que se utilizaron, el cómo y el porqué.
- ***Algoritmos:*** Explicación de los algoritmos utilizados, el porqué se utilizaron, los distintos hiper parámetros y un pequeño análisis de sus resultados.
- ***Conclusión Final:*** Para finalizar, una última conclusión en la que dirá cuál fue el algoritmo que mejor resultado nos dió,cual fue el mejor resultado y porque creemos que ese algoritmo en particular fue el que mejor se moldeó a nuestros set.

2. Procesamiento de Datos

Esta es la parte más extensa de todo proceso de Machine Learning y así mismo, donde surge toda la magia. También es un constante prueba y error, en el cual hay que ir validando constantemente los impactos que generan los nuevos features, ya sean de manera positiva como negativa. Por lo tanto, lo que se explicará a lo largo de toda esta sección no significa que fue utilizada en el set de entrenamiento final, la forma final del mismo va a ser explicada en un apartado al final de este capítulo. Como última aclaración, todo lo que se probó fue en base al Análisis Exploratorio realizado en el Trabajo Práctico Nro 1.

2.1. Datos Utilizados

Comenzaremos explicando los datos que fueron utilizados para este trabajo, se explicara el propósito de dichos datos y cómo fueron obtenidos de ser el caso.

2.1.1. Set de Entrenamiento

	id	keyword	location	text	target
0	1	NaN	NaN	our deeds are the reason of this #earthquake m...	1
1	4	NaN	NaN	forest fire near la ronge sask. canada	1
2	5	NaN	NaN	all residents asked to 'shelter in place' are ...	1
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1
4	7	NaN	NaN	just got sent this photo from ruby #alaska as ...	1

Figura 1.1: Set de Entrenamiento

En este caso no hay mucho que decir, es el csv que se obtiene de la competencia con las columnas que fueron explicadas en el informe del Trabajo Práctico Nro 1. La mayor parte del procesamiento de datos está relacionado con este DataFrame.

2.1.2. Ciudades

	city	country
0	Tokyo	Japan
1	New York	United States
2	Mexico City	Mexico
3	Mumbai	India
4	São Paulo	Brazil

Figura 1.2: Dataframe de Ciudades

Csv donde vienen las ciudades del mundo relacionadas con su país, esto nos va a servir para cuando empecemos a procesar la columna de 'location' del set de entrenamiento.

2.1.3. Mundo

	name	iso_a3
0	Fiji	FJI
1	Tanzania	TZA
2	W. Sahara	ESH
3	Canada	CAN
4	United States	USA

Figura 1.3: Dataframe de Países del Mundo

Dataframe que tiene el nombre de todos los países del mundo con su abreviación en tres letras, así como sucede con el anterior, este también es utilizado con la columna de 'location' del set de entrenamiento. Este csv fue obtenido de la biblioteca geopandas y extraído de la misma, para ver cómo se realizó, se puede ver en el Github adjuntado en la carátula, específicamente en la carpeta de extracciones.

2.1.4. Estados de USA

	Nombre_Estado	Abreviacion_Estado
0	Washington	WA
1	Montana	MT
2	Maine	ME
3	North Dakota	ND
4	South Dakota	SD

Figura 1.4: Dataframe de Estados de USA

Dataframe que tiene el nombre de los estados de Estados Unidos con su abreviación en tres dígitos como sucede con el de los países del mundo. En este caso en particular, el Dataframe fue obtenido del siguiente link: https://www.arcgis.com/home/item.html?id=f7f805eb65eb4ab787a0a3e1116ca7e5 y su extracción se encuentra en el mismo lugar que el anterior.

2.1.5. Hashtags

	Unnamed: 0
Hashtag	
News	1
Hot	0
Best	1609
Prebreak	1167
Nowplaying	3

Figura 1.5: Hashtags

Este csv contiene todos los hashtags de los tweets del set de entrenamiento, este mismo fue extraído de la columna ‘text’ del set de entrenamiento y se realizaron diferentes tipos de procesamientos, para ver esto en detalle nuevamente se recomienda revisar la carpeta Extracciones que se encuentran en el Github. Este Dataframe en particular va a utilizarse en el procesamiento de la columna ‘text’. A su vez, está ordenado desde el que más cantidad de veces apareció al que menos.

2.1.6. Mención

	Unnamed: 0
Mencion	
youtube	0
arianagrande	5
potus	6
foxnews	7
change	1

Figura 1.6: Menciones

En este Dataframe se encuentran las cuentas más mencionadas, así como se hizo con los Hashtags, también se obtuvo de la columna 'text' mediante un proceso que se puede ver en detalle en Github en la carpeta mencionada en los apartados anteriores. Así mismo como sucede con los hashtags, también este dataframe va a ser utilizado en el procesamiento de la columna 'text'. Así como sucede con el Dataframe de Hashtag, también se encuentra ordenado desde los más mencionados a los menores.

2.2. Feature Extraction

Ahora bien, en este set de datos tenemos el campo de texto que refiere al tweet en sí, por lo que se puede suponer que es el que major impacto va a tener en nuestro modelo. Para ello probamos Feature Extraction, el cual se encarga de extraer features, en este caso, de los texto utilizando conceptos que hemos visto durante la cursada, en particular fueron utilizados tres:

- ***CountVectorizer:*** Este algoritmo se encarga de convertir un texto en una matriz de contadores de tokens. En nuestro caso utilizamos el hiper parámetro de ***stop_words*** el cual se seteo en ***english*** el cual establece el idioma del cual tomar las palabras , después ***analyzer*** el cual establece cual establece cómo se van a analizar los n-gramas, en este caso utilizamos la función ***build_analyzer*** que viene con la biblioteca y por último el hiper parámetro ***min_df*** el cual, al construir el vocabulario, ignora los rare words, términos que tienen una frecuencia en los documentos menores al valor establecido, en este caso, ***2***.
- ***TF-IDF:*** Este concepto no necesita explicación ya que es fuertemente abarcado en la cursada por lo que se nos ocurrió ver qué pasaba utilizándolo, en este caso se

utilizaron los mismos hiper parámetros que se usaron en el ***CountVectorizer*** pero sin utilizar el ***min_df***.

- ***HashingVectorizer:*** Se encarga de convertir un texto a una matriz de apariciones de los tokens. Los hiper parámetros son similares a los dos anteriores, en este caso solamente se usaron el de ***stop_words***, establecido en ***english*** y ***n_features*** el cual establece la cantidad de columnas de la matriz final, probamos con varios números pero no logramos notar una diferencia significativa.

2.3. Análisis de Texto

Si bien en el apartado anterior se explicó como se analizó el tweet, dado lo que se pudo observar en el análisis exploratorio, se decidió realizar un análisis mucho más minucioso.

2.3.1. Palabras (stems)

Para considerar las palabras más importantes utilizamos la técnica de stemming, que consiste en llevar las palabras a su raíz.

Primero quitamos los conectores y dividimos el texto en dos, por un lado las palabras verídicas y por otro las falsas. Luego en ambos grupos se ordenó los stems por la cantidad de veces que aparecen, quedándonos con los de mayor aparición. Y de ambos grupos quitamos las palabras que pertenecen tanto a los stems verídicos como falsos. Finalmente, con un ciclo, se verifica si tal stem se encuentra en el texto, y de estarlo se le asigna **True**, de lo contrario, **False**.

2.3.2. Adjetivos y Verbos

A su vez se realizó un procedimiento similar al de las palabras, pero en este caso en base a los adjetivos y los verbos, aunque aquí no se los dividió por tomar las n que más cantidad aparecen en los tweets verdaderos o falsos, sino que se tomó los n mayores en general, sin importar el target. Esto mismo se trató de hacer con los sustantivos pero nos generó problemas y así mismo, no generó un impacto muy importante.

2.3.3. Longitud y Cantidad de Palabras

Ya que se hecho el análisis posterior, decidimos tomar en cuenta las longitudes de las palabras y la cantidad de las mismas en los tweets, el problema era generar features en

base a estos valores, ya que, no tendría sentido generar una columna por cada valor, ya que pueden surgir dos malinterpretaciones:

- Primero que se van a generar miles de columnas por la cantidad de palabras y longitud por lo que hay menor posibilidad de coincidencia y serían valores innecesarios.
- Segundo se van a mezclar los valores de ambos, ya que, por ejemplo, que la cantidad de palabras sea 9 no es lo mismo que la longitud también lo sea.

Para solucionar estos dos puntos, se realizó lo siguiente:

- Primero se generaron bins para agrupar asignarle un rango que pertenencia a cada longitud y cantidad de palabras, se tomó bins de cada 10 para ambos.
- Por último, se realizó one hot encoding para estos rangos y se le agregó un un sufijo para diferenciar a ambos.

2.4. Análisis por Locación

Seguimos con el análisis por locación, en este caso vamos a dividirlos en dos partes, uno que es el análisis por país y otro por el estado de Estados Unidos.

2.4.1. Por País

En este caso se utilizó el csv que fue mencionado en el 2.1.3 y 2.1.2, esto es porque en el Trabajo Práctico Nro 1 notamos que en la locación podía venir tanto el país de origen, o la ciudad, por ende, si queremos utilizar el pais, deberiamos poder calcular el país por la ciudad. Otra cosa que se noto es que, en algunos casos, puede que en la locación se encontraba el país pero era la abreviación del mismo, por ejemplo, en vez de Estados Unidos dice USA. Para este análisis se realizaron tres pasos:

- Se le joineó por el nombre la tabla de *ciudad* para poder mapear el país.
- Se le joineó por la abreviación la tabla de *mundo* para poder mapear el país.
- Por último, se utilizó una función en la cual se verifica cual de los tres valores (la locación que venía en el set, el resultado del mapeo de ciudad y el resultado de mapeo de mundo) no era nulo para que sea la locación final. En el caso de que los tres sean nulos, se le asignó el valor *'Sin Locación'.*

2.4.2. Por Estado de USA

Mismo procedimiento que se hizo con el país se hizo con el Estado, obviamente utilizando el csv mencionado en el 2.1.4. Aquí se realizaron nuevamente tres pasos:

- Se le joineó por el nombre la tabla de estados y luego se utilizó una función para verificar que, si la abreviación es nula, significa que en la locación original no estaba ningún estado, por lo que devolvía un nulo, en cambio, de no ser nula la abreviación mapeada, significa que efectivamente había un estado, por lo que se devuelve la locación.
- Se le joineó por la abreviación la tabla de estados para poder mapear el estado.
- Por último, similar al caso anterior, se verifica cual de los valores no es nulo para que sea el estado final. En caso de que sean todos nulos, se le coloca un nulo.

2.5. Tiene Símbolo

En este caso se generaron features en base a si tiene o no unos determinados símbolos, los cuales simbolizan cosas propias de las redes sociales u otras cosas tales como preguntas, aunque no es 100% que la presencia de uno de estos símbolos referencia a lo que efectivamente se quiere, salgo el Hashtag(#), el cuál, en el caso particular de Twitter, siempre que se encuentre un #, todo lo que sigue después se refiere a un Hashtag. Los símbolos que se buscaron son:

- ***@:*** En este caso se estableció que si se tiene este símbolo, tiene una mención.
- ***#:*** Como se mencionó anteriormente, está referenciando a que tiene un hashtag.
- ***? o ¿:*** Se presume que se tiene una pregunta.
- ***! o ¡:*** Se presume que se tiene una exclamación
- ***https:*** Se piensa que se tiene un link a alguna otra página.

En caso de tener alguno de estos símbolos, se le asignó ***True***, sino ***False***.

2.6. Hashtags y Menciones

Este proceso fue similar a lo que se hizo con las palabras, adjetivos y verbos principales, en este caso, como los Dataframes tanto de Hashtags como el de las Menciones venían ordenamos desde el más popular al menos, solamente fue necesario quedarnos con los n hashtags o menciones de la parte superior para quedarnos con los mejores y luego, recorriendo cada uno de los n mejores hashtags o menciones, se verifica que el tweet tenga o # + hashtag o @ + mención, asignando ***True*** en el caso de que esté o ***False*** en el caso que no.

2.7. Nulos

Durante el análisis exploratorio pudimos ver que solamente las columnas ***keyword*** y ***locacion*** son lo que presentan valores nulos, para ello lo que hicimos fue utilizar una función que verifica que si el valor es efectivamente nulo, devuelve ***True***, de lo contrario, ***False***.

2.8. Conclusión General

En este apartado se enunciaron los Dataframes complementarios que fueron utilizados, tanto para el mapeo como para poder establecer si tiene o no un respectivo Hashtag o Mención. Luego también se mencionaron los algoritmos de Feature Extraction que se utilizaron, así también que tipo de análisis se realizó con respecto al texto ya de una manera más específica. Así mismo, también como se calculó el país como el estado de Estados Unidos a partir de la locacion, luego la manera de verificar si tienen distintos símbolos que nos permitian pesar que se referían a una cosa en particular, claro ejemplo el del hashtag con el símbolo #. También cómo se utilizaron tanto los dataframes de Hashtag como el de Meción para poder establecer si el tweet tiene alguno de los principales y por último, el manejo que se hizo de los dos únicos campos nulos. Una vez que se ejecutó todo esto para el set de entrenamiento, se ejecuta todo lo mismo en el set de test.

3. Algoritmos

Una vez realizado todo el procesamiento necesario y habiendo aplicado tanto al set de entrenamiento como al set de test, es momento de aplicar los algoritmos necesarios, en este caso, al ser un problema de predecir si un cierto tweet es real o falso, se deben utilizar algoritmos de clasificación y/o variantes de clasificación de algunos algoritmos, ya que pueden ser tanto de regresión como clasificación.

3.1. Procesamiento Previo

Si bien dijimos que ya tenemos todo lo necesario para poder aplicar los algoritmos, esto no es tan así ya que se pueden realizar dos cosas que nos permiten verificar que tal está funcionando el algoritmo mediante un set de verificación y también se pueden normalizar los datos.

3.1.1. Verificación Local

Se utiliza la función ***train_test_split*** la cual permite dividir el set de entrenamiento en dos, una parte que sirve para entrenar el algoritmo en sí y luego otra que sirve para verificar de manera local, este set se suele llamar de verificación. Esta distribución se puede setear con el hiper parámetro ***test_size***, se probaron varios valores pero el que mejor resultado nos dio fue el valor ***0.33***, el cual se refiere a que un 33% del set de entrenamiento es el set de verificación mientras que el 67% restante es el que entrena al algoritmo, siendo el set de entrenamiento real.

3.1.2. Estandarización de Datos

En este caso se usó la función ***StandardScaler*** el cual estandariza los features removiendo la media y escalando a la varianza de la unidad. Esta estandarización nos permitió en algunos algoritmos (principalmente los lineales) alcanzar mejores resultados pero, sinceramente, la diferencia era mínima.

3.2. Perceptrón

Este es el algoritmo lineal por excelencia, sumado a que, si bien funciona para clasificar varias clases, es utilizado principalmente para clasificación binaria, por lo cual nos pareció interesante probarlo. Perceptrón lo que hace es buscar un hiperplano que separe a las clases de la mejor manera posible, para ello utiliza una función de activación llamada w. Ahora, los hiper parámetros utilizados y/o probados fueron:

- ***penalty:*** La penalización que utiliza, en nuestro caso probamos las tres opciones disponibles, siendo ***l1*** la que mejores resultados nos dieron.
- ***eta0:*** Es la constante por la cual se multiplican las actualizaciones, nuevamente como el caso anterior, se probaron varios valores siendo ***0.1*** el de mejor.
- ***suffle:*** Blooleano que establece si se hace un suffle después de un tiempo, este valor fue seteado tanto en ***False*** como en ***True***, pero no tuvo ningún tipo de impacto.
- ***random_state:*** Se utiliza para hacer un shuffle del set de entrenamiento, este valor tiene importancia si ***suffle*** es ***True***. Probando varios valores, el que mejor resultado dió fue ***0***.

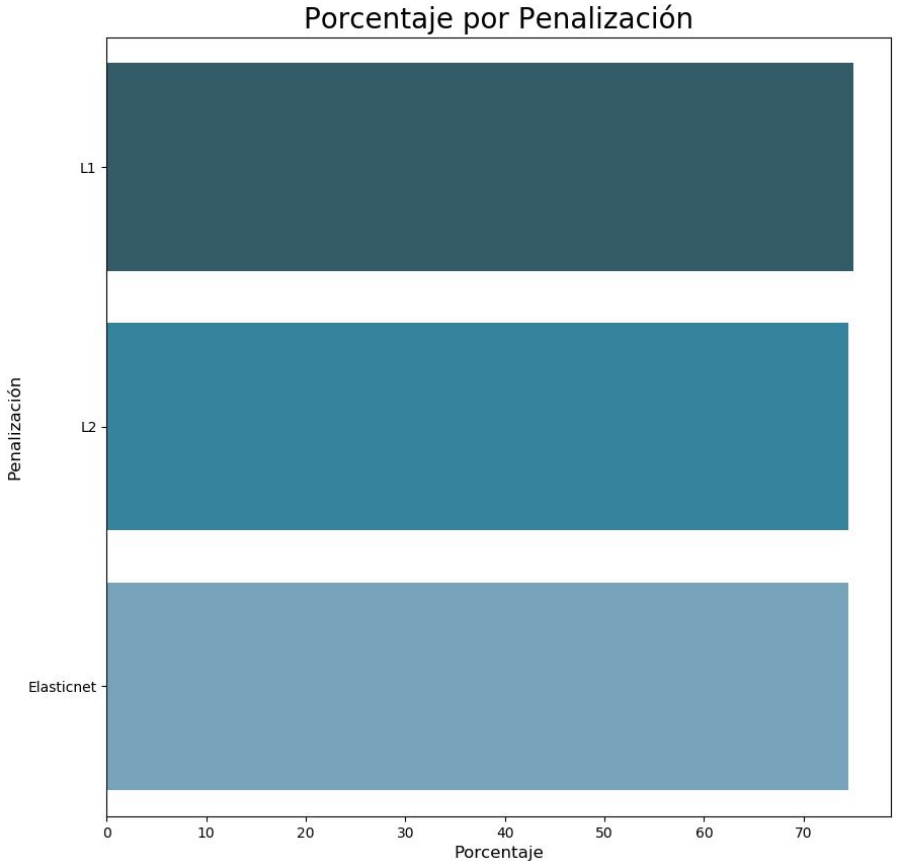

Figura 2.1: Porcentaje por Penalización

Viendo el gráfico podemos confirmar que ***l1*** es la penalización que mejor resultado nos da, con un 75.05%, mientras que tanto ***l2*** como ***eslasticnet*** nos da 74.45%. Este algoritmo particularmente nos dio valores similares a este último número, siempre variando entre el 73% y el 75%, lo cual no es ni cerca de nuestro mejor resultado.

3.3. SVM

Habiendo probado Perceptrón, no podíamos haber ignorado SVM, ya que, si Perceptrón busca un hiperplano separador, SVM encuentra el mejor, por lo que nos pareció interesante ver cuán diferente era a este. Los hiper parámetros utilizados fueron:

- ***penalty:*** Igual al explicado en Perceptrón, para tener una consistencia y que la diferenciación con este sea más correcta, se utilizó el mismo valor, ***l2***.
- ***C:*** Parámetro de regularización, es la que controla la penalización de clasificar mal, este valor es muy sensible ya que si es muy grande hace que el modelo sobre-ajuste mientras que si es muy chico no generaliza bien, dado esto, el valor óptimo que encontramos fue ***1.2***.
- ***random_state:*** Igual el de Perceptrón, nuevamente le colocamos el mismo valor que en este, siendo ***0***.
- ***max_iter:*** Es el máximo número de iteraciones, se puede ver en el siguiente gráfico el que mejor resultado dió fue 100

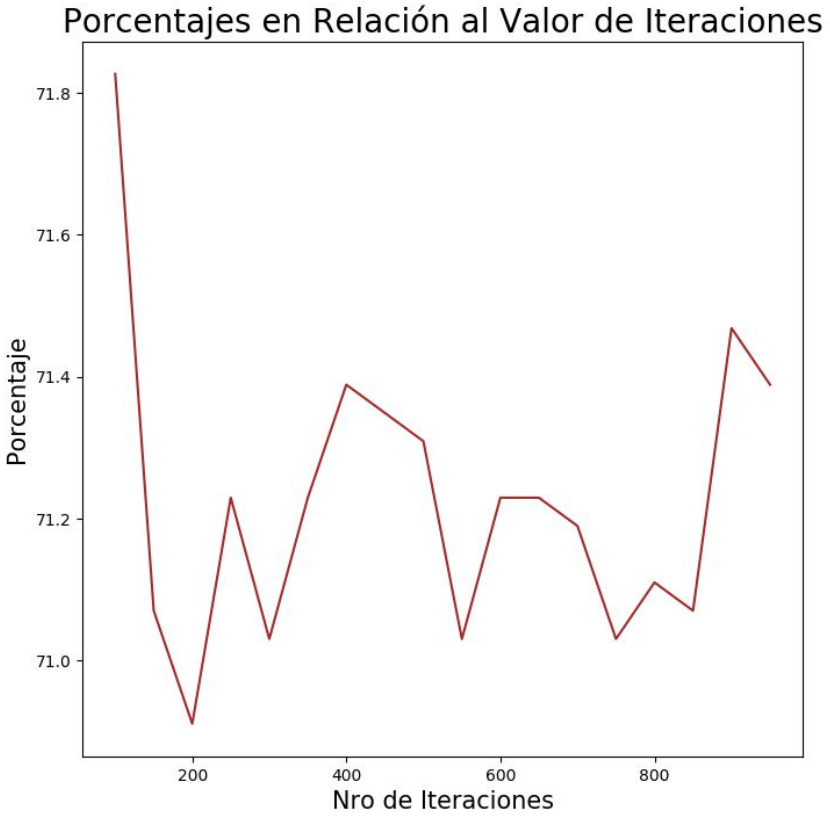

Figura 2.2: Porcentaje en Relación al Valor de Iteraciones

Extrañamente este algoritmo dio peores resultados que Perceptrón, teniendo un 71.83% con el mejor set de entrenamiento que utilizamos, en el promedio general, pudimos observar que variaba mucho este resultado con las distintas pruebas que hicimos, llegando a tener el porcentaje más bajo de validación.

3.4. Logistic Regression

Otro algoritmo de clasificación lineal es Logistic Regession, el cual utiliza una función logarítmica para entrenar el modelo, el cual recibe los siguiente hiper parámetros

- ***penalty:*** Igual a los descritos anteriormente, en este caso, el de mejor resultado fue ***l1***.
- ***solver:*** Es el algoritmo utilizado en el problema de optimización, después de probar distintas variaciones, nos quedamos con ***saga***.
- ***class_weight:*** Peso asociado a las clases, dado que el dict nos daba muchos inconvenientes con los otros hiper parámetros, se decidió dejarlo en ***balanced***.

A continuación, un gráfico que muestra los resultados de las diferentes combinaciones

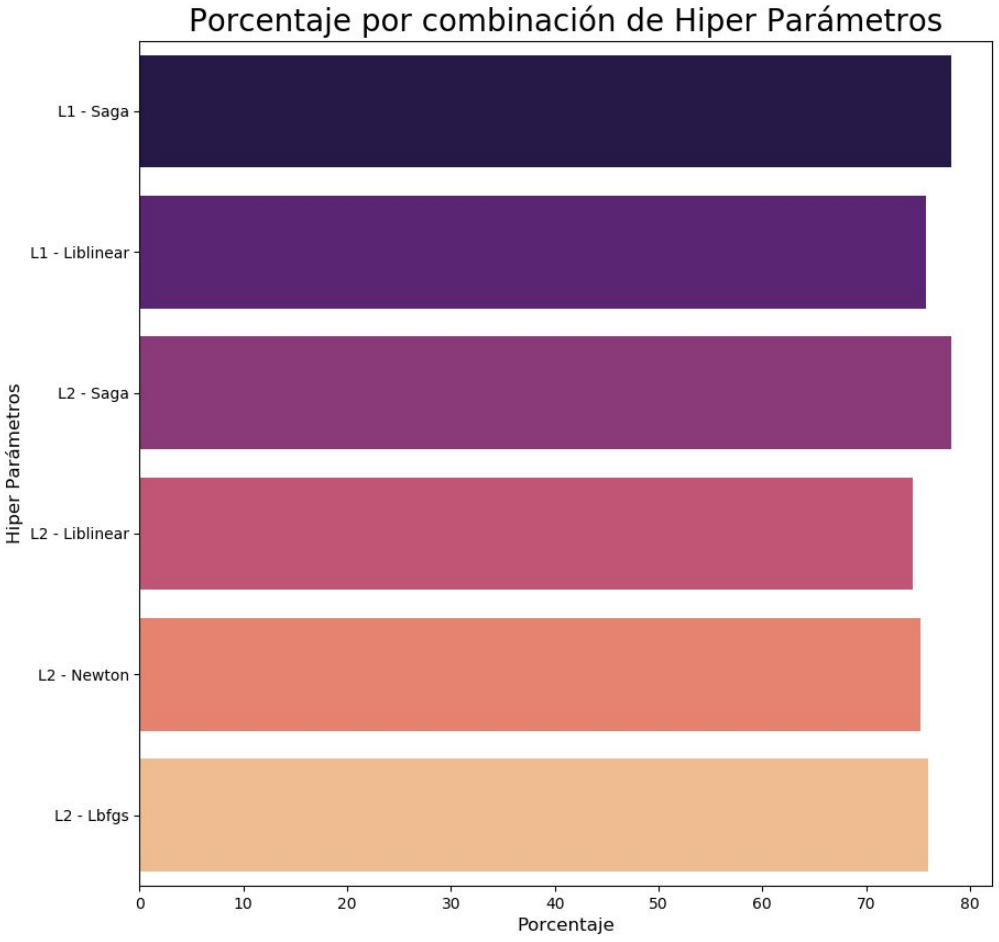

Figura 2.3: Porcentaje por Combinación de Hiper Parámetros

Como puede verse, la diferencia no es realmente tan grande, pero puede observarse que la combinación de l1 y saga es la que nos dió el mejor porcentaje, de 78.27%, bastante mejor comparado con los otros dos algoritmos de clasificación lineal.

3.5. Multi-Layer Perceptron

Probado todos estos anteriores se nos ocurrió utilizar una variación de Perceptrón, llamada Multi-Layer Perceptron, la cual se encarga de resolver problemas que no son lineales, una limitación del Perceptrón original. Para ello, se utilizaron los siguiente hiper parámetros:

- ***solver:*** Igual al explicado en ***Logistic Regression***, en este caso, luego de haber probado todas las variaciones, se utilizó ***sdg***.
- ***activation:*** Función de activación a usar, en este caso, el mejor resultado fue con ***logistic***.
- ***random_state:*** Igual a los anteriores siendo ***0*** el que mejor resultado dió.
- ***max_iter:*** Mismo que en ***SVM***, el mejor valor fue ***100***.

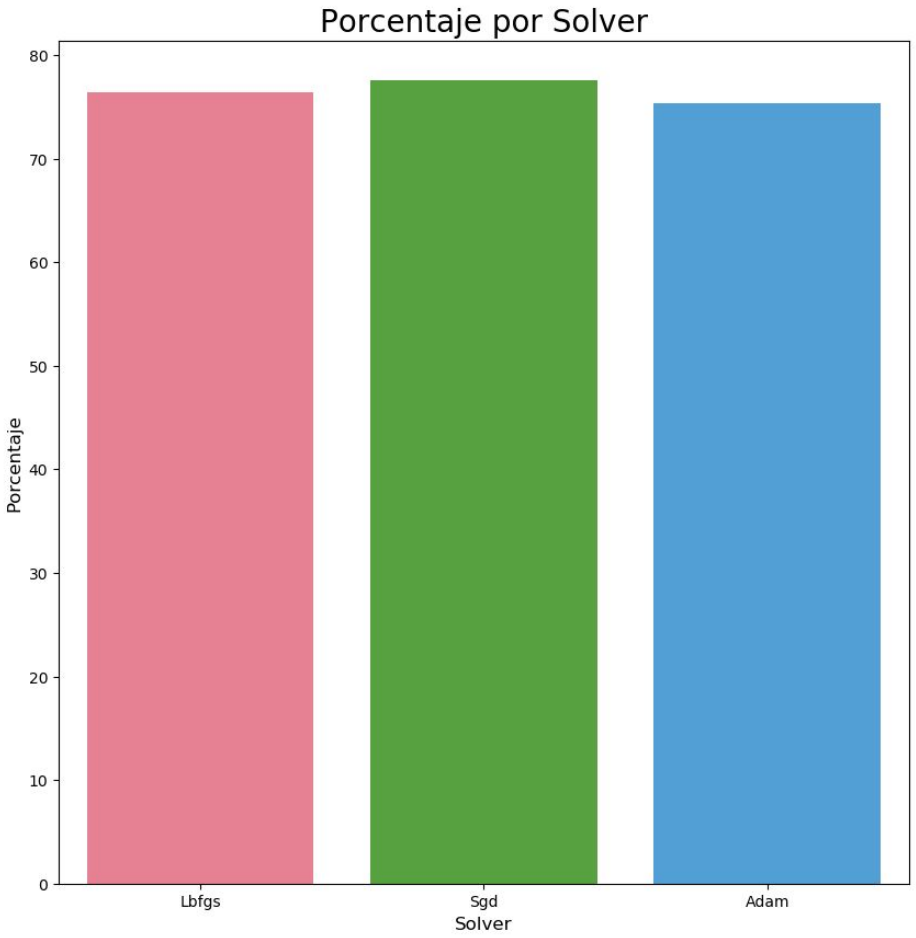

Figura 2.4: Porcentaje por Solver

Puede verse a simple vista que el solver ***sdg*** fue el que mejor resultado dio, con un porcentaje del 77.52%, aunque la diferencia con los otros valores posibles fue baja. A continuación veremos las diferentes variaciones de ***activation***, todas con ***sdg*** como ***solver***.

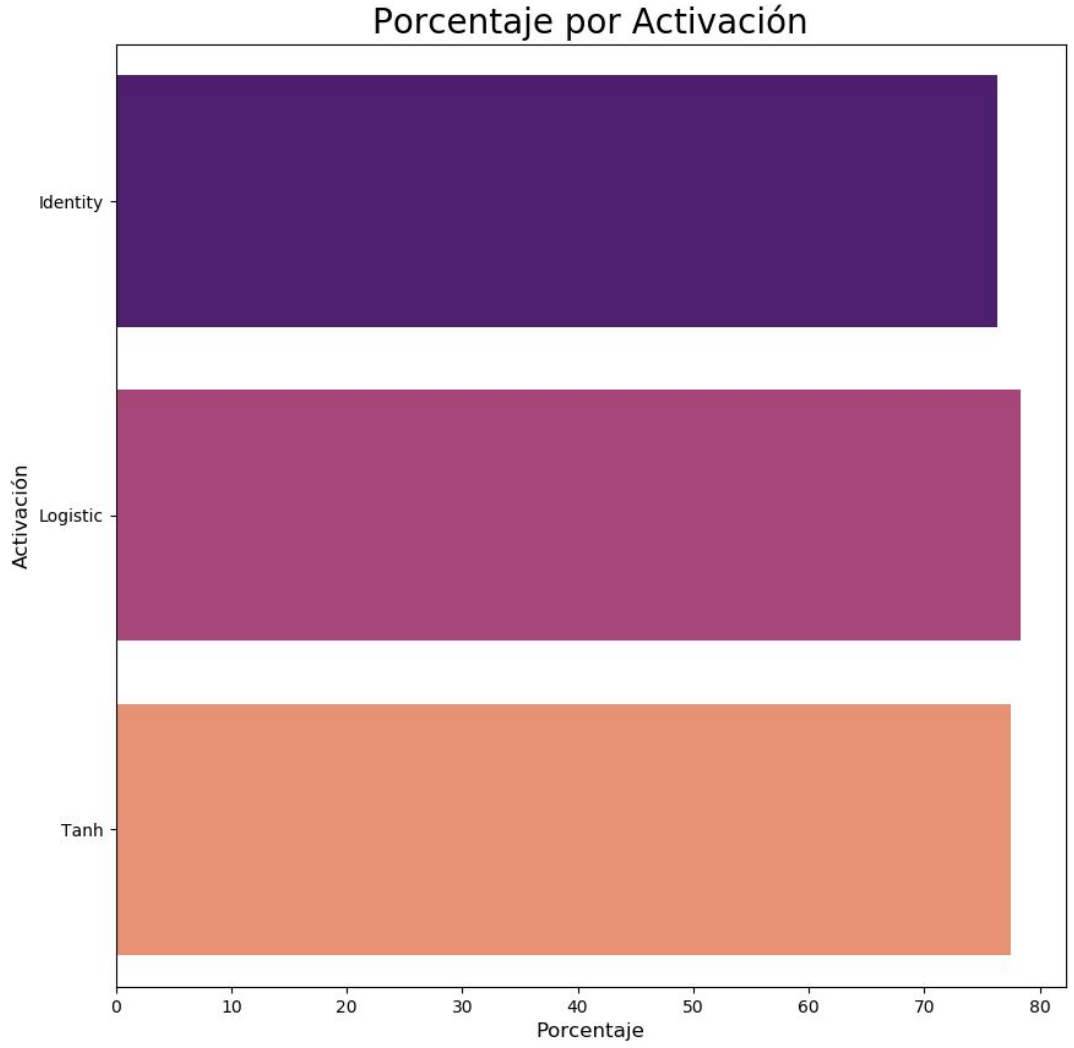

Figura 2.5: Porcentaje por Activación

En este caso ***logistic*** fue el que mejor resultado nos dió con un porcentaje del 78.31%, con una diferencia del 0.79% con el default.

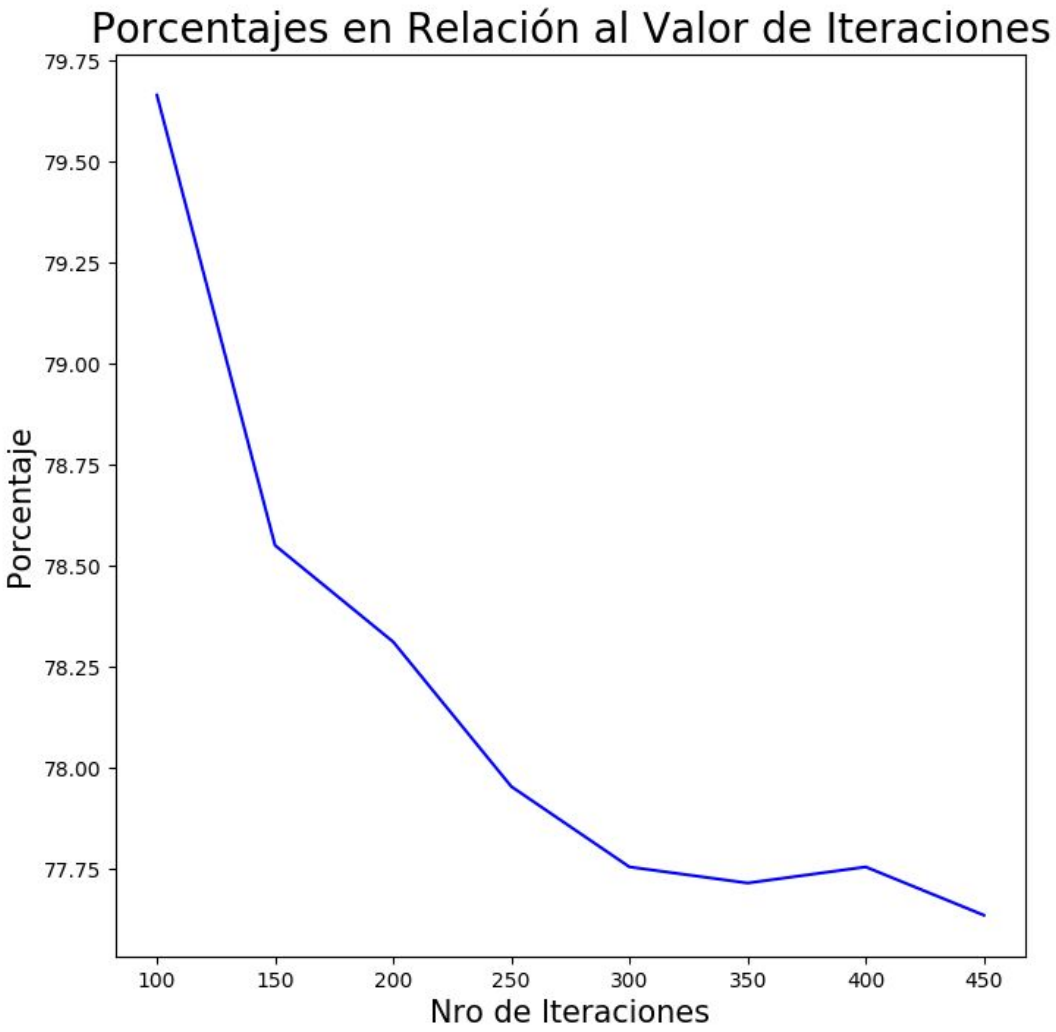

Figura 2.6: Porcentajes en Relación al Valor de Iteraciones

Rápidamente se observa que cuanto menos es la cantidad de estimadores, mayor es el porcentaje, siendo ***100*** el que nos da el mejor resultado, con un porcentaje del 79.67%.

3.6. KNN

Ahora vamos a hacer un cambio radical a lo que estuvimos viendo, esto es utilizar KNN (K-Nearest-Keighbors) , algoritmo el cual clasifica en base a los K vecinos más cercanos. En este caso los hiper parámetros son:

- ***n_keighbors:*** Cantidad de vecinos cercanos a tomar, en este caso, hicimos varias iteraciones pero el que mejor resultado nos dió fue ***1***, aunque con esto nos dimos cuenta de que este algoritmo no funciona de buena manera para este set en

particular, ya que, cuanto más pequeño es el valor, más probabilidad hay de que haya overfitting.

- ***algorithm:*** Algoritmo que calcula los vecinos más cercanos, en este caso,de las tres posibilidades posibles, el que mejor resultado nos dió fue ***kd_tree***.
- ***metric:*** Es la métrica utilizada en el árbol, en este caso nos quedamos con ***euclidean***.
- ***n_jobs:*** Consiste en la cantidad de procesadores a utilizar, en este caso fue puesto en ***-1*** para que utilizara todos los disponibles.

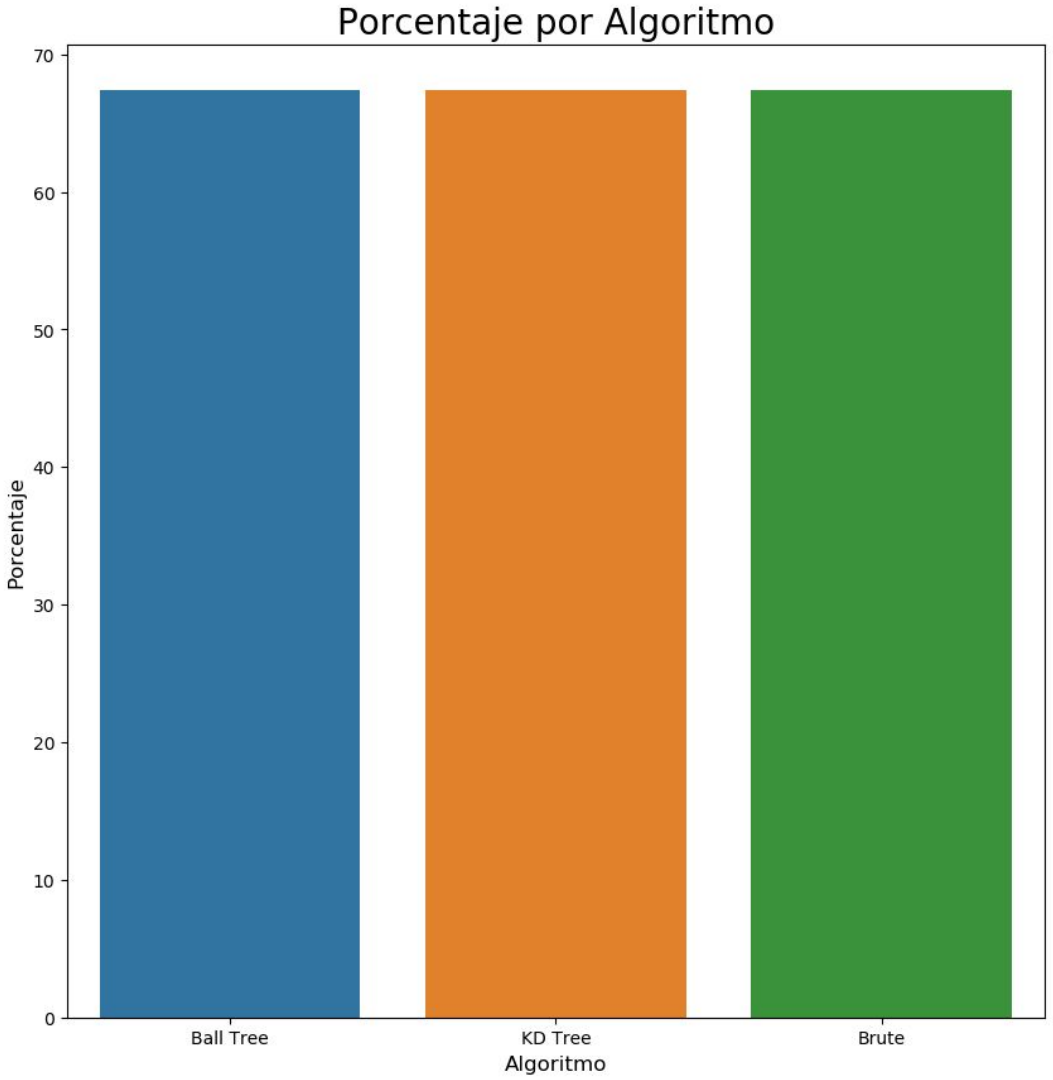

Figura 2.7: Porcentaje por Algoritmo

En cuanto al algoritmo, no hay ninguna diferencia entre cual se quiera utilizar, ya que todos nos dió el mismo valor, por lo que en este caso, utilizamos el ***kd_tree***

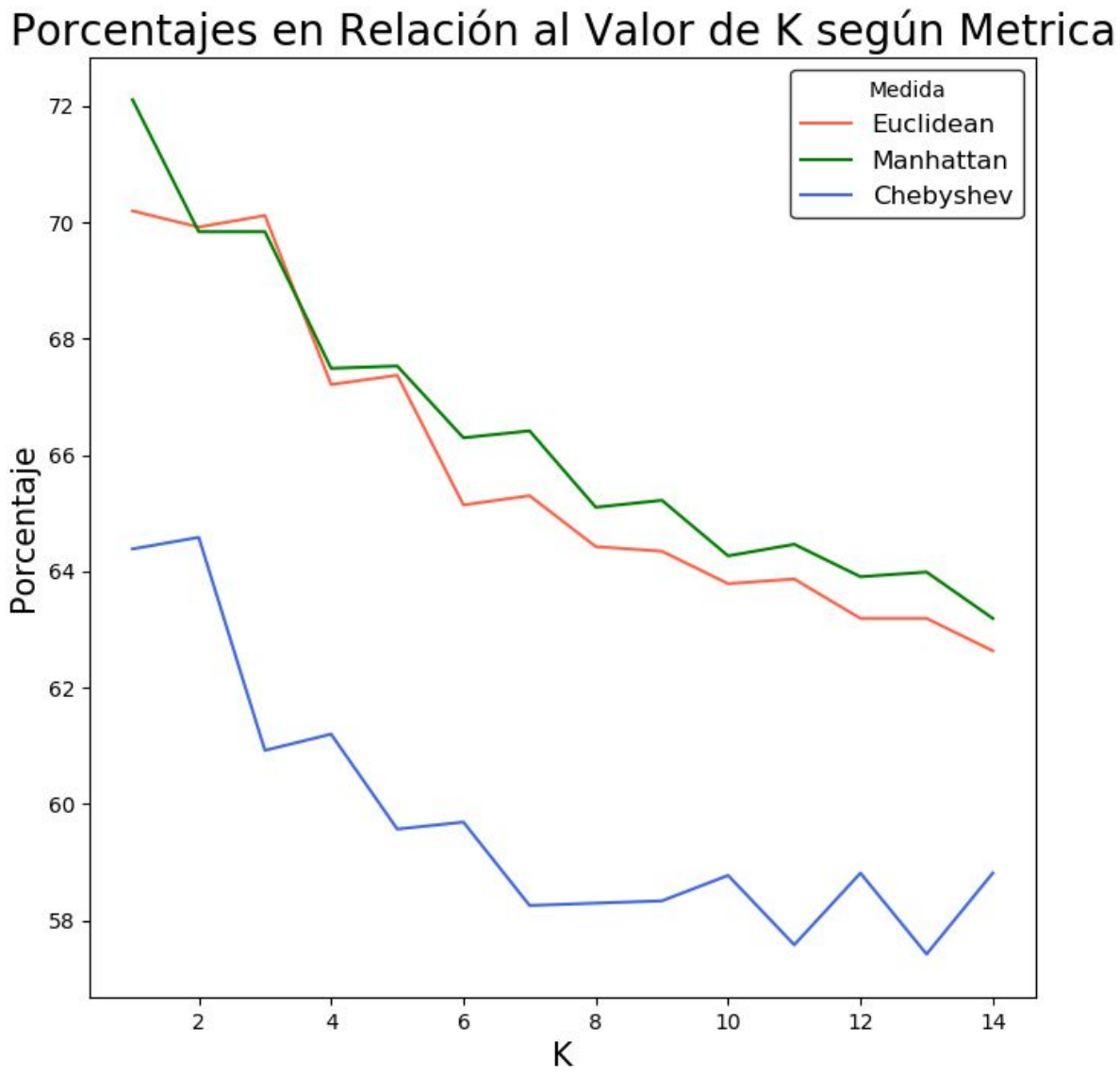

Figura 2.7: Porcentaje en Relación al valor de K según Métrica

Se puede observar que para todas las métricas, a medida que se toman mayor cantidad de vecinos, el porcentaje disminuye. Luego que las métricas ***euclidean*** y ***manhattan*** dieron resultados similares, siendo la primera la de mejor resultado, con un porcentaje 72.11%, muy lejos de los otros algoritmos probados.

3.7. Random Forest

Muy bien, ahora iremos para el terreno de los árboles de decisión, comenzaremos con Random Forest, este algoritmo es uno de los más populares ya que suele generar buenos resultados en la mayor cantidad de los casos debido a que evita caer en el

overfitting mediante el uso de bootstrap, el cual consiste en tomar una muestra del set de entrenamiento pero con reemplazo y también debido a que funciona en base de ensambles. Los hiper parámetros utilizados fueron:

- ***random_state:*** Igual a los que se describieron anteriormente, y por ende, se le asignó también el valor ***0***.
- ***n_jobs:*** Igual al descrito en ***KNN*** y nuevamente, se utilizó el valor ***-1*** para utilizar todos los procesadores disponibles.
- ***n_estimators:*** Es el número de árboles a utilizar, en este caso se hizo un ciclo probando varios valores para ver cual es el mejor, el cual nos dió ***200***.
- ***oob_score:*** Booleano que establece si se usan samples de Bag of Words para estimar la precisión de la generalización. En este caso, lo seteamos en ***True***.
- ***max_features:*** Es el número de features que se consideran al buscar la mejor división, en este caso se tomó el valor de ***log2***.
- ***min_samples_split:*** El mínimo número de samples requeridas para dividir un nodo interno, de varios valores probados, ***100*** fue el de mejor resultado.

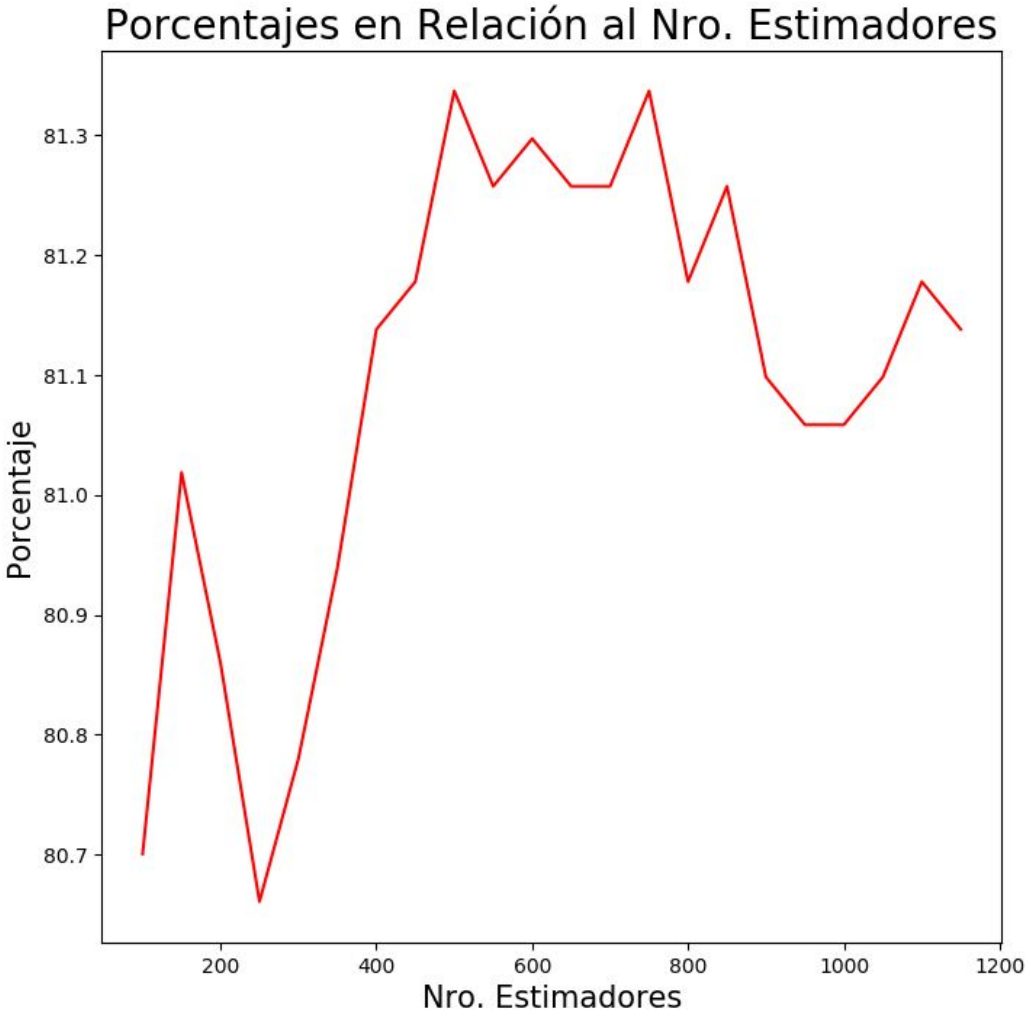

Figura 2.8: Porcentajes en Relación al Nro de Estimadores

Claramente observando el gráfico se puede ver que este algoritmo fue el que venció a todos los anteriores, y, spoiler!, también a los posteriores. En este caso, se ve que con ***500 n_estimators***, nos da un porcentaje del 81.34%, el mayor aunque tanto con el 500 o 650 se llegaron a porcentajes bastante buenos, de un 81.22%.

3.8. Árboles de Decisión

Dado a los grandes resultados que nos dió Random Forest, no perdíamos nada en probar que tal funcionaba con los árboles de decisión, el cual es un árbol binario en donde en cada nodo se divide en base a un cierto criterio. Veamos los hiper parámetros:

- ***splitter:*** Estrategia utilizada para dividir cada nodo. Se probaron las dos posibilidades que hay y ***Random*** fue la que mejor resultado dió.
- ***criterion:*** Es la función que mide la calidad de la división, en este caso, los mejores resultados se dieron con ***gini***.
- ***max_features:*** Igual al explicado en Random Forest, pero en este caso se tomó el valor ***sqrt***.

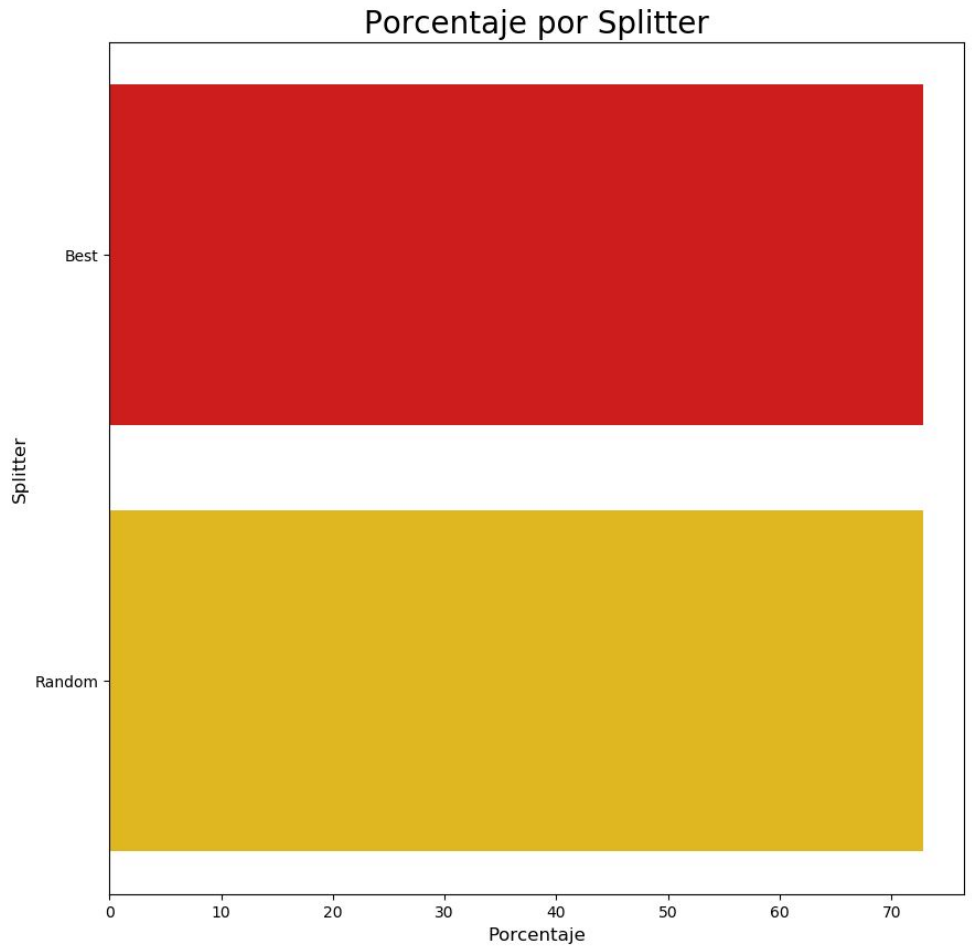

Figura 2.9: Porcentaje por Splitter

La diferencia entre ambas es muy pequeña pero por una diferencia de 0.08%.

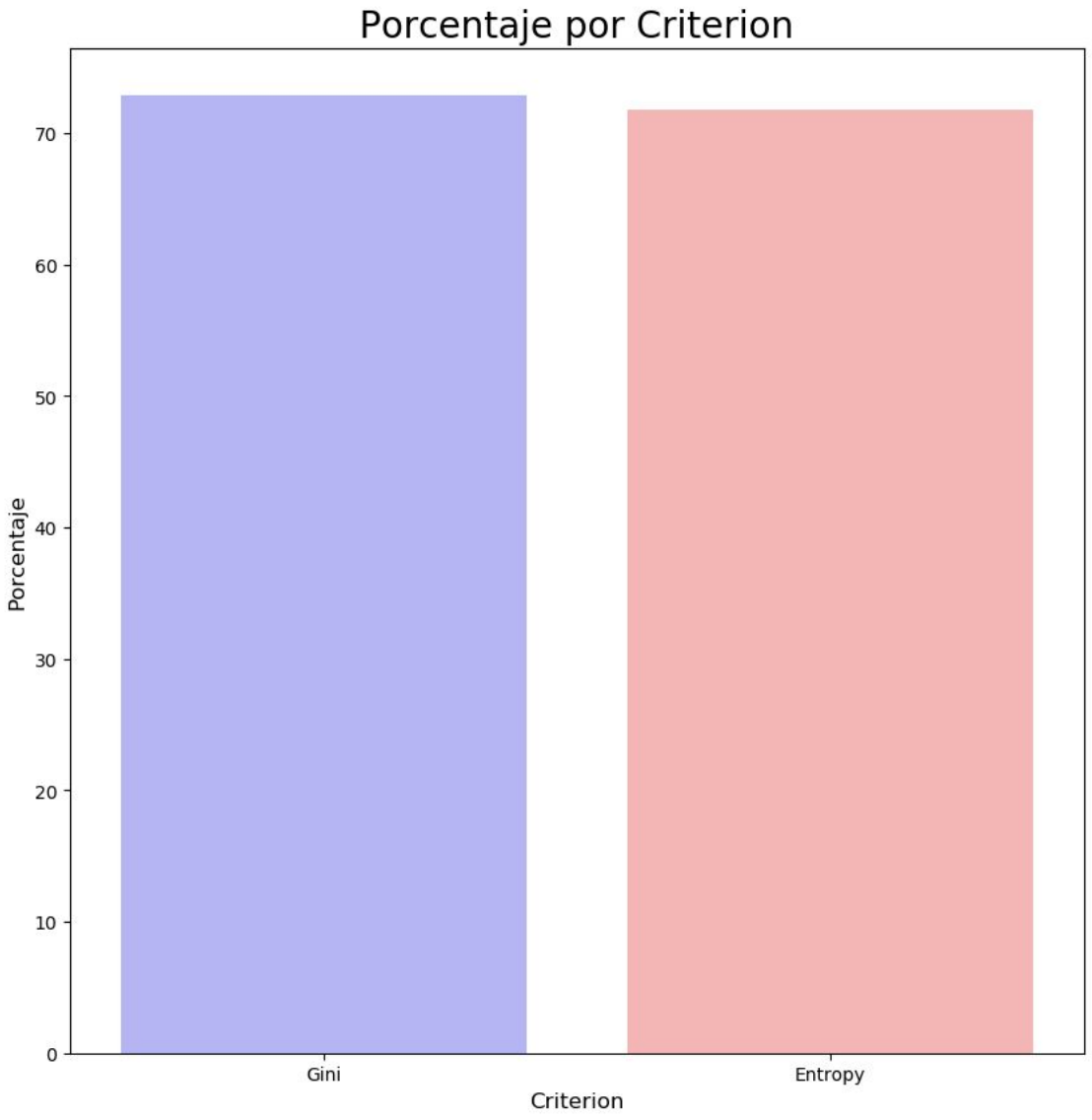

Figura 2.10: Porcentaje por Criterion

En este caso la diferencia es un poco mayor, un poco más del 1%.

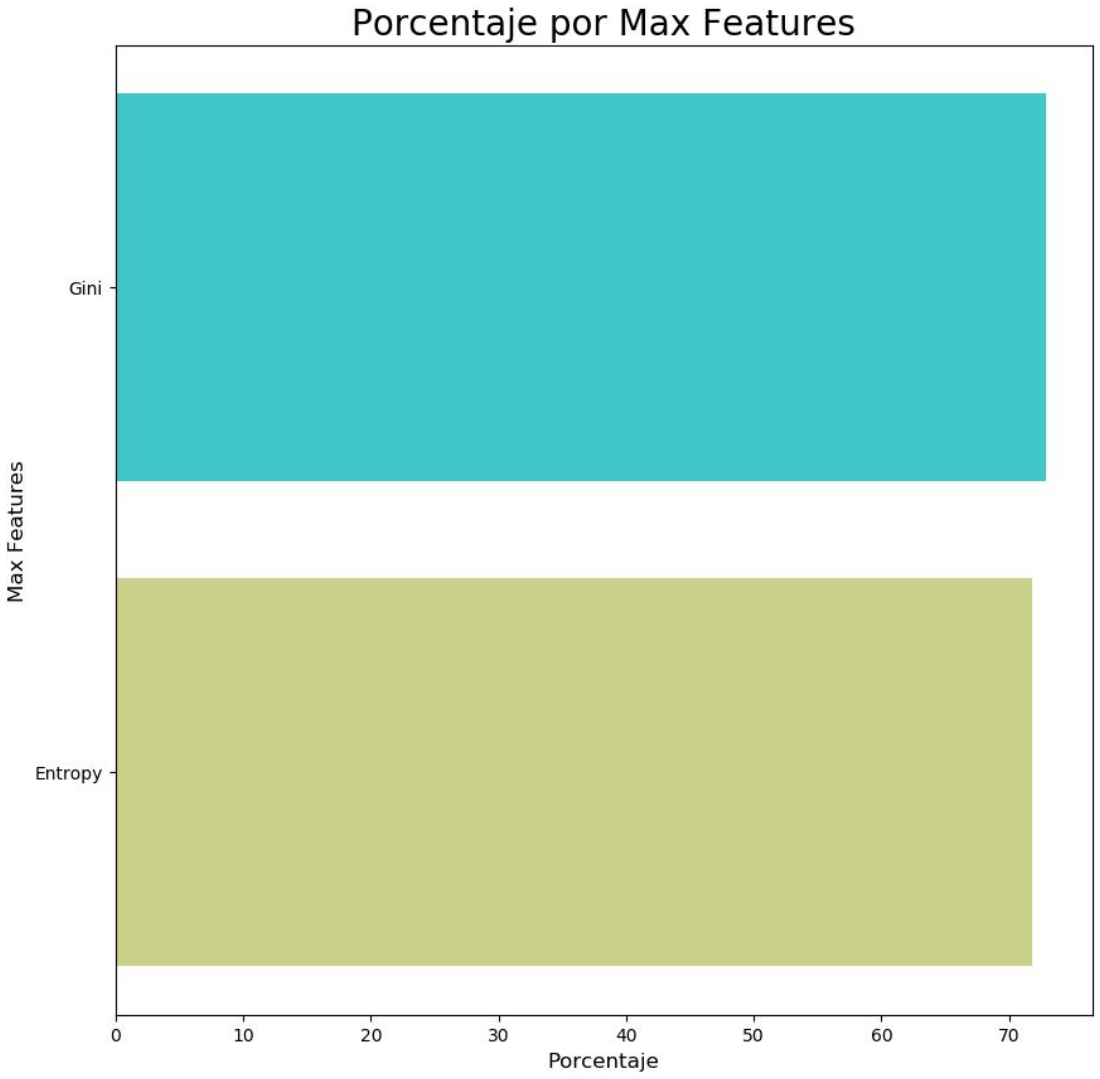

Figura 2.11: Porcentaje por Max Features

Aquí nos dió una diferencia más grande pero,a su vez, también notamos que el valor por default (que es ***auto***) es el de mejor resultado, ya que para los otros valores son menores.

3.9. Ada Boosting

La base de este algoritmo consiste en entrenar una secuencia de algoritmos débiles en versiones modificadas de los datos, esto quiere decir, que al entrenar un nuevo algoritmo se intenta mejorar la predicción de los datos que clasificaron mal en el algoritmo anterior. Las predicciones de todos estos algoritmos son combinados usando el promedio ponderado

con sus respectivos pesos para generar la clasificación. Los hiper parámetros en este caso fueron:

- ***random_state:*** Igual a todos los anteriores, nuevamente seteado en ***0***.
- ***n_estimators:*** Al igual que en Random Forest, seguimos con la estructura de realizar un ciclo con varios valores para ver cuál fue el mejor, el resultado fue el siguiente

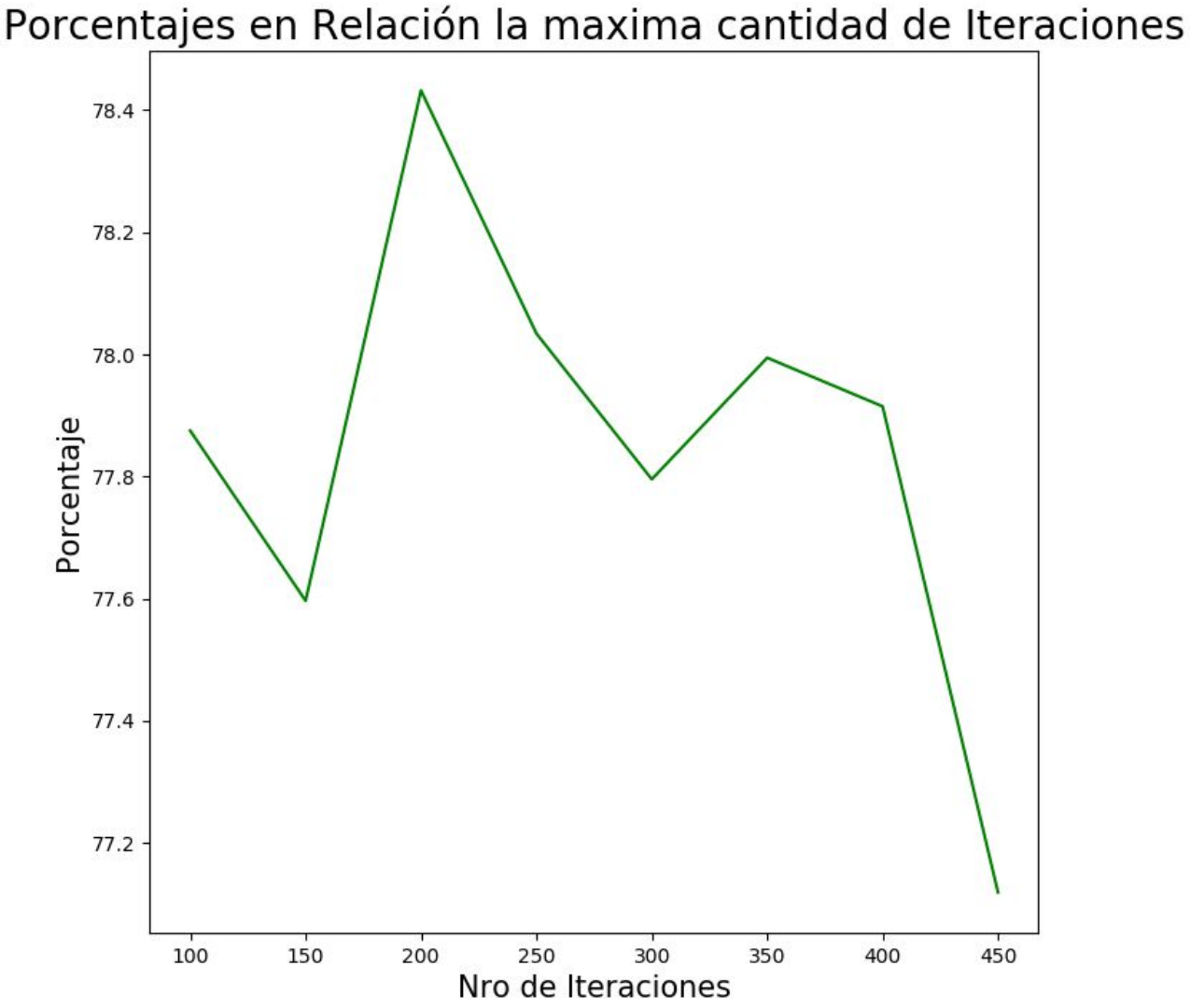

Figura 2.12: Porcentajes en Relación a la máxima cantidad de Iteraciones

El de mejor resultado fue ***200*** con un valor de 78.43% para ser exactos, podemos ver que la diferencia con los demás fue bastante comparado con los demás algoritmos con el mismo tipo de estructura.

3.10. Gaussian Naive Bayes

Este algoritmo está basado en el teorema de Bayes, es extremadamente sencillo, rápido y funciona bastante bien en la mayoría de los casos aunque casi nunca es el mejor pero tampoco el peor. En cuanto a los hiper parámetros, solamente hay dos los cuales, en nuestro caso particular, no generaron ningún tipo de diferenciación en cuanto al resultado, siendo el mejor con un porcentaje del 79.07%.

3.11. Bagging de Random Forest

Dado que Random Forest fue el que mejor resultado nos dió, nos pareció interesante ver que sucede en el caso utilizar Bagging, el cual lo que hace es aplicar el mismo clasificador (en este caso Random Forest) n veces y luego promedia sus resultados para obtener el resultado final. Los hiper parámetros de este son ***n_estimators*** y ***random_state***, seteado en ***500*** y ***0*** respectivamente. Creíamos que en este caso los resultados iban a ser mejores pero utilizando como algoritmo base el mejor de Random Forest , el resultado de usar Bagging fue de 80.54%.

3.12. Majoritary Voting

Es la utilización de Majoritary Voting, el cual genera un resultado final en base a qué clase es la que tiene mayoría entre todos los clasificadores. Para este algoritmo, los hiper parámetros son:

- ***estimators:*** Los algoritmos a utilizar para la votación
- ***voting:*** El tipo de voto a tener en cuenta, en este caso se utilizó ***hard***, el cual elige la clase en base a la mayoría.

Se utilizaron los algoritmos de ***Random Forest*** con ***n_estimators*** en ***500***, ***600*** y ***750*** pero así mismo, no nos dió mejor resultado que el que utiliza ***Random Forest*** solo, ya que este tiene un porcentaje del 81.26%.

3.13. Ensamble Manual de Random Forest

Por último se probó hace un ensamble manual, el cual, calculaba todas las predicciones de ***Random Forest*** desde ***100*** a ***1150 n_estimators*** y luego, se hacía un

promedio y redondeo para calcular en qué clase quedaba. Los resultados de este caso no fueron tan buenos como se esperaba, ya que llegó a un 79.83%, el más bajo de todos los algoritmos que utilizamos con ***Random Forest***.

3.14. Catboost

Probamos utilizando también un algoritmo que utilice gradient boosting en árboles de decisión , en este caso encontramos ***Catboost*** el cual lo utiliza

En el pre-procesamiento de datos, para generar nuevos features a partir de los tweets utilizamos la librería para el procesamiento de lenguaje llamada Spacy, el cual a partir de un texto en un idioma dado (en este caso ingles) obtiene por ejemplo los sustantivos, verbos , adjetivos ,etc . En este caso puntual los feature nuevos que generamos para entrenar con este algoritmo en particular fueron :

- 'cant_verbos'
- 'cant_adjetivos'
- 'cant_sustantivos'
- 'cant_simbolos'
- 'cant_alfanumericos'
- 'cant_palabras'
- 'longitud_texto'
- 'fire'
- 'LIKE'
- 'bomb'
- 'signopregunta'
- 'sustantivos'

Los hiperparametros utilizados fueron:

- ***custom_loss:*** Es la métrica utilizada para entrenar, en este caso la que mejor resultado dió fue ***Accuracy.***
- ***random_seed:*** lo mismo que el ***random_state*** explicado en los otros algoritmos, en este caso, luego de haber probado varios valores con un for, el mejor resultado fue ***42.***
- ***logging_level:*** El tipo de logging que da como salida , en este caso se dejó ***Silent*** ya que deja la salida de la manera original.

Los features cómo 'fire', 'LIKE', 'bomb' son booleanos si tiene esas palabras en el tweet o no, se me ocurrió poner esas palabras en específico ya que en el Análisis Exploratorio del Trabajo Práctico Nro 1, estas palabras se destacaban sobre las otras , en específico la palabra Fire era muy recurrente en los tweets falsos,

Los resultados en contra el set de entrenamiento utilizando como métrica el Error cuadrático medio fue de 0.34459 y contra el set de test fue de 0.39362, el algoritmo con los hiperparametros que utilizamos y los features que mencionamos pareció no estar dando ovefitting y lucía bien , pero aunque el puntaje en Kaggle que dio fue de 67.39%, de manera local nos llegó a dar un score del 80.06%.

3.15. Wordvec + Ada Boosting

Viendo el video de Redes convoluciones de la materia , donde se explicaba las redes convoluciones para imagenes y decia que tambien se podria aplicar para texto , llevando los textos a un vector representando cada palabra a un vector por ejemplo embedding , y se mencionaba ***werdvec*** para crear embedding que tienen embedding ya creados para el idioma inglés (nuestro caso)

Para el pre procesamiento además de aplicar el embedding de wordvec hay que eliminar los nulos o reemplazarlos, nosotros decidimos reemplazarlo por el promedio de la columna una vez hecho eso definimos el modelo con sus hiperparametros para el algoritmo de ***Ada Boosting*** (estos no serán explicados ya que fueron hechos en el apartado 3.9):

- ***n_estimators:*** 800
- ***random_state:*** 1

Los resultados en contra el set de entrenamiento utilizando como métrica el Error cuadrático medio fue de 0.16124 y contra el set de test fue de 0.34520 , y finalmente el porcentaje una vez submiteado en Kaggle fue de 77.22%.

3.16. Conclusión General

Para finalizar este capítulo, veremos el promedio de porcentajes de todos los algoritmos

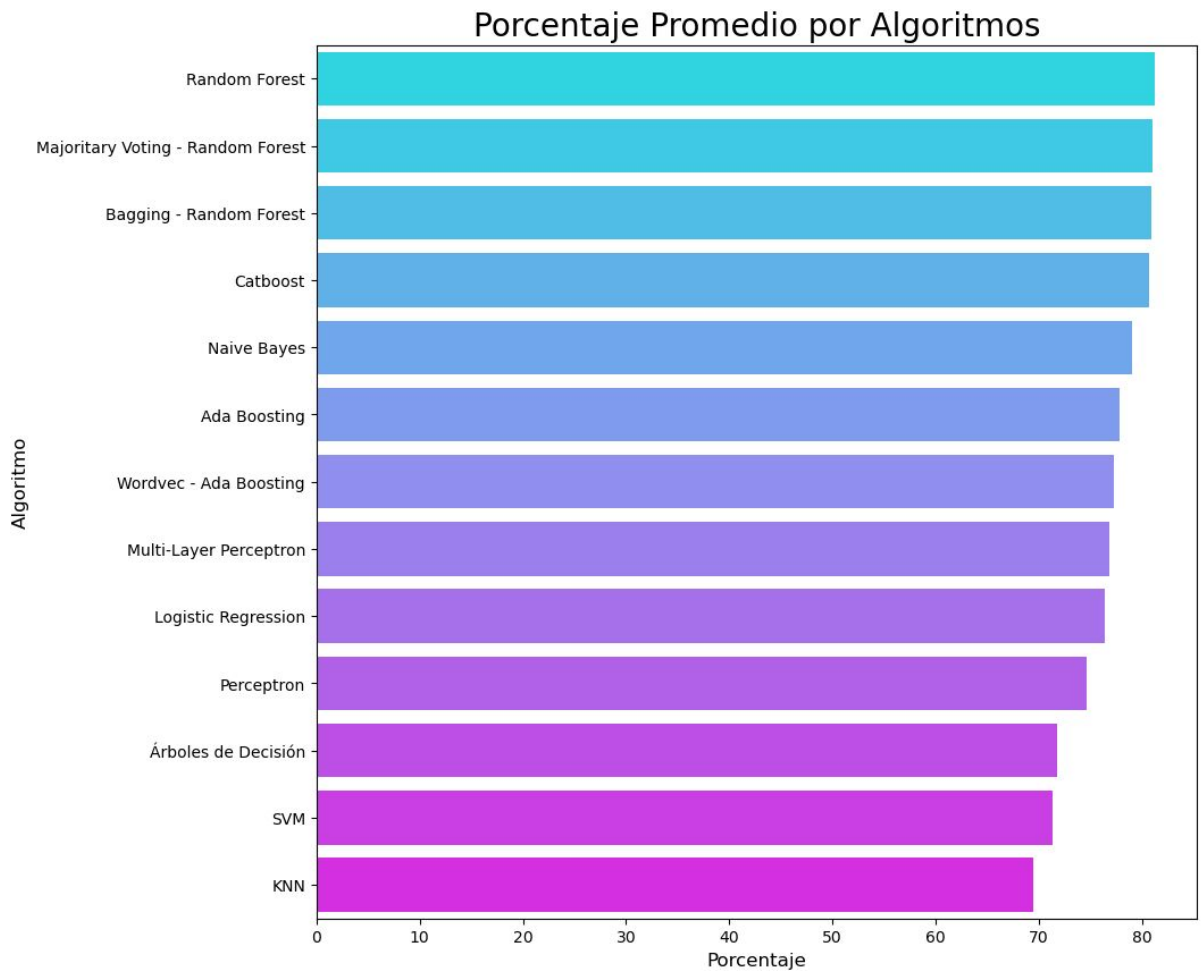

Figura 2.13: Porcentaje Promedio por Algoritmo

Como habíamos mencionado a lo largo de todo este capítulo, vemos que ***Random Forest*** es efectivamente el que mejor porcentaje nos dió, seguido por el ***Majoritary Voting*** y ***Bagging - Random Forest***, esto obviamente debido a que el algoritmo base utilizado es justamente Random Forest. Luego vemos que ***Catboosting*** nos dio resultados bastante buenos, aunque tiene sentido ya que es un ensamble de árboles de decisión, similares a los que están inmediatamente por arriba de este. Les sigue ***Naive Bayes*** lo cual es sorpresivo ya que no suele ser un algoritmo que funcione tan bien pero parece ser que se adaptó de buena manera a nuestro set de datos. Luego de este último, vemos que las diferencias no fueron tan grandes, la particularidad es que tan bajo se encuentran los ***Árboles de Decisión***, comparado con ***Random Forest***. Y vemos que ***KNN*** fue el que peores resultados arrojó, lo cual es comprensible dado que si el mejor resultado lo dio con el 1 como cantidad de vecinos.
Ahora si vemos los resultados por el tipo de algoritmo

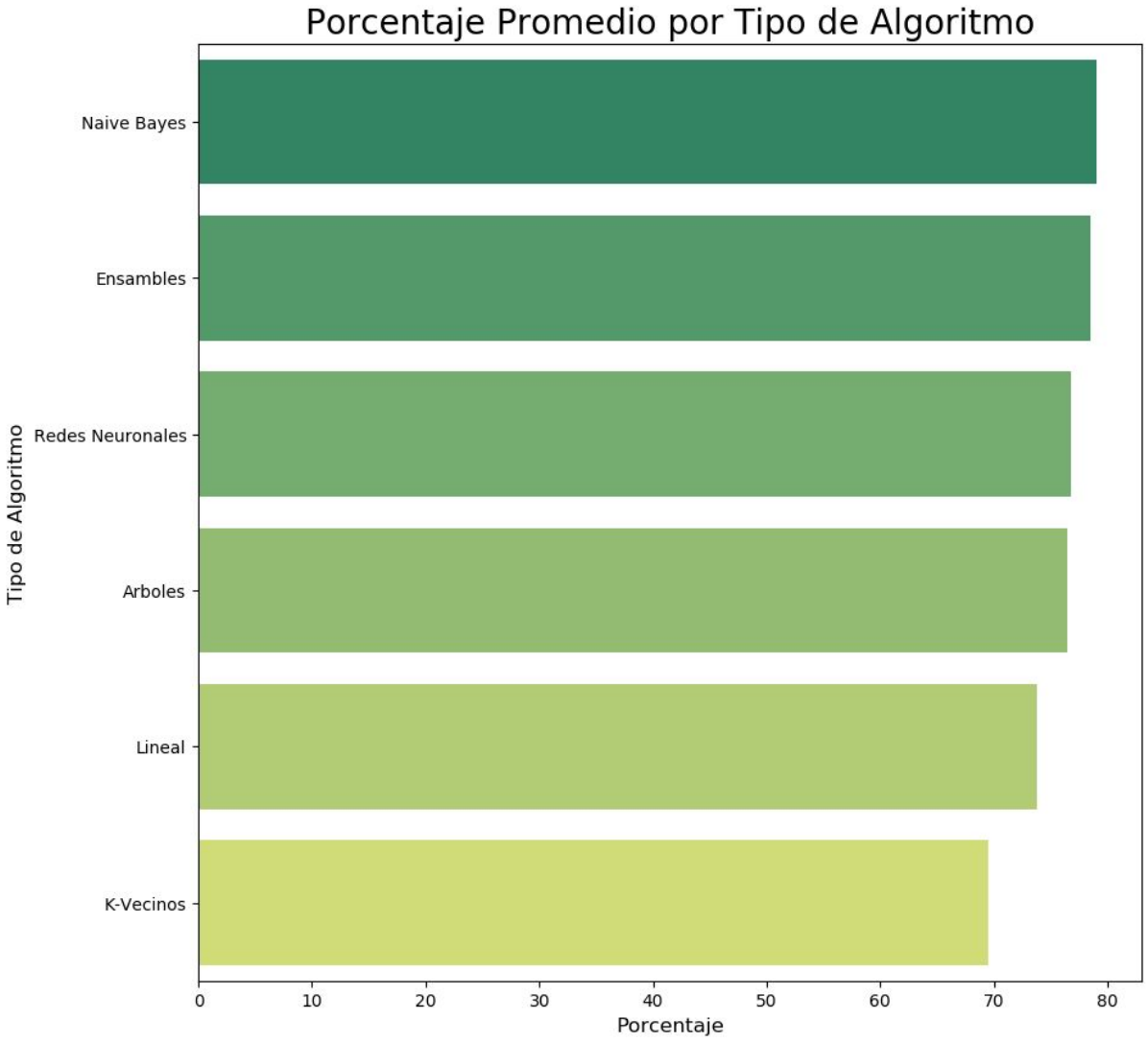

Figura 2.14: Porcentaje Promedio por Tipo de Algoritmo

Vemos como ***Naive Bayes*** es el que mejor promedio nos dió, totalmente esperable dado que es el único algoritmo utilizado de este tipo mientras que los demás se encuentran conformado con más de uno. Luego le siguen los ***ensambles***, comprensible también que tengan un buen promedio dado que ambos se hicieron con el mismo algoritmo base como se mencionó anteriormente, el cuál es el que mejor resultado nos dió. Podemos ver también como en el caso de los ***Árboles*** y los ***Lineales***, como los ***Árboles de Decisión*** y ***SVM*** influyeron de manera negativa en este promedio, dado que los demás algoritmos que forman parte de este tipo dieron resultados bastante buenos. Y por último, así como ***Naive Bayes*** dio el de mejor promedio por tener un solo algoritmo, que ***K-Vecinos*** sea el de peor resultado no debería extrañarnos ya que solamente está conformado por ***KNN*** y sus resultados fueron muy bajos.

4. Conclusión Final

Para finalizar, vamos a hacer una recapitulación. En el primer capítulo se pudo ver el procesamiento que se hizo, tanto con la locación de la cual se hizo una limpieza para poder obtener de una manera más fiel el País y Estado (siempre y cuando el Estado pertenezca a los Estados Unidos) al que se refería, así mismo una mejor limpieza del texto, el cual a nuestro criterio, es la que mayor influencia tuvo sobre el resultado final y fue la que más se intentó procesar. Tambíen como fueron utilizados los distintos CSVs que fueron extraídos, cómo manejar las distancias, la cantidad de palabras, la manera de manejar los nulos y los distintos símbolos que son importantes en una red social, tal como el hashtag y las Menciones. De todo lo que fue explicado, todo fue utilizado en el resultado final, lo unico que se descartó fue que se intentó utilizar los sustantivos así como se hizo con los verbos y adjetivos pero los resultados no eran consistentes, mismo sucedió con el análisis del idioma del texto, esto también se intentó utilizar en el Análisis Exploratorio pero nuevamente, nos dimos cuenta que los resultados no eran lo esperado ya que no eran precisos.

Con respecto a los algoritmos, como se aclaró en varios lados, Random Forest fue el que mejor resultado nos dió tanto en la competencia de Kaggle como en la verificación de manera local, aunque el porcentaje en esta última sea un poco superior. Se intentó probar la mayor cantidad de algoritmos posibles y de todos los tipos posibles para tener un entendimiento de cómo podían llegar a estar distribuidos los datos. Hubo muchos algoritmos que quisieron ser usados pero había que hacer muchos cambios al set de datos, otros porque no tenían mucho sentido utilizarlos por como era el funcionamiento del mismo y otros fueron probados pero sus resultados fueron muy malos.

La idea de este informe fue la de mostrar todo el procesamiento que se hizo con los datos y el porque hicimos ese procesamiento en particular y luego la de dejar plasmados los algoritmos que fueron utilizados, con una breve explicación de los mismos y de sus hiper parámetros.