UNIVERSIDAD DE BUENOS AIRES

FACULTAD DE INGENIERÍA

75.06 Organización de Datos

Trabajo Práctico 2

Segundo Cuatrimestre de 2019

Grupo 30: DataHunter

Integrantes	Padrón
Bobadilla Catalan, German	90123
Calvani, Sergio Alejandro	98588
Fernandez Pandolfo, Franco	100467

Link a Github: https://github.com/bobadillagerman/75.06-Datos-TP2-2019

Índice

1. Introducción

El objetivo del presente informe es el de explicar el criterio de cómo se modelaron los datos para la utilización de los distintos algoritmos de Machine Learning. El mismo será divido en tres secciones:

- **Procesamiento de datos:** Aquí se mostrarán los distintos criterios y como fueron procesados los datos con el fin de mejorar la predicción.
- **Algoritmos:** Explicación de los algoritmos utilizados, por qué lo utilizamos y los distintos hiper parámetros que fueron probados.
- **Conclusión Final:** Para finalizar, una conclusión de cuál fue el algoritmo que mejor funcionó, qué resultado fue el mejor y porque creemos que funcionó de mejor manera.

2. Procesamiento de Datos

Este es la parte más larga en todo proceso de Machine Learning ya que se basa en ir analizando los datos para ver cuales nos sirven para que nuestro algoritmo mejore en su predicción. Cabe aclarar que se utilizó información obtenida durante el primer Trabajo Práctico. Así mismo también describiremos las cosas que probamos pero no generaron ningún tipo de mejora en nuestra predicción. Tambien las columnas de *'piscina'*, *'antigüedad'*, *'centroscomercialescercanos'*, *'escuelascercanas'*, *'idzona'*, *'garage'*, *'baños'*, *'usosmultiples'*, *'metroscubiertos'* y *'metrosdecubiertos'* fueron dejados tal y como está, ya que como se vio en el Análisis Exploratorio, son datos numéricos que no tenía sentido procesarlos.

2.1. Está Capital

Como se pudo ver en el Análisis Exploratorio, en todos los Estados de México la mayor concentración de propiedades en venta se encontraba dentro de las capitales de dichos estados, por lo que creamos un feature en el que nos dice si dicha propiedad está en una capital, siendo 1 si se encuentra o 0 si no se encuentra.

2.2. Tipo de Propiedad

Dado que los tipos de propiedades no son tantos, utilizamos el proceso de One Hot Coding, en el cual, genera un feature por cada tipo de propiedad y coloca un 1 en la columna correspondiente al tipo de propiedad en cuestión y 0 en las otras.

2.3. Provincia

Con las provincias utilizamos el mismo procedimiento que con el tipo de propiedad, dado que la cantidad de provincias es chica, hicimos One Hot Coding.

2.4. Año

Con la fecha, simplemente nos quedamos solo con el año.

2.5 Ciudad

A diferencia de las provincias, el número de ciudades es muy grande, por lo que no pudimos usar el One Hot Encoding. Probamos el Binary Encoding pero lo que nos dio una mejor predicción fue el Target Encoder.

2.6. NaNs

En el caso de los NaNs, en un principio los reemplazamos por cero o algún otro valor que no se encontraba dentro de los valores que tomaba la propiedad en dicho feature. Pero luego descubrimos que daba una mejor predicción llenar esos valores utilizando la media de cada columna en cuestión. También probamos usar la mediana, pero el promedio nos daba una mejor predicción.

2.7. Zona Turística

Mismo procedimiento al utilizado con las capitales, se dispusieron un par de ciudades, las cuales son parte de la zona turística de México y luego creamos una columna en la cual hay un 1 si pertenece a una zona turística importante o 0 si no lo está.

2.8. Zona Metropolitana

Creamos un nuevo feature en el que indicamos, según la ciudad y provincia de la propiedad, en que zona metropolitana se encuentra. Luego a este feature le aplicamos One Hot Encoding.

2.9. Provincia cara

Para la creación de este nuevo feature, calculamos el precio promedio de cada provincia, e indicamos con un uno si se encuentra en el top ten de provincias con el precio promedio más elevado.

2.10. Descripción

Para sacarle provecho al campo de Descripción (el cual es texto) primero nos encargamos de eliminar las palabras que no nos son de utilidad como artículos, pronombres, preposiciones, etc. (stop words). Luego aplicamos el método llamado stemming el cual consiste en reducir todas las palabras a su raíz o stem.

Una vez hecha esa “limpieza”, se procedió a buscar las 100 palabras más repetidas de las propiedades más caras. Luego se hizo lo mismo, pero buscando las 100 palabras más repetidas de las propiedades más baratas. De estas dos listas, descartamos las palabras que aparecen en ambas listas con aproximadamente la misma frecuencia, quedándonos solamente con las palabras que aparecen en una sola de las listas, o que aparecen en ambas, pero con una notoria diferente frecuencia.

Con cada una de esas palabras creamos diferentes features los cuales indican con un uno si la palabra se encuentra en la descripción de la propiedad, o cero en caso contrario.

2.11. Títulos

Con los Títulos, los cuales también son textos, hicimos exactamente lo mismo que con la Descripción.

2.12. Set a Predecir

Todo lo que explicamos anteriormente tuvo que ser aplicado al set de datos a predecir para poder realizar la predicción.

2.13. Procesos que no mejoraron la predicción

Ahora enunciaremos los distintos tipos de procesamientos que estuvimos probando, pero no generaron ningún tipo de mejora.

2.13.1. Zona

En este caso intentamos relacionar cada provincia por la zona del país en donde se encontraba, de la siguiente manera

	Zona	provincia
0	nor_oeste	Baja California Sur
1	nor_oeste	Baja California Norte
2	nor_oeste	Chihuahua
3	nor_oeste	Durango
4	nor_oeste	Sinaloa
5	nor_oeste	Sonora
6	nor_este	Coahuila
7	nor_este	Nuevo León
8	nor_este	Tamaulipas
9	occidente	Nayarit

Figura 1.1: DataFrame de las Zonas

Luego por cada zona realizamos un one hot encoding para generar los features correspondientes. Este caso no impactó de ninguna manera en los resultados.

2.13.2. Latitud y Longitud

Entrenar y predecir con la latitud y longitud empeoraba nuestra predicción .También quisimos procesar las longitudes y latitudes, debido a que son las columnas con mayor cantidad de datos faltantes, intentamos llenar estos dos valores con la mediana de cada provincia. La idea detrás de estos era que, como cada propiedad tiene una longitud y latitud distinta, era innecesario utilizarlo como feature ya que es un valor único por propiedades, en cambio, al poner un valor general para cada provincia podía generar una mejor predicción, pero en este caso, no sucedió nada.

2.13.3. Mes

Entrenar y predecir con el mes no mejoraba la predicción. La empeoraba ligeramente.

2.13.4. Centros Comerciales más importantes

Dado a que como pudimos observar en el Análisis Exploratorio, que el precio de las propiedades con los Centros Comerciales cercanos aumentaba, quisimos probar si relacionando a cada propiedad con los 10 centros comerciales más importante de México (la información se obtuvo de este link[1]) generaba algún tipo de mejora. Creamos un DataFrame que quedaba de la siguiente manera:

	centro_comercial	ciudad	lat	lng	provincia
0	Perisur	Coyoacán	19.303899	-99.189435	Distrito Federal
1	Parque Tezontle	Iztapalapa	19.383660	-99.082329	Distrito Federal
2	Las Américas Cancún	Cancún	21.147655	-86.823800	Quintana Roo
3	Antea LifeStyle Center	Querétaro	20.674103	-100.435181	Querétaro
4	Andarés	Zapopan	20.710305	-103.412591	Jalisco
5	Plaza Satélite	Naucalpan de Juárez	19.510331	-99.235556	Edo. de México
6	Forum Buenavista	Cuauhtémoc	19.449796	-99.151794	Distrito Federal
7	Multiplaza Aragón	Ecatepec de Morelos	19.530644	-99.028458	Edo. de México
8	Toreo Parque Central	Naucalpan de Juárez	19.454400	-99.218174	Edo. de México
9	Centro SantaFe	Cuajimalpa de Morelos	19.360861	-99.272838	Distrito Federal

Figura 1.2. DataFrame de los Centros Comerciales más importantes

Para esto, buscamos que si la propiedad tenía un centro comercial cercano, de tenerlo, calculaba la distancia Haversine y se fijaba si se encontraba a distancia menor a 10 KM. Luego, probamos dos enfoques distintos:

1) Poner la cantidad de centros comerciales importantes Cercanos
2) Hacer una lista de los Centros Comerciales cercanos más importantes y luego generar los features con one hot encoding

Ninguno de estos dos enfoques sirvió, ya que empeoró nuestros resultados.

[1] https://tipsparatuviaje.com/centros-comerciales-mas-grandes-de-mexico/

2.13.5. Estaciones

Habiendo analizado el tema de las estaciones en el TP1, quisimos ver si en este caso generaba alguna mejora en el resultado. Para ello, hicimos nuevamente one hot encoding con cada estación para generar los features en cuestión, pero no sirvió de nada.

2.13.6. Metros Descubiertos

También intentamos colocar los metros descubiertos como features, siendo esto la diferencia entre los metros totales y los cubiertos, pero nuevamente no genero ningún tipo de mejora.

2.13.7. Agrupar Antigüedad

Se nos ocurrió también agrupar a la antigüedad por rango de años, probamos cada 5 años o cada 10 años, como habíamos notado en el TP1 que había una anomalía con estos dos valores pero utilizando la antigüedad de la manera que estaba nos daba mejores resultados por lo que descartamos esta opción.

2.13.8. Avenida

En base al feature Dirección, creamos uno nuevo en el que indicábamos con uno si la propiedad se encontraba en una avenida. El feature no presentó mejoras.

2.14. Conclusión General

Entrenar solo con los valores numéricos nos dio una buena primera aproximación, pero para lograr una mejor predicción se tuvo que codificar las variables categóricas lo cual nos fue muy útil mejorando mucho nuestro puntaje en Kaggle. Posteriormente creamos nuevos features con información externa y de público conocimiento, algunos de ellos mejoraron la predicción, como indicar si la propiedad se encuentra en la capital del Estado o en una zona turística, pero otros no, como la zona (puntos cardinales) o la estación del año.

Sin embargo ya sea que mejoren o empeoren nuestras predicciones, siempre fue poco el efecto. Lo que produjo una significativa mejora fue trabajar con los features que tienen texto como la descripción y el título. Es lo más dificultoso de todo, pero puede rendir muchos frutos.

3. Algoritmos

A continuación, enunciaremos los distintos algoritmos que utilizamos, una breve descripción del mismo, el porqué de su utilización, sus hiper parámetros y los resultados que nos dieron. En todos los casos fueron probados distintos valores para los hiper parámetros y algunos, fueron buscados mediante un ciclo y otros, mediante otros métodos como Grid Search.

3.1. Métrica

Antes de comenzar a hablar de los algoritmos, nos pusimos de acuerdo en que íbamos a utilizar la métrica de MAE (Mean Absolute Error) para calcular el error de cada algoritmo y ver su precisión en relación al set de validación de manera local.

$$\mathrm{MAE} = \frac{1}{N}\sum_{i=1}^{N}|y_i - \hat{y}_i|$$

Figura 2.1: Mean Absolute Error

3.2. Random Forest

Random Forest es uno de los algoritmos que mejor suele funcionar y que es poco propenso a que tenga overfitting, ya que es una aplicación directa de bagging sobre los árboles de decisión. Debido a esto, pensamos que podía llegar a dar buenos resultados, lo complicado es encontrar los hiper parámetros óptimos, en este caso son:

- **n_estimators:** cantidad de árboles a utilizar, en este caso notamos que a mayor cantidad mejor era la predicción, ya que cuando esté hiper parámetro el error era menor, pero llegó un momento en el que al aumentarlo no generó ningún tipo de diferencia. En este caso el valor óptimo fue de 450.
- **random_state:** Refiere a la distribución random, en este caso utilizando distintos valores no notamos ningún tipo de cambio por lo que lo dejamos en 0
- **n_jobs:** Refiere a la cantidad de procesadores que usa, en nuestro caso lo seteamos en -1 para que utilice todos los procesadores disponibles
- **boostrap:** Esto es si se usa boostrap en los árboles, esto es, tomar una muestra del set de entrenamiento del mismo tamaño del set de entrenamiento, pero con

reemplazo. En nuestro caso lo pusimos True el cual también es el default ya que tampoco notamos una diferencia en el resultado.

- **criterion:** Es el tipo de función que se usa para medir la calidad de la división. En este caso utilizamos MSE.
- **max_depth:** Esta refiere a la profundidad de los árboles, en este caso también notamos que, modificando el valor, el resultado cambiaba, aunque no de gran manera.

En conclusión, usando este algoritmo los resultados fueron favorables, siendo el error promedio de 564363, con 525614 como el mejor resultado.También notamos que, cuanto más grande era el set de entrenamiento, mejor predecía y, a pesar de modificar los datos, este se mantuvo siempre como el mejor predictor.

3.3. KNN

Este es uno de los más populares y que suele utilizarse en la mayoría de los casos, ya que se basa en buscar los K-vecinos más cercanos. En este caso, los hiper parámetros son:

- **n_neighbors:** Es la cantidad de vecinos cercanos que queremos. En este caso probamos de manera iterativa distintos valores de K en un rango del 1 al 50 y estos fueron los resultados

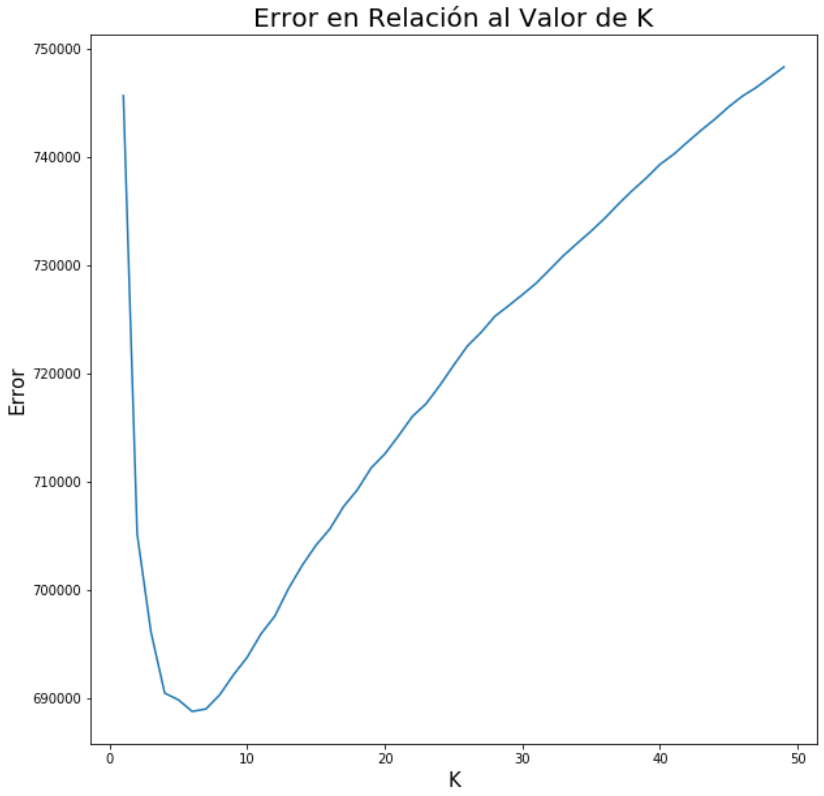

Figura 2.2: Error en relación a los valores de K

En todos los casos vimos que los valores entre 4 y 7 son los que mejor resultado daban, como se puede observar en gráfico anterior.

- **algorithm:** Es el algoritmo utilizado para calcular los vecinos más cercanos, el que mejor resultado nos dio fue el 'kd_tree', el cual es un tipo de índice especial el cual es similar a un árbol binario pero cada nivel está relacionado con alguna coordenada de nuestras n-dimensiones.
- **metric:** La métrica a utilizar, en este caso, el que mejor resultado nos dio es la 'manhattan', la cual, es la forma de calcular distancias en una ciudad en la que solo nos podemos mover de manera vertical u horizontal.
- **n_jbos:** Igual al descrito en Random Forest y nuevamente lo seteamos en -1 para utilizar todos los procesadores.

KNN en este caso no nos dio los mejores resultados, teniendo un promedio de error de 758195, con el de menor error de 742413, y a pesar de todo, es el que más cerca estuvo en rendimiento a Random Forest. También pudimos notar que era muy sensible a cualquier tipo de modificación que hiciéramos sobre los datos.

3.4. Árboles de Decisión

Ya que habíamos probado Random Forest, el cual es un árbol de decisión de por sí, pensamos que sería interesante probar árboles de decisión de manera directa. Este es un árbol binario en el que cada nodo es dividido de acuerdo a algún tipo de criterio. En este caso no vamos a analizar en detalle los hiper parámetros ya que son prácticamente iguales a los que se enunciaron en Random Forest. En cuanto a los resultados, el error promedio es de 916369 y el menor error 856589, si bien se podría pensar que iban a ser similares a los obtenidos en Random Forest debido a que este último es un tipo de árbol de decisión, esto no fue así.

3.5. Extra Tree Regressor

Este algoritmo puede ser visto como un ensamble de Árboles de Decisión, ya que Implementa un estimador que ajusta un número de árboles de decisión con varios sub muestras del set de datos y usa el promedio para mejorar la precisión y controlar el overfitting. Los hiper parámetros son similares a los vistos en Arboles de Decisión y Random Forest, ya que tiene *'n_estimators'*, *'criterion', 'random_state', 'max_depth'*,etc. En cuanto a resultado, el error promedio es de 1019129 con el menor error de 869559, estos resultados no llegaron ni de cerca a ser similares a los obtenidos con Random Forest sino a los de Árboles de Decisión, lo cual tiene sentido con lo que enunciamos al principio de este apartado, lo extraño es que los resultados fueron peores con respecto a estos últimos, ya que al ser un ensamble de estos, los resultados deberían haber sido un poco mejores.

3.6. Bagging

Este concepto viene de los ensambles, el cual es la idea de combinar varios algoritmos. En el caso particular de bagging, implica aplicar el mismo clasificador n veces y luego promediar sus resultados para obtener el final. Así mismo utiliza el concepto de boostraping, explicado en el algoritmo de Random Forest. Dado que Random Forest fue el que mejor resultado nos venía dando, nos pareció interesante ver a que se podía llegar con un bagging de este, los resultados fueron 697771 como error promedio y 689815 como menor error, lo cual genera una diferencia bastante grande con los resultados observados con Random Forest solo. Respecto a aplicar bagging con KNN nos dio 762308 como error promedio y 729645 como error mínimo, en este caso, nos dieron resultados similares a KNN solo, es más, el error mínimo en este caso es menor.

3.7. Ada Boosting

La base de Ada Boosting 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 una predicción final. Con respecto a los hiper parámetros, utilizamos los siguientes:

- **n_estimators:** Mismo a lo visto en los algoritmos anteriores, en este caso lo seteamos en 450.
- **learning_rate:** Refiere a la contribución de cada regresor, en este caso probamos varios valores como puede verse en el siguiente gráfico.

Figura 2.3: Error en relación al Valor de Learning Rate

Se puede observar que hay altibajos y no es constante, y el valor que menos error generó fue 7.

- **loss:** La función que se utiliza para calcular cuales son los datos que peor clasificaron y darles más relevancia al correr el siguiente regresor, en este caso el que mejor resultado nos dio fue el 'exponential'.

Los resultados fueron 1546931 como error promedio y 1333081 como el error mínimo, este algoritmo a pesar de que se puede pensar que daría buenos resultados ya que utiliza ensambles y en general, suelen dar buenos resultados, en este caso fue todo lo contrario, de todos los algoritmos probados fue el que peor resultado nos dio.

3.8. RIDGE

Es un método de regresión que utiliza como función de pérdida la función lineal de cuadrados mínimos y como regularización es la norma de la función L2 regularización, la cual refiere a la norma 2 de un vector, para evitar el overfitting.

Los hiper parámetros utilizados fueron:

- **alpha:** Corresponde al alpha de la función de pérdida, de varios valores posibles, el que mejor resultado nos dio fue 1, ya que a medida que este valor crecía, el error se hacía más pronunciado, como se puede ver en el siguiente gráfico.

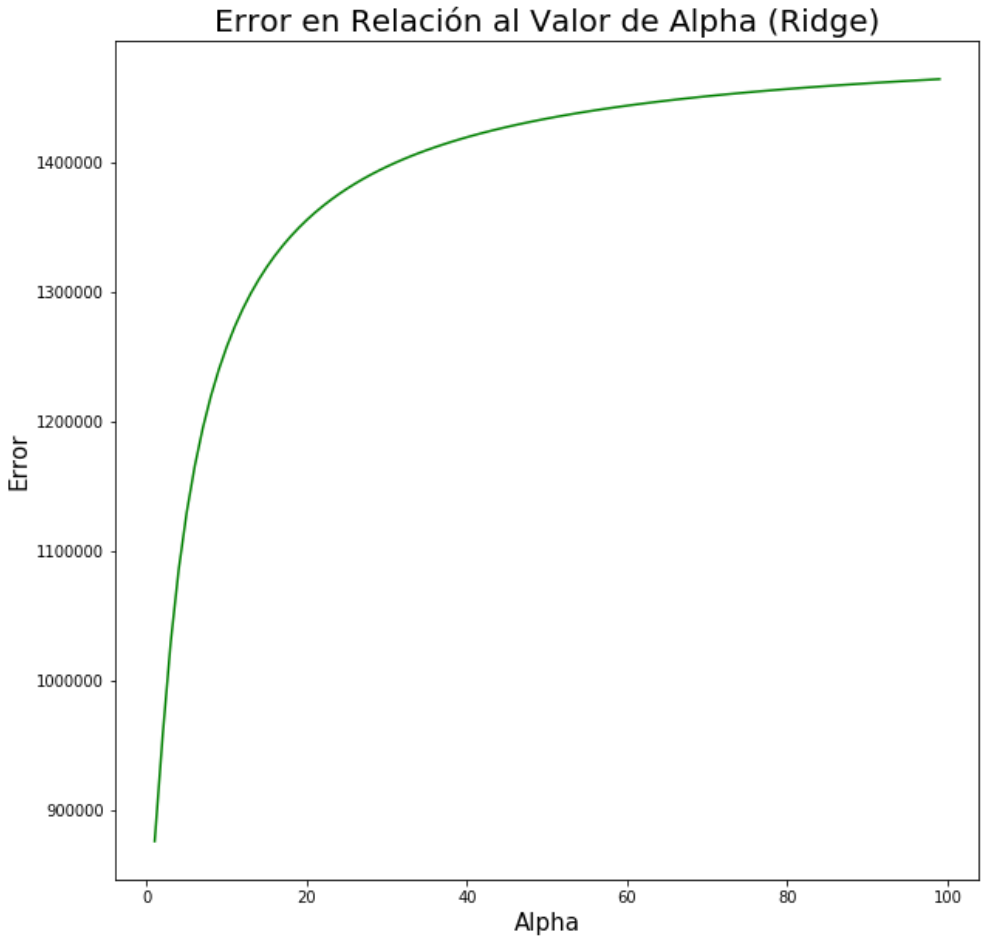

Figura 2.4: Error en Relación al Valor de Alpha (RIDGE)

- **fit_intercept:** Es para ver si intercepta los datos en el cálculo de la función de pérdida, esto quiere decir que los datos no están normalizados
- **normalize:** Este hiper parámetro suele ignorarse si el anterior está seteado en False, en cambio, suele utilizarse en True ya que normaliza los datos antes de hacer la regresión.

- **solver:** Es usado en las rutinas computacionales, en este caso utilizamos ‘svd’, el cual, utiliza la Descomposición de Valores Singulares para calcular los coeficientes de Ridge.

Este algoritmo nos dio como resultado 1074500 como error promedio y el de menor error 937815, nos dio mejor resultado que el Ada Boosting pero sigue estando lejos de los mejores algoritmos tales como Random Forest y KNN.

3.9. Lasso

Es un modo lineal que utiliza la función L1 como regularización, también conocida como Lasso. Se creó para mejorar la exactitud de la predicciones e interpretabilidad de los modelo estadísticos de regresión al alterar el proceso de construcción del modelo al seleccionar solamente un subconjunto de (y no todas) los features provistos para usar en el modelo final.

Los hiper parámetros son los siguientes:

- **alpha:** Corresponde al alpha de la función de regularización, haciendo lo mismo que en el caso anterior, podemos ver, en este caso, el error no aumenta de manera pronunciada, sino que va creciendo lentamente, siendo 8 el valor con menor error.

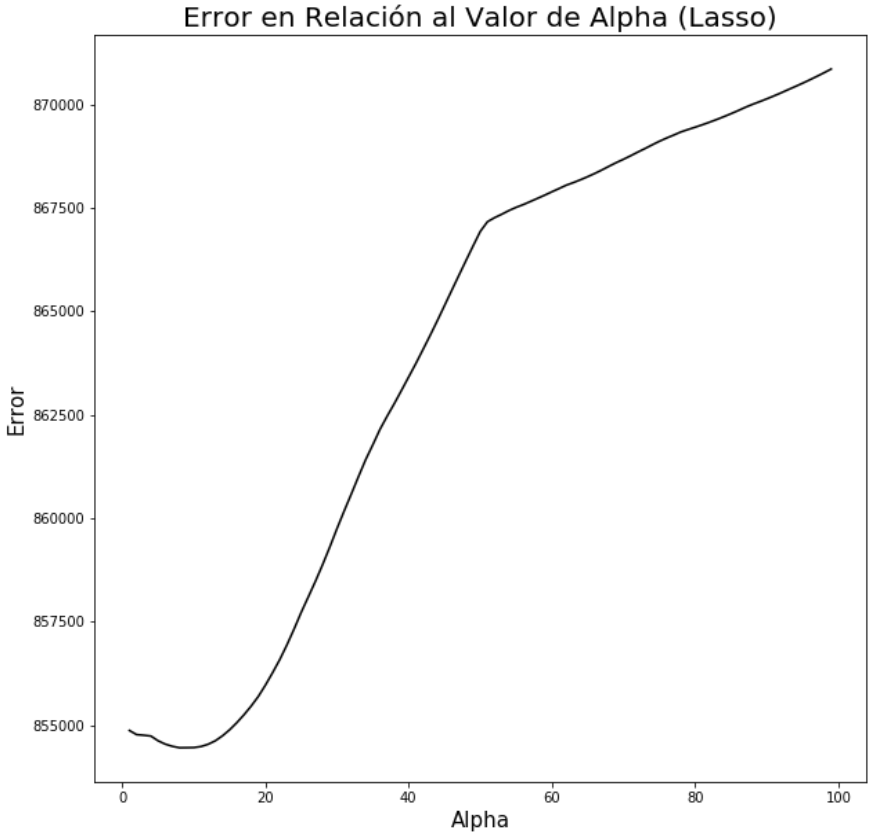

Figura 2.5: Error en relación al Valor de Alpha (Lasso)

- **positive:** Fuerza a que los resultados sean valores positivos, en este caso lo pusimos en True.

El error promedio es de 987229 mientras que el menor error fue de 945845, si bien el error promedio fue más chico que lo visto en RIDGE, el menor error fue más pequeño usando RIDGE, aun así, sigue estando lejos de los algoritmos con mejores resultados.

3.10. SVR

Es la versión regresora de SVM, la cual intentaba encontrar el mejor hiperplano separador, el cual, era el que maximiza el margen entre las clases. Este, al ser regresor, consiste en realizar un mapeo de los datos de entrenamiento $x \in X$, a un espacio de mayor dimensión F a través de un mapeo no lineal, donde podemos realizar una regresión lineal.

Los hiper parámetros son:

- **kernel:** Es el tipo de kernel que usa el algoritmo, en este caso, de todos los probados, el que mejor resultado nos dio fue el que viene por default, el 'rbf'.
- **gamma:** Es el coeficiente de la función de kernel, en este caso, al utilizar 'rfb', debe estar seteado en algún valor, utilizamos 'scale' ya que en vez de hacer el 1 / cant_features, el denominador lo multiplica por SIGO
- **c:** Es la penalización al error, tal como sucede en SVM, dependiendo del valor que se use puede haber overfitting (C muy grande) o undefitting (C más chico). En este caso 3 es el valor que mejor resultado nos dio.
- **epsilon:** Puede ser visto como el margen de tolerancia del modelo de SVR, de todos los valores probados, 0.1 fue el que mejor resultado nos dio.

Los resultados fueron 974873 como error promedio y 893890 como error mínimo, en este caso nos dieron valores similares a los obtenidos con RIDGE y Lasso, lo cual tiene sentido ya que en SVR, se utiliza una regresión lineal, igual a los casos anteriores, por lo que tiene coherencia que sus resultados sean similares.

3.11. XGBoosting

Este algoritmo utiliza el concepto de boosting, el cual fue explicado en el algoritmo Ada Boosting, aplicado a árboles, donde se suman los resultados de cada árbol. Este algoritmo conviene siempre ser usado ya que nos permite saber si estamos haciendo las

cosas bien, ya que aunque no siempre es el mejor, siempre suele dar buenos resultados. Los hiper parámetros en este caso fueron:

- **objective:** El tipo de función de regresión a usar, probamos distintas variantes, siendo la de mejor resultado la de *'squaredlogerror'*.
- **n_estimators:** Cantidad de árboles, de todos los valores posibles 350 es el que mejor nos dio.
- **max_depth:** La profundidad de los árboles, en este caso seteado a 5.
- **learning_rate:** Reduce el peso de los features para evitar overfitting y hacer el proceso de boosting más conservativo, en este caso lo pusimos en 0.1
- **gamma:** El Gamma de la función de XGBoost, en este caso, seteado en 1.

En cuanto a resultado, nos dio 795471 como el error promedio y 770777 como error mínimo. En este caso, no nos dieron valores tan malos en líneas generales, muy similares a los obtenidos por KNN pero igualmente, ni cerca a los que se tuvo con Random Forest.

3.12. Conclusión General

Para finalizar, haremos un pequeño resumen de los errores promedio de cada algoritmo

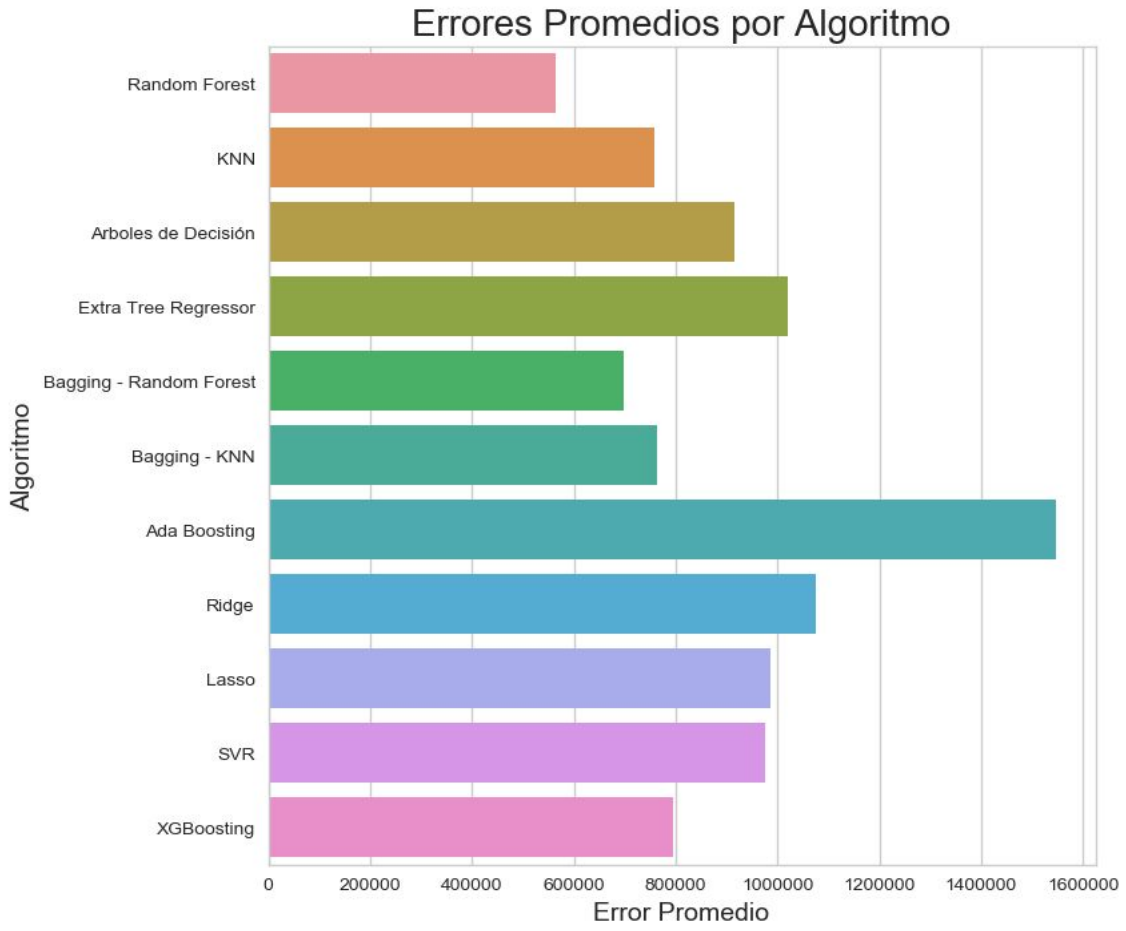

Figura 2.6: Errores promedio por Algoritmo

Podemos ver que Random Forest fue efectivamente el algoritmo que menor error promedio nos dió (por ende el que menor error mínimo nos dio) , luego, despreciando los Bagging tanto de KNN como de Random Forest, le sigue KNN aunque ya la distancia es mayor, al igual que con XGBoosting. Luego puede verse que los demás algoritmos no nos dieron los mejores resultados, ahora bien, sería interesante observar como es el error promedio pero agrupando cada algoritmo con respecto al tipo que es, por ende, veremos el siguiente gráfico.

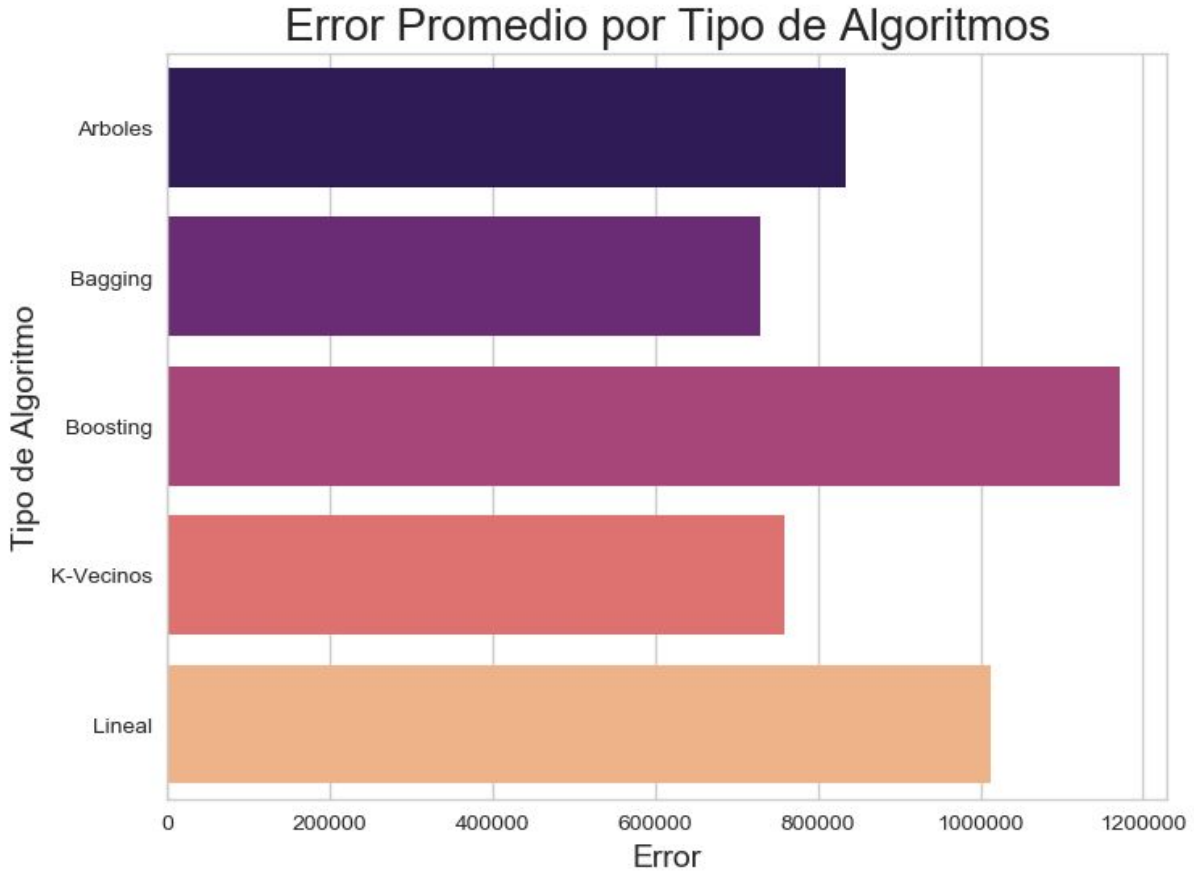

Figura 2.7: Error Promedio por Tipo de Algoritmo

Aquí podemos ver unas cuantas particularidades, por ejemplo que Bagging es el tipo que menor error generó, lo cual tiene sentido ya que nosotros lo utilizamos justamente con los algoritmos que mejor resultado nos dieron. Luego le sigue el de los K-Vecinos (KNN) lo cual es medio obvio ya que solamente tiene a KNN y fue de los que mejor resultado nos dio. Luego podemos ver como los algoritmos de tipo Árboles aumentaron notablemente, a pesar de tener a Random Forest, esto es debido a que con los otros tipos de algoritmos no funcionó particularmente bien. El caso particular está en los modelos de Boosting, los cuales suelen generar buenos resultados y es más, XGBoosting fue de los 3 mejores pero Ada Boosting fue el que peor funcionó, la balanza se inclinó para el segundo, siendo el tipo

con peor error promedio, en cuanto al lineal no hay mucho que decir ya que los tres algoritmos utilizados de este tipo funcionaron de manera similar. Para finalizar, si hablamos del tamaño del set de entrenamiento, en este caso, no tuvo incidencia en el resultado, si bien, cuanto mayor era el set de entrenamiento el error bajaba, la diferencia no era tan grande, por ende, no nos detuvimos a analizar esta cuestión.

4. Conclusión Final

Para finalizar, vamos a decir cuál fue el algoritmo que utilizamos para tener el Score que se encuentra en la competencia y nuestra opinión de todo el trabajo. Con respecto al procesamiento de datos, claramente fue la parte que más trabajo requirió, principalmente la *'descripción'* y los *'títulos',* ya que se necesitaba hacer un análisis de qué palabras eran las necesarias para usar y limpiar dichas columnas para evitar features inconsistentes o innecesarios. También se probaron muchas otras cosas con los datos utilizando información externa pero la influencia en los resultados fue casi nula. Luego con respecto al algoritmo, el que nos dió el resultado de la competencia fue claramente Random Forest con un error de 525614, como habíamos demostrado anteriormente. Creemos que fue el que mejor resultado nos dió principalmente porque es un tipo de algoritmo que suele funcionar muy bien con los problemas de regresión y además, suele generar buenos resultados de por sí. Luego porque fue el primer algoritmo que probamos y todos los ajustes que hicimos a lo largo de todo el trabajo fue en base a este.

La idea de este informe fue la de mostrar cómo procesamos los datos y qué criterios utilizamos para los mismos y luego, mostrar todos los algoritmos utilizados, con la explicación del mismo y sus hiper parámetros.