FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Prediciendo compras de usuarios

[7506/9558] Organización de Datos
Segundo cuatrimestre de 2018

Grupo Random Name

Alumno	Número de Padrón	Email
Peña, Alejandro	98529	alee.pena.94@gmail.com
Blázquez O., Sebastián A.	99673	sebastian.blazquez96@gmail.com
Cavazzoli, Federico	98533	fedecavazzoli@gmail.com
Ruiz, Francisco	99429	fran7ruiz9@gmail.com

Repositorio en GitHub:
`https://github.com/seblaz/Predicting-user-conversions`

Nombre del grupo en Kaggle: # 27 Random Name

Índice

1. Introducción

En el siguiente informe se presentará el desarrollo y el análisis de los distintos algoritmos de Machine Learning utilizados para generar la predicción de la probabilidad de realizar una compra, para cada usuario dado, en el intervalo de tiempo dado desde el 1º de junio de 2018 y el 15 de este mismo mes, en la plataforma virtual `www.trocafone.com`, encargada de la compra, reacondicionamiento y venta de celulares usados y venta de dispositivos nuevos.

El análisis presentado se hizo utilizando Python3 con las siguientes librerías:

- Pandas
- NumPy
- skLearn
- XGBoost
- bayesian-optimization
- Time

Para la integración del trabajo se utilizó un repositorio GitHub: `https://github.com/seblaz/7506-datos-tp2` donde se pueden encontrar los notebooks utilizados tanto para la limpieza de los datos como para los algoritmos de machine learning y de feature engineering.

2. Pre-Procesamiento de los Datos

Al igual que en el primer trabajo práctico de la materia, nos fue dado un set de datos con eventos que se realizaron en la página de Trocafone. Obviamente se trata de una muestra representativa de usuarios que generaron eventos en la plataforma y fueron registrados por la web analytics del sitio.

Se trabajó con el archivo **eventsupto01062018.csv** que nos brindaba diferente información dependiendo del evento que se haya registrado desde el 1º de enero del 2018 hasta el 31 de mayo del 2018 inclusive.

Por otra parte, en el archivo **labelstrainingset.csv**, para un porcentaje de los usuarios del set anterior, tenemos labels que nos informaban si el usuario había comprado (label = 1) o no (label = 0) durante el 1º de junio de 2018 y el 15 de este mismo mes.

Aquellos usuarios que no tienen información en el archivo mencionado previamente, serán el objetivo principal del presente Trabajo Práctico debido a que para ellos debemos encontrar la probabilidad ya indicada.

Utilizando la misma limpieza que se realizó al set de datos para el análisis exploratorio, se vio que los datos estaban en 'buen estado' permitiéndonos trabajar casi con la totalidad de los eventos registrados, exceptuando con algunos que contenían datos nulos o que no nos daban información importante para el análisis.

Hay que destacar que nuestro set de datos **eventsupto01062018.csv** tiene la información dada por eventos a lo largo del tiempo, repitiendo usuarios que tengan más de un evento en la plataforma, por eso debemos transformar toda esta información para que quepa en una sola fila por usuario.

Para finalizar el pre-pocesamiento de los datos, creamos un dataFrame en donde se colocarán todos los features creados en el proceso de Feature Engineering para cada usuario. A este set de datos nuevos, se lo dividirá en dos partes: el 80 % estará destinado al entrenamiento de los modelos y el 20 % restante serán utilizados para testear lo entrenado. Luego, al momento de realizar la suba a Kaggle, realizaremos un entrenamiento final con el set de datos completo.

3. Feature Engineering

El proceso de Feature Engineering es tan o más importante que la elección del algoritmo de Machin Learning, pues de estos features es de donde los algoritmos sacarán la información necesaria para realizar sus predicciones. Debido a esto, le dedicamos especial importancia a esta parte del trabajo práctico, algunos han sido muy útiles y otros no tanto como se hubieran esperado, además de que features bastantes simples han sido elegidos en múltiples ocaciones como los más importantes, esto según las elecciones del Random Forest.

Se separó a los features en dos grandes grupos, los básicos y los features basados en fechas.

3.1. Features básicos

Denominamos features básicos a aquellos que son simplemente una suma, promedio o desvío estándar de alguna de las columnas del set de datos original. Entre ellos se pueden destacar:

- event-count: la sumatoria de los eventos que realizó un usuario.
- viewed product: la cantidad de veces que el usuario visitó un producto.
- screen-resolution-width std
- screen-resolution-width mean
- scree-resolution-height mean

Estos 5 features básicos son a los cuales el Random Forest, generalmente, les dio mayor importancia. Comparando esto con lo obtenido en el análisis exploratorio realizado en el Trabajo Práctico nº1, se puede observar que se mantiene la tendencia de que aquellos usuarios más activos, con mayor cantidad de eventos registrados y mayor cantidad de vistas a los productos se diferencian mucho de aquellos que no lo hacen, suponemos que los primeros tienden a tener una mayor probabilidad a la compra.

Por otro lado, sorpresivamente aparecen features relacionados con el screen resolution de los dispositivos utilizados para visitar el sitio. La razón que le encontramos a la importancia de estas características es que están estrechamente relacionas con el tipo de dispositivo que representan, computadoras o celulares. Como observamos en el trabajo anterior, las compras se realizan mayormente desde computadoras a pesar de que las visitas al sitio son mayormente desde celulares. Esto nos puede llevar a concluir que a mayor valor en estas características, más posibilidades de que el usuario realice una compra en los próximos días.

3.2. Features según fecha

Aprovechando que los eventos tienen la fecha en la cual fueron generados, pensamos que analizar la actividad de los usuarios en la plataforma de Trocafone a lo largo del tiempo, nos iba a dar una gran información. No es lo mismo que una persona haya tenido la mayoría de sus eventos en Enero, que uno que haya tenido la mayoría de los mismos en Mayo, probablemente aquel que desde Enero no visita la página o haya bajado considerablemente su actividad ya haya comprado un celular o se haya bajado de la búsqueda del mismo en esta plataforma. Por este motivo, decidimos crear los features *days until 31-05 mean* y *days until 31-05 std*, ambos fueron tenidos muy en cuenta según el Feature Importance generado por el Random Forest.

Otro tipo de features relacionados con el tiempo, fueron aquellos que nos decían hace cuantos días un usuario no generaba determinado evento. Principalmente se analizaron los checkout, conversion (compras), vistas al producto y visitas al sitio. Estos fueron los seleccionados, porque los últimos tres están directamente relacionados a las ventas como se pudo analizar en el TP1, mientrás que el primero era un feature que al contar la cantidad de checkouts de un usuario tenía una gran importancia y cuando se generó el feature *dias ultimo checkout*, este pasó a ser el más importante por una gran diferencia.

Los mencionados anteriormente, se tratan de los features relacionados con el tiempo que más peso tuvieron a la hora del cálculo de la probabilidad, pero no fueron los únicos. Otros features relacionados con el tiempo que hemos fabricado son:

- Eventos segun dia y franja horaria
- Eventos por franja horaria
- Eventos según día
- Compras por mes

La razón de estos features, se basan en conclusiones obtenidas en el análisis exploratorio:

- Los martes, miércoles y jueves son los días en los cuales se generan más ventas, a diferencia del fin de semana que tienen muy poca actividad de este tipo de eventos. Por lo tanto, supusimos que aquellos que tienen más eventos en esos días van a tener mayor probabilidad de compra.
- Luego, los meses de Abril y Mayo tienen una gran diferencia de ventas respecto a los primeros 2 meses del año. De esta manera, suponemos que una mayor actividad en estos meses puede ser indicio de probabilidad de compra. Por el contrario, una compra en los meses de Abril y Mayo nos puede indicar que el usuario ya no está interesado en la compra de un celular, salvo que sea una persona con gran cantidad de compras registradas, y una compra en los meses de Enero y Febrero o no tener registrada una compra y que tenga actividad en los meses cercanos a Junio puede aumentarla la posibilidad de comprar un nuevo teléfono celular.

- Respecto al horario, podemos inferir que una persona con actividad en Trocafone a la tarde/noche tendrá una mayor probabilidad de compra, porque estas se efectúan, mayormente, entre las 13 y 23 horas.

Para finalizar, también se realizó un hashing sobre las fechas en donde los usuarios tuvieron un evento, se hashearon todas las fechas en un array de 50 posiciones para cada usuario. Algunos de estos tuvieron una mayor importancia para la predicción deseada pero en general no funcionaron tan bien como se espero en un principio. La razón por la cual se hizo feature hashing es que la cantidad de fechas distintas que existen en el dataset es enorme, por lo que no se pueden representar cada una como una columna.

Estas son algunas de las conclusiones utilizadas del trabajo práctico anterior que nos llevaron a pensar ciertos features que pueden ser de utilidad para los algoritmos de machine learning.

3.3. Features según precio

A partir de las conclusiones del trabajo práctico anterior, donde observamos que aquellos celulares de mayor 'gama' tenían un precio superior y tenían más vistas en el producto, pero esto no se veía plasmado en las ventas del mismo. Teniendo en cuenta esto se buscaron los precios de los modelos que aparecen en el dataset en la página de Trocafone (para encontrar estos resultados de forma rápida y eficiente desarrollamos un script en python que realiza un scrapping en la pagina obteniendo dichos valores), si bien esto tuvo un resultado positivo, la aplicación de los mismos como features para los usuarios no fueron tan buenas como esperábamos.

De la página de Trocafone se obtuvieron varios datos para cada sku, de los cuales se consideraron los siguientes como features:

- cellphone_condition: condición del celular.
- enable_membership: no sabemos.
- featured: el celular aparece como promocionado.
- discount: descuento del producto.
- stock: cantidad de productos restantes.
- super_discount: booleano. El producto tiene un descuento especial.
- installments: cantidad de cuotas con la que se puede pagar el producto.
- installment_value: precio de las cuatas, si son 12.
- interest: intereses de las cuotas.
- price_new: el precio del producto nuevo.
- price_current: el precio del producto actual.

A su vez derivamos dos features de los anteriores:

- diferencia de precio: price_new - price_current.
- diferencia de precio porcentual: diferencia de precio/price_current.

Hay que tener en cuenta que estos datos podrían estar desactualizados, dado que fueron obtenidos aproximadamente 5 meses después del momento en el cual se obtuvieron los datos provistos para el trabajo práctico. Sin embargo los que más nos interesaban eran los features relacionados con los precios, y dado que la inflación en Brasil en los últimos meses fue aproximadamente de 1,5 %[6] esperamos que no haya mucha diferencia. Igualmente hay que tener en cuenta que probablemente la variabilidad de los precios será mayor por la desactualización de los equipos que por la inflación.

En la práctica el algoritmo de Random Forest le dió una buena importancia a los features 'diferencia de precio porcentual' y 'diferencia de precio', en ese orden, confirmando de cierta forma nuestra hipótesis.

3.4. Features geográficos

Aprovechando lo que realizamos en el trabajo práctico 1, obtuvimos las ciudades de cada usuario, y luego con ellas las coordenadas de dichas ciudades. El detalle de cómo se realizó esto se puede encontrar en las notebooks de features. Con estas coordenadas idealmente quisieramos calcular las distancias de todos los usuarios contra todos los usuarios (o lo que es lo mismo, todas las ciudades contra todas las ciudades en el dataset). Como esto probablemente generaría mucho ruido decidimos calcular la distancia de cada usuario a las k ciudades con más ventas.

En la práctica $k = 5$ dio un buen resultado, por lo que fue el valor que utilizamos.

4. Selección de hiper-parámetros

Para cada uno de los algoritmos que serán presentados en la sección que sigue, se realizó un estudio para ver cuales eran aquellos hiper-parámetros que hacían que cada modelo ajuste mejor a los datos, esto quiere decir que no provoquen ni overfitting ni underfitting. Se probaron dos maneras distintas para resolver este problema:

- GridSearch
- RandomSearch
- Bayesian Optimization

4.1. GridSearch

Este algoritmo prueba todas las combinaciones posibles dado un diccionario con cada hiper-parámetro y su lista de valores posibles asociada y corre un entrenamiento para cada una de ellas evaluando su resultado, en nuestro caso de estudio utilizamos una función de scoring propia que es la misma con la que se realizan las evaluaciones en Kaggle. Este algoritmo fue probado en varias ocasiones para encontrar los mejores hiper-parámentros en AdaBoost y en Random Forest pero luego de dejar el algoritmo calculando por más de 14 horas se decidió dar un paso al costado en esta búsqueda de la perfección y se paso a utilizar el método que describiremos a continuación. Solo para que quede clara la terrible performance de este método se adjuntan las especificaciones de la computadora que corrió dicho proceso.

- Procesador: intel i7 quad core 2.6 GHz (Turbo Boost de hasta 3.5 GHz)
- Ram: 16GB 2133 MHz
- Graficos: AMD Radeon Pro 450
- Disco: 256GB SSD (con particion de swap)

4.2. RandomSearch

Este fue el algoritmo que finalmente fue utilizado para entrenar los modelos ya que tuvo respuestas buenas en Score y en tiempos de ejecución. Su funcionamiento es relativamente similar al GridSearch pero difiere en una cosa fundamental, solo elige algunos valores dadas las listas de posibilidades y en base a esas "kçombinaciones muestra los hiper-parámetros con mejores resultados. En aquellos casos donde las pruebas se realizaban relativamente rápido, aumentamos la cantidad de pruebas a realizar por el algoritmo con el fin de obtener mejores resultados. RandomSearch fue utilizado para buscar cada hiper-parámetro de cada algoritmo de ML posible dados sus buenos resultados.

4.3. Bayesian Optimization

Este es un método numérico que permite encontrar máximos (o mínimos) locales en una función multivariable. Dado que podemos ver a los algoritmos de machine learning como una funcíon de sus hiperparámetros, utilizamos Bayesian Optimization para encontrar estos máximos. Una de las ventajas de este método es que no requiere calcular derivadas por lo que en la práctica resulta muy útil para los algoritmos de machine learning que tienen parámetros discretos.

La librería que utilizamos es[7], y allí también se puede encontrar una explicación más detallada de cómo funciona el método.

5. Algoritmos de Machine Learning

Los algoritmos de Machine Learning utilizados fueron:

- AdaBoost
- Decision Tree
- XGBoost
- Random Forest
- KNN
- SVM

- Perceptron
- Red Neuronal

Todos estos fueron corridos como algoritmos de regresión ya que es lo que estábamos buscando.

5.1. AdaBoost

AdaBoost es un algoritmo de Boosting, donde se intenta construir un algoritmo muy preciso a partir de Decision Trees donde el resultado final es el promedio de todos los árboles de decisión. Hasta la creación del Blending, fue el algoritmo que mejor resultado nos dio, rondando cerca del 0.871706 con los siguientes hiper parámetros:

- n-estimators: 128
- loss: linear
- learning rate: 0.07
- base-estimator: DecisionTreeRegressor con profundidad máxima de 4.

Al ser unos de los mejores algoritmos que teníamos formó parte del Blending que lo destronó y del Stacking que actualmente es el mejor ensamble que logramos formar.

Para el entrenamiento del algoritmo, se utilizaron todos los features creados y el 80 % del set de entrenamiento como se indicó en la sección de pre-procesamiento de datos.

Luego nos dedicamos a la mejora de los hiper parámetros a través de un RandomSearch con Cross Validation, y terminamos obteniendo los mencionados previamente.

5.2. Decision Tree

Simplemente se trata de un árbol de decisión, es un algoritmo muy simple que corre muy rápido y dio un resultado muy bueno cuando se lo entrenó previamente para el AdaBoost.

Se pensó que iba a ser muy útil para los ensambles y por eso se le dio importancia, cumplió con nuestras expectativas y formó parte del Blending.

No se corrió ningún tipo de optimización sobre este algoritmo ya que se utilizó el mismo árbol que formaba parte del AdaBoost. Se necesitaba un algoritmo veloz y con una buena puntuación, 0.8316, y eso fue lo que se logró. Nunca fue de los mejores puntajes pero nos resultó útil.

Algunos hiper parámetros utilizados en el DecisionTree fueron:

- criterion: mse
- max-depth: 4
- max-features: None
- min-samples-leaf: 1
- splitter: best

5.3. XGBoost

XGBoost se trata de otro algoritmo de Boosting, aunque más complejo que el AdaBoost, debido a que tiene mayor cantidad de hiper parámetros y no es tan rápido como el anterior. Se trata de unos de los algoritmos más importantes de la actualidad y por eso decidimos utilizarlo.

Al arrancar a probar este modelo, tuvimos el mejor resultado hasta ese momento pero luego fue superado por el AdaBoost.

Luego al correr un Grid Search se encontraron los hiper parámetros óptimos que están detallados a continuación:

- objective: reg:logistic
- colsamble-bylevel: 1
- colsamble-bytree: 1
- min-child-weight: 1
- learning-rate: 0.3

- max-delta-step: 5
- n-estimators': 25
- reg-lambda': 3
- max-depth': 9
- silent: True
- subsample: 1
- reg-alpha: 2
- gamma: 10

La corrida del XGBoost con estos hiper parámetros dio un puntaje de 0.87773, este es el mejor puntaje para un algoritmo individual que llegamos a obtener, pero ya fue encontrado cuando el Stacking estaba por encima de los 0.88, ayudando a mejorar el nivel del mismo.

5.4. Random Forest

El Random Forest fue de vital importancia para el desarrollo del trabajo práctico ya que nos fue indicando el feature importance y donde nos debíamos ir enfocando para ir mejorando cada vez más nuestros features.

Además de esto, siempre obtuvimos buenas predicciones llegando a un 0.8703 con los hiper parámetros señalados a continuación:

- bootstrap: True
- max-depth: 9
- max-features: 37
- min-samples-leaf: 30
- min-samples-split: 7
- n-estimators: 784

Al igual que el AdaBoost, para el entrenamiento del algoritmo, se utilizaron todos los features creados y el 80 % del set de entrenamiento como se indicó en la sección de pre-procesamiento de datos.

Al ver que obtuvo tan buenos resultados, se decidió incluir al Random Forest dentro del Blending y también fue incluido en el Stacking que es el mejor ensamble que hemos creado.

Luego nos dedicamos a la mejora de los hiper parámetros a través de un RandomSearch con Cross Validation, y terminamos obteniendo los mencionados previamente.

Se intentó correr un GridSearch, pero no terminó en buenos términos.

5.5. KNN

KNN fue unos de los primeros algoritmos probados ya que no requiere entrenamiento. Nos dio una aproximación inicial desde la cual partimos, aquellos algoritmos que den por debajo de este umbral serían descartados ya que KNN es un algoritmo súper sencillo y rápido. Probando con 161 vecinos y distancia Euclidea nos dio 0.829 como umbral mínimo (esta cantidad de vecinos y la distancia utilizadas son las que maximizaron el score en un random search).

Cuando fue notoria la velocidad de KNN para resolver el problema fue reemplazado el randomSearch por un "for"para calcular exactamente la mejor cantidad de vecinos a utilizar. Los hiper parámetros llegamos a un óptimo de 0.83219 con 1872 vecinos, distancia Euclídea y una distribución de los pesos Uniforme.

A este KNN refinado no lo hemos descartado pero aún no está siendo utilizado en ningún ensamble.

5.6. SVM con Kernel

Otro algoritmo descartado en el proceso de selección para armar los ensambles, debido a que nunca supero el umbral de los 0.8 habiendo variado los valores de penalización por clasificar mal (C) y el valor de sigma (σ)

Además se trata de un algoritmo que tarda mucho tiempo en ser entrenado para poder separar de la mejor manera, pero esto no se vio reflejado en las pocas corridas que llegamos a hacer.

5.7. Perceptrón

Antes de implementar una red neuronal, la cual será detallada en la siguiente subsección, usamos un perceptrón y variamos el learning rate para buscar el resultado más alto con nuestra función de scoring sin hacer una subida a Kaggle. Su mejor score fue de 0.75. Por lo tanto pensamos en implementar un multilayer regresor es decir una red neuronal para ver si podíamos mejorar la respuesta de nuestro programa.

5.8. Red Neuronal

Se intentó implementar una red neuronal para obtener las predicciones pero nunca conseguimos un buen resultado, el mejor fue aproximadamente 0,80 con una red neuronal de 50 capas con 100 neuronas cada una. Las otras redes dieron siempre por debajo del 0.8.

Al estar tan lejos de los mejores algoritmos, decidimos desestimar el uso de las redes neuronales para nuestros ensambles. Además de que es un algoritmo que al aumentar la cantidad de capas ocultas o la cantidad de neuronas que éstas tengan varia mucho su perfomance llegando a tardar mas de una hora en entrenar obteniendo resultados muy bajos, en este caso en particular sospechamos que el modelo se sobreajustó a los datos, es decir que caímos en un problema de overfitting.

6. Ensambles

Luego de probar los modelos mencionados anteriormente, en vez de quedarnos con el que nos dio el mejor resultado decidimos hacer un ensamble. Para ello consideramos los siguientes ensambles:

- Majority Voting
- Averaging
- Stacking
- Blending

6.1. Majority Voting

Resumidamente este ensamble se trata de elegir como predicción final el resultado más frecuente entre los modelos (moda). Este tipo de ensamble tiene sentido cuando la predicción es directamente la clase. Si la predicción es la probabilidad de cada clase entonces otros métodos funcionan mejor[1]. Por esto y dado que estamos tratando con un problema de regresión, se decidió descartar este método .

6.2. Averaging

Esta forma de combinar predictores es muy práctica para los problemas de regresión, ya que podemos simplemente promediar los resultados de los distintos modelos. Este es claramente un buen ejemplo de *bagging* y la gran ventaja de este método es que no se necesita de un set de entrenamiento.

Debido a la simpleza de este modelo se prefirió utilizar, en principio, otros con una complejidad un poco mayor que permitieran obtener un resultado más preciso. Los mismos fueron Stacking y Blending. Sin embargo para combinar los resultados de estos dos sí se utilizó Averaging, justamente por lo sencillo que resulta realizar la combinación.

6.3. Stacking

Podemos encontrar una explicación detallada de como funciona este modelo en [2]. Resumidamente tanto Stacking como Blending buscan utilizar los resultados de distintos modelos como features (llamados metafeatures) y luego aplicar uno o varios algoritmos de machine learning para obtener un mejor resultado final. En el peor de los casos estos algoritmos nos darán una mejora sobre el peor de los algoritmos [3].

En particular Stacking realiza *cross validation* para encontrar los metafeatures y también se pueden utilizar features originales. La idea del modelo de stacking es que sea relativamente rápido y que no tenga demasiada complejidad, por lo que resulta conveniente sólo agregar los features más significativos. Como criterio para elegirlos se utilizaron los features a los cuales Random Forest les dio más importancia.

Sin embargo este método filtra en una pequeña medida los labels. Esto ocurre debido a que al realizar *cross validation* para encontrar los metafeatures los labels se encuentran, en parte, embebidos en estos features. Dicha filtración resulta muy pequeña y se encuentra claramente explicada en [2].

De todas formas Blending soluciona el problema de la filtración y es por ello que también fue utilizado para realizar un ensamble. Por otro lado podemos encontrar una idea de como implementar Stacking en [4].

6.4. Blending

Los detalles de este método se encuentran explicados en [1] por lo que no vamos a explayarnos en su funcionamiento. Resumidamente se trata de utilizar el 90 % del set de entrenamiento para entrenar los modelos base, y de esa forma predecir los labels del 10 % restante. Luego se entrena un modelo Blending que utiliza dichas predicciones como features (metafeatures).

Este método tiene los siguientes beneficios:

- En su versión sencilla es más simple que Stacking.
- No hay filtración dado que los modelos base y el Blending usan distintos datos.

Los contras son:

- Se utilizan muchos menos datos para entrenar el Blending (10 %).
- El modelo final es más propenso a *overfitting*.[3]

En la práctica este ensamble nos dio muy buenos resultados, por lo que finalmente lo utilizamos promediado con Stacking.

7. Conclusiones

Luego de estar semanas trabajando en nuestro algoritmo de Machine Learning, confirmamos que la unión hace la fuerza porque los ensambles ya mencionados fueron los algoritmos que le dieron un salto de calidad a nuestra performance. Pero es importante destacar que aquellos algoritmos que forman parte de algún ensamble, deben tener un puntaje significativo y que le den información valiosa al Blending y al Stacking.

Otra cosa que pudimos observar es que la búsqueda de los hiper parámetros es una labor importante y que puede llevar a algún algoritmo a mejorar casi un 2 %. Una de las desventajas que le observamos a esto es que si se modificaron features, estos hiper parámetros ya encontrados suelen modificar y no resultar tan útiles como antes. Por este motivo, se deben correr repetidamente los Random Search probando diferentes hiper parámetros. Pero sabemos que lo óptimo seria hacer un GridSearch en torno a valores que anteriormente fueron muy potentes, el gran problema de esto es la performance ya que cuando fue intentado uno de los algoritmos tardo mas de 14hs para correr.

Por último, queremos mencionar que los algoritmos de AdaBoost y de Random Forest fueron aquellos que mejor puntaje dieron respecto al tiempo de aprendizaje que tardaron, ambos estuvieron alrededor del 0.87. Esto nos permitió tener ensambles rápidos y de muy altos puntajes que nos permitieron correr optimizaciones sobre ellos.

En contrapartida, el XGBoost es un algoritmo que obtuvo un puntaje similar a los anteriores pero conlleva un entrenamiento más exhaustivo y los hiper parámetros en XGBoost juegan un rol muy importante que modifican sustancialmente su puntaje, cosa que no se vio tan reflejada en los algoritmos anteriores.

Referencias

[1] Luis Argerich, Natalia Golmar, Damián Martinelli, Martín Ramos Mejía, Juan Andrés Laura, *75.06, 95.58 Organización de Datos, Apunte del Curso*, Universidad de Buenos Aires Facultad de Ingeniería, v2.0, 2018.

[2] Ben Gorman, *A Kaggler's Guide to Model Stacking in Practice*, http://blog.kaggle.com/2016/12/27/a-kagglers-guide-to-model-stacking-in-practice, 12.27.2016.

[3] ML Wave, https://mlwave.com/kaggle-ensembling-guide, Kaggle ensembling guide, 11.6.2015.

[4] Emanuele Olivetti, https://github.com/emanuele/kaggle_pbr/blob/master/blend.py, 19.7.2016.

[5] https://scikit-learn.org/stable/modules/svm.html#svm

[6] Global Rates, https://es.global-rates.com/estadisticas-economicas/inflacion/indice-de-precios-al-consumo/ipc/brasil.aspx, Inflación Brasil - índice de precios al consumo (IPC), 30.11.2018.

[7] BayesianOptimization, https://github.com/fmfn/BayesianOptimization.