Contents

1 Procesamiento y análisis de datos

En esta sección se introduce brevemente el producto a analizar y las herramientas que se utilizaron para realizar el análisis y la predicción requerida.

1.1 Datos utilizados

Se estudiaron datos provistos por la empresa Trocafone, analizando un conjunto de eventos de web analytics de usuarios que visitaron www.trocafone.com, su plataforma de e-commerce de Brasil.

1.2 Sobre la empresa

Trocafone es un side to side Marketplace para la compra y venta de dispositivos electrónicos que se encuentra actualmente operando en Brasil y Argentina.

La empresa realiza distintas actividades que van desde la implementación de plataformas de trade-in (conocidos en la Argentina como Plan Canje), logística directa y reversa, reparación y recertificación de dispositivos (refurbishing) y venta de productos recertificados por múltiples canales (ecommerce, marketplace y tiendas físicas).

Para conocer más de su modelo de negocio, pueden visitar el siguiente artículo:
https://medium.com/trocafone/el-maravilloso-mundo-de-trocafone-5bdc5761856b

1.3 Lenguajes y librerías utilizados

- Se utlizó como lenguaje de programación **Python3**.
- Para las visualizaciones, se utilizaron las librerías **MatPlotLib** y **Seaborn**.
- Como editor se utilizó **Jupyter Lab** (o **Jupyter Notebook**)
- Para el manejo de DataFrames, se eligió **Pandas** como librería a utilizar.
- Se utilizaron algunas herramientas como std o argsort de la librería **Numpy**. calendar

- Se importaron diferentes métricas como "accuracy score", "f1 score", "precision score", "recall score", "roc auc score" de la librería **sklearn**.
- Se utilizaron diferentes busquedas de hiperparametros, entre ellas "GridSearch" y "RandomizedSearchCV" de **sklearn**.
- Para poder realizar cross-validation, se importó "StratifiedKFold" de **sklearn**.
- Se realizó Clustering mediante la funcion "KMeans" de la librería **sklearn**.
- Para la entrega final, los siguientes clasificadores fueron importados: "xgboost", "lightgbm", "RandomForestClassifier", "CatBoostClassifier".
- Se utilizo como ensamble de clasificadores a "VotingClassifier".

1.4 Repositorio de Github

Para el trabajo en conjunto del equipo, se utilizo un repositorio en github, donde se encuentran todos los archivos necesarios del análisis y predicciones y este informe propiamente dicho.

Link: https://github.com/emabrea/7506-DATOS-TP2.git

2 Breve análisis del set de Datos

2.1 Balance del set de datos

Al haber muchas más visitas al sitio que compras de productos, el set de datos resulta ser muy desproporcionado, encontrándose muchas mas labels (95%) con 0 (no compra) que 1 (compra). Esto es un gran problema para ciertos algoritmos de machine learning, que fallan con set de datos desbalanceados.

Las soluciones a este problema son acotadas, ya que se dispone de un set de datos chico. Con la librería **imbalanced-learn** probamos distintas soluciones con oversampling, undersampling o combinaciones de ambos, pero no resultaron. Esto era lo esperado, ya que hacer oversampling significaría inventar la actividad de clientes, mientras que haciendo undersampling quedan muy pocos datos.

El problema se solucionó utilizando los parametros de los algoritmos de clasificación que permiten darle distinto peso a cada label. En particular, encontramos que la mejor relacion de pesos era de 1:7.

2.2 Métrica usada

Al tener el set de datos con 95% de una clase, no se puede usar como métrica la precision, pues un algoritmo que solo arroje labels de esa clase, tendría un 95% de precisión, a pesar de no predecir nada. Es por esto que se decidió utilizar otra métrica. En este caso, se utilizo el ROC AUC SCORE (Area Under Receiver Operating Characteristic Curve). En dicha métrica, se toman en cuenta los falsos positivos y negativos, y los verdaderos positivos y negativos. Con un random guess, se obtendría un 0.5, y con un puntaje perfecto, un 1.0. En la competencia, los valores altos se mantuvieron entre 0.85 y 0.88. Creemos que por la naturaleza del problema, sería muy difícil superar esa cota, pues puede haber dos usuarios con los mismos eventos, y uno compró y el otro no.

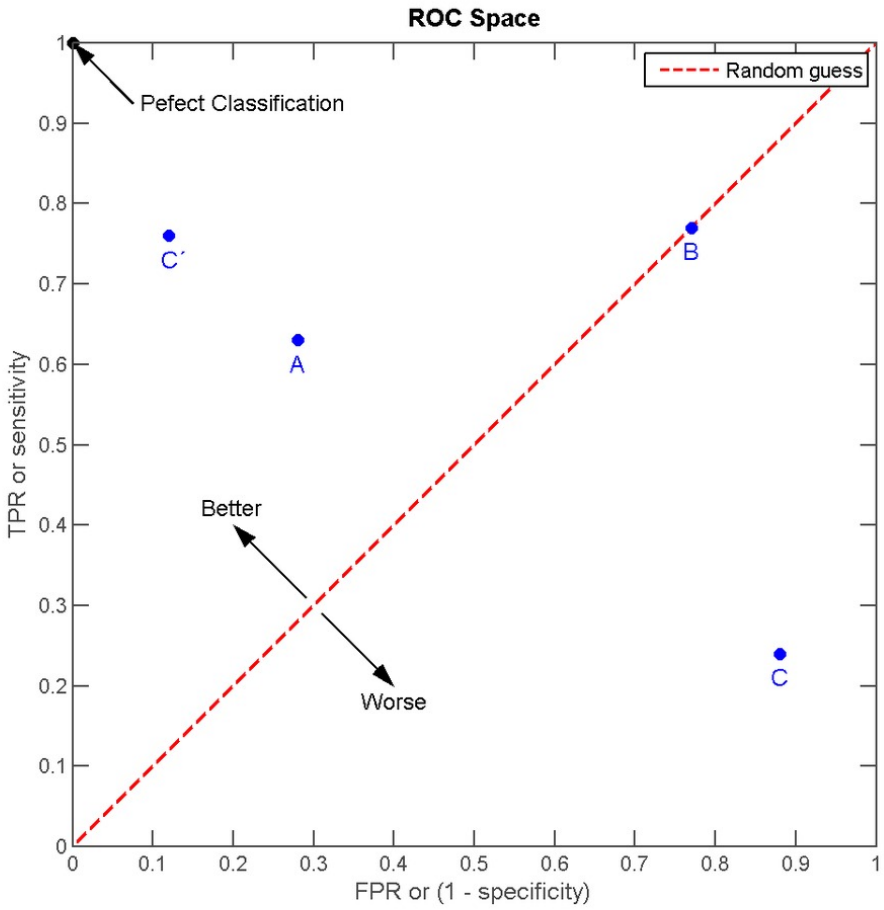

Figure 1: Métrica usada

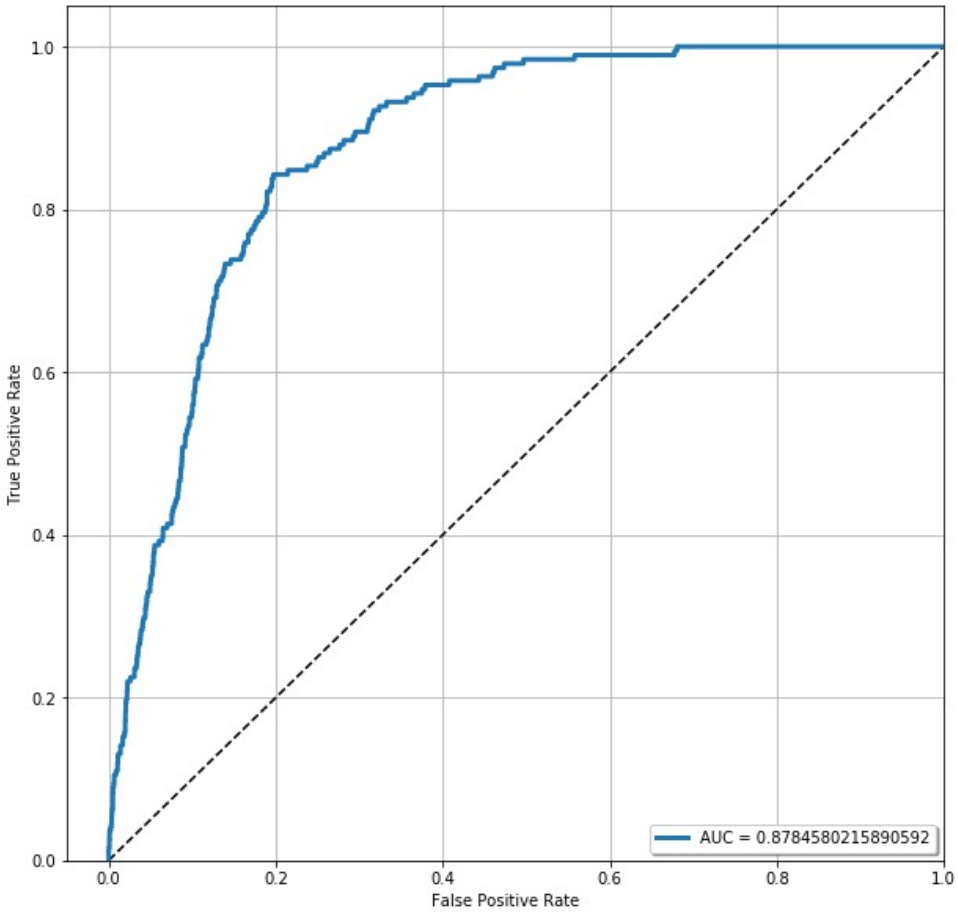

Figure 2: Curva obtenida

3 Featuring Engineering

3.1 Encoding

El dataset contenía varias variables categóricas, las cuales no son aceptadas por múltiples algoritmos de clasificación como por ejemplo XGBoost.

Otro problema es que a cada usuario se le debe asignar una sola muestra (fila), por lo tanto, surge el problema de que seleccionar luego de hacer el encoding. Por ejemplo, se puede codificar el almacenamiento de los modelos vistos, pero luego hay que asignarle uno solo a cada usuario. Es por esto que para este problema, encoding no fue muy útil.

Para poder utilizar las variables categóricas, se utilizaron los siguientes algoritmos:

3.1.1 Count Encoding

Consiste en reemplazar las categorías por la cantidad de apariciones de las mismas en el set de datos. Resultó la codificación más simple de implementar y fue la primera que utilizamos.

3.1.2 One Hot Encoding

Consiste en dividir el atributo en tantas columnas como valores posibles puede tener y usar cada columna como un dato binario indicando si el atributo toma o no dicho valor. One-hot encoding es una codificación muy popular pero lamentablemente no escala muy bien. En primer lugar es necesario construir en memoria un diccionario por cada feature en donde asociaremos un número a cada posible valor de cada columna, este diccionario puede ser demasiado grande. Otro problema serio es que si una columna puede tomar muchos valores diferentes posibles, por ejemplo la columna **model**, con 208 representantes diferentes, entonces necesitamos cientos de nuevas columnas para representar los datos, esto hacía que ciertas a ciertas columnas ni siquiera se le intente aplicar dicho método ya que no es viable.

3.1.3 Mean Encoding

Esta codificación consiste en reemplazar las categorías con el promedio de los valores a predecir en la categoría. Esto implica que por cada categoría, se calcula el promedio entre las postulaciones y no postulaciones y se reemplaza la categoría por ese promedio. Esto se utilizo con los precios de los modelos, aplicandole el precio promedio a dichos eventos sin modelos (NaN).

3.2 Featuring Engineering primera entrega (2º puesto)

3.2.1 Primeros features

Para el primer submit, se utilizaron unos 15 features, en su mayoria referidos al tiempo, como cantidad de visitas/ checkouts/ compras en los últimos X días. Con este sencillo enfoque, se obtuvo un puntaje de 0.85289 . Para las siguientes entregas, se uso el 100% del set de datos, con más features, y mejores hiper-parametros.

3.3 Featuring Engineering segunda entrega (Train 100%)

3.3.1 Features sobre acciones por rango de tiempo

3.3.2 Acciones en el último mes

- Visitas último mes
- Checkouts último mes
- Compras último mes
- Subscripciones último mes

3.3.3 Acciones en los últimos 15 días

- Visitas últimos 15

3.3.4 Acciones en la última semana

- Visitas última semana
- Checkouts última semana
- Compras última semana
- Campaña ultima semana

3.3.5 Acciones en los últimos 3 días

- Visitas últimos 3

3.3.6 Features de acciones del usuario

- Total visitas usuario
- Total checkout
- Total compras
- Búsqueda celular
- Días distintos
- última visita

3.3.7 Features de modelos

- Modelos distintos vistos

3.4 Featuring Engineering tercera entrega

Mejores hiper parametros

3.5 Featuring Engineering cuarta entrega

3.6 Cantidad de valores diferentes por característica

- horas distintas
- dias distintos ultima semana
- dias distintos de la semana
- dias distintos de la semana last month
- marcas distintas
- celular mas visto * (hablar de precios de celulares)
- compras por dia semana
- modelos por color
- eventos distintos

3.7 Featuring Engineering quinta entrega

En la quinta entrega se decidió ir mas alla del set de datos provisto por el enunciado, debido a que se noto la ausencia de precios en los modelos de los equipos, es por eso que investigando en el sitio web de Trocafone, se decidió incluir los precios de los modelos más populares.

A continuación se muestra la lista de precios incluidos para entrenar el algoritmo: Dicho esto, se decidió incluir los siguientes features representativos sobre el precio de los equipos en los eventos de cada usuario:

Model	Capacidad(GB)	Precio
iPhone 5s	32	939
iPhone 5s	16	619
iPhone 6	16	949
iPhone 6	128	1499
iPhone 6	64	1069
iPhone 6 Plus	64	1769
iPhone 6 Plus	16	1589
iPhone 6S	64	1939
iPhone 6S	32	1679
iPhone 6S	16	1439
iPhone 6S Plus	128	2199
iPhone 6S Plus	64	2089
iPhone 6S Plus	16	1959
iPhone 7	32	2229
iPhone 7	128	2469
iPhone 7	256	2819
iPhone 7 Plus	128	3009
iPhone 7 Plus	256	3559
iPhone 7 Plus	32	2869
iPhone 8 Plus	64	4000
iPhone SE	64	1309
iPhone SE	16	999
iPhone X	64	5000

Tabla de celulares Iphone

Model	Capacidad (GB)	Precio
LG G4 H818P	32	829
Motorola Moto X Style	32	950
Motorola Moto X2	16	569
Motorola Moto X2	32	619
Samsung Galaxy A7 2017	32	849
Samsung Galaxy A9 Pro 2016	32	1036
Samsung Galaxy J5	16	399
Samsung Galaxy J7 Prime	32	730
Samsung Galaxy S6 Edge Plus	64	2099
Samsung Galaxy S6 Flat	32	849
Samsung Galaxy S6 Flat	32	669
Samsung Galaxy S6 Flat	32	669
Samsung Galaxy S7	32	1089
Samsung Galaxy S7 Edge	32	1299
Samsung Galaxy S7 Edge	128	1479
Samsung Galaxy S8	64	1799
Samsung Galaxy S6 Edge	64	1039
Samsung Galaxy S6 Edge	32	900
iPhone 5c	8	450
Samsung Galaxy S8 Plus	64	1859
iPhone 5	16	550
Samsung Galaxy A5 2017	32	689
iPhone 4S	8	319
Samsung Galaxy J7	16	550
Motorola Moto X Play 4G Dual	4	599
Samsung Galaxy S5	16	439
Motorola Moto G3 4G	16	379

Lista de Moviles con su almacenamiento y precio correspondiente.

De la lista de precios de moviles se pudo extraer algunas características para ser tomadas como features:

- Precio máximo
- Precio mínimo
- Desviación estandar de los precios
- Media de los precios

Con esta información, se pudo mejorar levemente el score, pero debido a la gran cantidad de NaN en la columna "Model", la mejora no fue la esperada.

3.8 Featuring Engineering sexta entrega

Se pudo observar en Select K-Best y en los feature importance de cada algoritmo que los features relacionados a Visitas, Checkouts y Conversion tenían un papel relevante a la hora de realizar predicciones, es por eso que se decidió intensificar la cantidad de features en dichas areas. Esto dió como resultado la inclusión de los siguientes:

- visitas último día
- checkouts últimos 15 días
- checkouts últimos 3 días
- checkouts último día
- compras ultimos 3 días
- compras ultimos 15 días
- compras ultimo día

Sin embargo, para los submits finales, se usaron todos los features, pues observamos que aunque un feature no parezca importante para un algoritmo, eso no implica que quitandolo se mejore el resultado.

3.9 Featuring Engineering septima entrega

Siguiendo la corriente de pensamiento que nos llevó a la sexta entrega, se decidió reforzar los features con el agregado de los siguientes:

- visitas últimos 2 días
- checkouts últimos 2 días
- checkouts últimos 4 días
- compras últimos 2 días
- compras últimos 5 días
- compras últimos 4 días

Es decir, se profundizo en características temporales, y haciendo varias segregaciones, se logró mejorar considerablemente el modelo, superando los 0.86. También se observo que el tipo de evento más "importante" para los algoritmos resulto ser el "checkout".

4 Algoritmos de Clustering

Luego de la clase sobre clustering, propusimos agregar como features los resultados de aplicar distintos algoritmos de clustering al set de datos. Se probó con K-means, K-means++, DBScan y HDBScan. Por prueba y error se fueron determinando los hiper parámetros k más convenientes. Tanto K-means como DBScan fueron posteriormente descartados ya que se notó que empeoraban los resultados.

Una posible interpretación de estos features es que estos algoritmos de clustering lograron identificar distintos grupos o sectores entre los visitantes al sitio, que ayudan a determinar si es probable o no que compren.

Features agregados:

- k-means++, k=2
- k-means++, k=3
- k-means++, k=4
- k-means++, k=5
- k-means++, k=50
- k-means++, k=100
- HDBScan, k=100

5 Clasificadores utilizados

En la ultima entrega se utilizó un Voting Classifier de cuatro algoritmos:

- XGBoost
- Random Forest
- LightGBM
- CatBoost

Sin embargo, esta elección provino de un arduo proceso de elecciones previas y combinaciones de algoritmos a ensamblar, a continuación listamos algunos de los algoritmos utilizados y la motivación para ir reemplazando las diferentes combinaciones por otras.

- KNN

  Fue el primer algoritmo utilizado por su sencillez, pero no se pudo obtener más de 0.79 de score (local). Un problema es no poder indicarle el peso de las clases.

- Decision tree

  Primer algoritmo basado en arboles utilizado, pero debido a la naturaleza del problema (predicción de compras), no obtuvo buenos resultados.

- Logistic regression

  A pesar de que es posible indicarle el peso de las clases, este algoritmo fallaba en encontrar los labels positivos. Se probaron varios valores de C, y del solver (newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga'), pero no se pudo superar los 0.80.

- SVM

  SVM acertaba muchos labels 0, pero muy pocos 1, por lo cual tenía muy buena precisión, pero un ROC AUC Score muy bajo. Se probaron varios kernel con distintos C, pero no resultó.

- Neural network

  A este algoritmo se le dedicó mucho tiempo, con la esperanza de superar el puntaje obtenido. Para ello se normalizo (con varios métodos) todo el dataframe, y se trabajó con datos numéricos. Se probaron diversos

solvers ('lbfgs', 'sgd', 'adam'), obteniendo mejores resultados con lbfgs. Para la funcion de activación, se probaron: 'identity', 'logistic', 'tanh', 'relu', resultando mejor la funcion "relu". Luego, se probaron diversos valores de alpha, y de learning rate, como también varias capas (layers), entre 2 y 4. Esto aumento el tiempo del algoritmo, cuyo mejor resultado fue 0.8468. Al hacer un ensamblador con otro algoritmo del tipo XGBoost, se obtuvieron buenos resultados, pero finalmente se decidió no incluirlo en la entrega final, entre otros motivos, por el tiempo tomado para entrenar.

- AdaBoost

  Este algoritmo dio buenos resultados, pero resulto ser un poco inferior a XGBoost, por lo cual, decidimos no incluirlo en el resultado final

  Como dato adicional, también se probo el famoso algoritmos Naive Bayes, pero como era de esperarse, los resultados no fueron buenos.

6 Tuning

En los diferentes algoritmos vamos a llamar parámetros a aquellos valores que el algoritmo encuentra a partir de los datos y vamos a llamar hiper-parámetros a aquellos datos que el algoritmo necesita para poder funcionar.

Llamaremos óptimos a los hiper-parámetros que logren para un set de Datos maximizar determinada métrica, en este caso, utilizamos la métrica ROC AUC.

Para encontrar los hiper-parámetros óptimos para un algoritmo pueden usarse dos métodos: Grid-Search o Random-Search.

6.1 Grid-Search

En un Grid-Search probamos todas las combinaciones posibles dentro de una lista de valores posibles para cada hiper-parámetro.

En este caso, debido a la cantidad de hiper-parámetros , se decidió comenzar en las listas de valores posibles para cada uno con un "paso" grueso.

Esto reduce en un principio la cantidad de combinaciones a ejecutar. Luego se podra refinar la busqueda en la zona donde resultaron optimos nuestros hiper-parametros iniciales.

Este proceso fue especialmente necesario cuando los hiper-parámetros tomaban valores reales.

Cuando la cantidad de hiper-parámetros es realmente muy grande la combinatoria a realizar puede resultar muy ineficiente, en estos casos puede recurrirse al método de Random-Search.

6.2 Random-Search

Como la cantidad de hiperparametros era elevada, posteriormente se recurrió al método de Random-Search,

En este método controlamos cuantas iteraciones realizamos de nuestro algoritmo y por cada iteración seleccionamos los valores de los hiper-parámetros al azar dentro de un rango preestablecido.

Cabe aclarar que este método no es tan preciso como un grid-search pero es mucho mas rápido, pudiendo invertir mayor parte del tiempo a la creación de Features.

6.3 Aplicación a nuestro algoritmo

En uno de nuestros primeros ensambles (constituídos por un Random Forest y un XGBoost) se aplico el método de Random Search con el fin de poder encontrar (u obtener una buena aproximación a ellos) de los hiperparametros:

Para XGBoost:

- scale pos weight (tipicamente: entre 3 y 9).
- learning rate (tipicamente: 0.001, 0.01, 0.05, 0.1)
- max depth (entre 1 y 6)
- n estimators (tipicamente: 500, 1000, 1500, 3000, 5000)

Para Random Forest:

- Peso de una de las clase 1 : La lista de este hiperparametro contenía los números del 3 al 10
- Criterio: Este hiper-parámetro podía ser : {'gini', 'entropy'}

  Y en general, para los demás algoritmos, también se probaron los mismos hiper-parametros que xgboost, como cantidad de árboles, profundidad, gamma, etc.

Es muy importante aclarar que para la validación de hiper-parámetros se utilizó K-fold cross validation, con K= 10.

Es decir, de nuestro set de datos, el 10% fue usado para validar los hiperparametros cada vez.

7 Ensamble

Los mejores resultados en ML suelen surgir de la combinación de varios algoritmos. Es muy raro que un solo algoritmo de ML logre mejores resultados que un ensamble. Esto pudo verse en las sucesivas entregas, donde utilizar un ensamble nos traía resultados mas satisfactorios que usar los algoritmos individualmente.

7.1 Combinando Algoritmos Diferentes

Estos procesos suelen ser la clave para obtener un mejor resultado, ya que permitieron aprovechar el poder expresivo de varios modelos completamente diferentes para obtener un resultado común. En nuestro caso, el problema a abordar era de clasificación.

7.1.1 Majority Voting

Tenemos varios clasificadores distintos para un cierto problema, cada uno de ellos produce un resultado y queremos obtener un resultado final. Una aproximación simple es ver cual es la clase que tiene mayoría entre todos los clasificadores.

En un primer momento, se decidió utilizar este tipo de ensamble debido a su simplicidad, pero nos encontramos que el mismo tiene sentido cuando la predicción es directamente la clase. Si la predicción es la probabilidad de cada clase entonces obtuvimos que otros métodos funcionan mejor.

Cuando la correlación entre los modelos es baja el resultado del ensamble, en general, mejora notablemente el resultado de cada modelo individual.

Una primera conclusión es que dados muchos clasificadores es conveniente elegir un conjunto que tenga buenos resultados y que estén muy poco correlacionados.

7.1.2 Averaging

Promediar el resultado de varios clasificadores es un método muy popular que funciona en muchos problemas distintos: regresión, clasificación (ya sea para predecir clases o probabilidades), etc. La idea principal es reducir el overfitting.

En general, una separación suave entre las clases es mejor que una separación muy irregular y promediar clasificadores logra esto.

Cuando promediamos clasificadores que predicen la probabilidad de las clases, hubo que prestar especial atención porque cada clasificador individual puede tener una calibración completamente diferente.

Uno puede dar probabilidades muy cercanas a 1s y 0s mientras que otro, a lo mejor, se mantiene dentro de un cierto rango. Una solución para esto es convertir cada probabilidad de un rango entre 1 y n, siendo n el total de personas a predecir. La persona con mayor probabilidad tiene 1, el segundo 2, etc, y el de menor probabilidad, n, sin importar el valor de las probabilidades. Si hacemos esto para todos los clasificadores podemos luego promediar los rangos y convertir estos promedios en un número entre 0 y 1 para la probabilidad final.

8 Últimas entregas

69 labels

References

[1] Trocafone website, www.trocafone.com.

[2] NumPy - NumPy, http://www.numpy.org/.

[3] Python Data Analysis Library, https://pandas.pydata.org/.

[4] Matplotlib: Python plotting — Matplotlib 3.0.0 documentation, matplotlib.org.

[5] XGBoost documentation,
https://xgboost.readthedocs.io/en/latest/parameter.html

[6] Random Forest documentation,
https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

[7] Light GBM documentation,
https://github.com/Microsoft/LightGBM/blob/master/docs/Parameters.rst

[8] CatBoost documentation,
https://github.com/catboost/catboost

[9] Voting Classifier documentation,
http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html#sklearn.ensemble.VotingClassifier

[10] HDBScan documentation,
https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

[11] Select K best features documentation,
https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html

[12] KNN documentation,
https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[13] Decision tree documentation,

http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html

[14] Logistic regression documentation,

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

[15] SVM documentation,

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

[16] Neural network documentation,

https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html