Trabajo Práctico 2 - Conversión de usuarios en MercadoLibre

Inteligencia Artificial - Licenciatura en Tecnología Digital - UTDT

Zoe Borrone* Luca Mazzarello† Ignacio Pardo‡

8 de Octubre de 2023

Abstract

Link al repositorio: https://github.com/IgnacioPardo/TP2_TDVI

Contents

*zoeborrone@gmail.com | |
†lmazzarello@mail.utdt.edu | 44512364 | 21L720
‡ipardo@mail.utdt.edu | 43243336 | 21R1160

0.1 Análisis Exploratorio de los Atributos

Nuestro primera idea para conocer los datos, mas allá de leer el .csv, fue estudiar la correlación y covarianza entre los atributos del dataset y la variable a predecir.

En el siguiente gráfico podemos ver la correlación entre la conversión y cada uno de los atributos, inclusive con los que nuevos atributos que creamos que se explican mas adelante. Con esto pudimos tener una aproximacion tentativa a la importancia de los atributos previo a entrenar los modelos. Luego buscamos ver como eran las distribuciónes de algunos de estos aatributos con mas correlación. Tanto el offset como la posición en pantalla en la que esta el producto tienen bastante importancia por sobre la conversión de un usuario. También observamos como a menor cantidad de visitas de un producto es menor la proporción con la que ocurre la conversión, pero a medida que crece la cantidad de visitas, la proporción se equipara. Nos ideamos estos puntos de inflexión podrían ser posibles cortes para los arboles.

También nos interesó ver como afectaba a la conversión desde que dispositivo ingresaba el usuario a MercadoLibre, en el siguiente gráfico podemos observar como, en proporción, aquellos que entran desde la web convierten mas que aquellos que entran desde otra plataforma. Además, en nuestra exploración del dataset observamos que había columnas que no tenían variación de los datos, por ejemplo, `accepts_mercadopago`, ya que todos los productos aceptaban mercado pago, o la columna `boosted`, ya que ninguna estaba "boosteada".

También en un inicio nos llamó la atención la columna `main_picture` que nos pareció podía llegar a tener impacto sobre la conversión, rapidamente nos dimos cuenta que no estaban disponibles las imágenes, pero al analizar la columna de `tags` en el desarrollo encontramos que unos tags posibles eran "good_quality_picture" y "good_quality_thumbnail" contra "poor_quality_picture" y "poor_quality_thumbnail".

0.2 Creación de nuevas dimensiones

A partir de la exploración de los atributos, las nuevas columnas que agregamos en nuestro trabajo fueron:

0.2.1 Combinación de atributos existentes

price_diff: Esta columna calcula la diferencia de precio absoluta entre las columnas "price" y "original_price". discount: Calcula el porcentaje de descuento para cada artículo comparando la diferencia entre "original_price" y "price" sobre "original_price". price_ratio: Calcula la proporción de precio dividiendo "price" sobre "original_price". is_discount: Una columna binaria que indica si hay un descuento (1) o no (0) basándose en el valor de la columna "discount". title_length: Longitud en caracteres de la columna "title" para cada artículo. title_word_count: Cantidad de palabras en la columna "title" dividiendo el texto por espacios. title_length_word_count: La proporción entre "title_length" y "title_word_count", lo que puede ayudar a identificar títulos con diferentes niveles de verbosidad. domain_dominance: Cuán dominante es el artículo dentro de su dominio dividiendo "sold_quantity" entre "qty_items_dom".

Estas columnas creadas son derivadas de columnas ya existentes en el dataset original. Estas pueden ser útiles para analizar y categorizar artículos en función de su precio, descuento, longitud del título, patrones de visita y dominancia en el dominio.

0.2.1.1 Polynomial Features Por cada atributo especificado en la lista `poly_attrs = ["print_position", "offset", "discount", "price", "health", "original_price"]`, se crea una nueva columna elevando al cuadrado los valores de ese atributo. Los nombres de las columnas siguen el patrón 2 (ej. `price2`).

A su vez, para cada par de atributos especificados en la lista de `poly_attrs`, se crea tambien la suma de cuadrados dos atributos (por ejemplo, "print_position2 + offset2").

Estas nuevas caracteristicas pueden capturar relaciones no lineales entre los atributos.

0.2.2 Encoding de atributos categóricos

Además, aplicamos Label Encoding a las columnas "platform", "product_id", "domain_id" y "logistic_type" para convertir los datos categóricos en valores numéricos en nuevas columnas y One Hot Encoding (OHE)

a la columna “category_id” para crear representaciones binarias de los datos de categoría. Para aliviar la demanda memoria RAM, se decidió hacer OHE por sobre las 10 categorías más populares y el resto simplemente contaba con todos 0.

Para las columnas “free_shipping” y “fulfillment” de tipo Booleanas las encodeamos a 1 y 0, y para la columna “listing_type_id” se modifica si el valor es “gold_special,” se establece en 0; de lo contrario, se establece en 1. En retrospectiva quizas podríamos haber invertido los valores de 0 y 1, ya que el tipo de listado “gold_special” parece de mayor importancia, pero en su momento consideramos que XGBoost iba a poder discernir en como manejar esta columna y no se lo volvió a considerar hasta el momento de escribir este informe.

0.2.3 Imputación de valores faltantes

imp_is_pdp (Indicador Binario): Esta columna fue creada para indicar si la columna “is_pdp” tiene valores faltantes (NaN). Si “is_pdp” tiene un NaN, esta nueva columna se establecerá en 1; de lo contrario, se establecerá en 0. Además, los valores faltantes en “is_pdp” se llenan con 0 y luego se convierten a números enteros. Al crear esta columna se noto el primer “salto” en performance, ya que la columna is_pdp si es Falsa tiene correlación casi total con la columna a predecir ya que el usuario no pudo haber comprado el producto sin entrar a su página de detalles. Al notar la importancia del dato, una cuestión que se implemento a lo largo de gran parte del desarrollo fue imputar los missings con predicciones de un modelo XGBoost entrenado con los datos que no tenían missings de los datos de Train. Sin embargo el modelo final no contaba con este tipo de imputación.

En todos los modelos que entrenamos, la columna “is_pdp” fue la más importante, su ganancia rondaba siempre el rango 700-900, mientras que el resto de los atributos

imp_user_id (Indicador Binario): Al igual que con “imp_is_pdp”, esta columna se crea para indicar si la columna “user_id” tenía valores faltantes. Si “user_id” tiene valores faltantes, esta columna se establecerá en 1; de lo contrario, se establecerá en 0. Los valores faltantes en “user_id” se llenan con 0.

0.2.4 Columna “tags”

Observamos que los Tags del producto podían contener información interesante sobre los productos por lo que para cada etiqueta unica en la columna “tags”, se crea una nueva columna con el nombre de la etiqueta (como OHE). Estas nuevas columnas tienen valores booleanos (1 si el artículo contiene la etiqueta, 0 en caso contrario). Los nombres de estas columnas se basan en las etiquetas únicas encontradas en el conjunto de datos entero.

Luego en otra columna llamada “tag_counts” se almacena la cantidad de etiquetas asociadas a cada artículo.

Algunos de estos tags tuvieron bastante importancia sobre el modelo final, lo cual esperabamos cuando observamos su gran correlación con la variable conversión. Algunas destacadas son “deal_of_the_day”, “ahora-12”, “lightning_deal” y “extended_warranty_eligible”. Esta última nos impulso a parsear la columna “warranty”.

0.2.5 Garantías

Observamos que la columna “warranty” no estaba en un formato consistente. Algunos artículos tenían “Sin garantía” en el texto, mientras que otros tenían “Sin garantía” o “Sin garantía”, probablemente por ser textos que fueron ingresados manualmente. Además, algunos artículos tenían “Vendedor” o “Fábrica” en el texto de la garantía, mientras que otros no. Para volver los datos mas uniformes llevamos a cabo un preprocesamiento de la columna “warranty” original, modificandola convirtiendo todo el texto a minuscula y removiendo las tildes para luego crear nuevas variables representando distintosa aspectos de la garantia de cada artículo:

warranty_saler: Esta columna se crea como una variable booleana (0 o 1) que indica si el texto de “warranty” contiene la palabra “vendedor” (garantía del vendedor). warranty_factory: Similar a “warranty_saler”, esta columna se crea como una variable booleana que indica si el texto de “warranty” contiene la palabra “fabrica” (garantía de fábrica). warranty_no: Se crea otra variable booleana para indicar si el texto de

"warranty" contiene "sin garantia" (no hay garantía). warranty_missing: Esta columna se crea como una variable booleana (0 o 1) que indica si el texto de "warranty" está ausente o no (NaNs). warranty_days: Esta columna extrae valores numéricos del texto de "warranty", específicamente buscando patrones como "X días", "X meses" o "X años" para luego representar esta duración en días. warranty_days_missing: Similar a "warranty_missing", esta columna se crea como una variable booleana que indica si la columna "warranty_days" está ausente o no (NaN o no NaN).

Las columnas "warranty_saler", "warranty_factory", "warranty_no" y "warranty_days" luego se llenan con -1 en las filas donde la información correspondiente falta (NaN).

La columna original "warranty" se elimina del DataFrame después de procesarla.

0.2.6 Representación semantica con Word2Vec

Con el uso de la librería `NLTK`, creamos la columna `tokenized_title` tokenizando la columna "title". Primero usa dividir el texto en oraciones, y luego cada oracion en tokens, resultando es una lista de oraciones tokenizadas.

Para generar los embeddings, entrenamos un modelo Word2Vec utilizando los títulos tokenizados. El modelo se configura con hiperparámetros específicos como el tamaño del vector (300 embedings). El modelo luego se guarda como "title_w2v.model" que nos permite utilizar el modelo Word2Vec preexistente sin volver a entrenar.

Para generar una representación semantica uniforme de los titulos, generamos el atributo `title_embs` que contiene el promedio de todos los embeddings de los títulos tokenizados. Otra opción posible representación podría haber sido solo la suma de todos los embeddings o quizas el producto de cada componente.

Finalmente creamos una serie de columnas con nombres: "embeddings_i," $\forall i \in [0, n)$ siendo n la cantidad de dimensiones de un embedding para que cada muestra contenga la representación vectorial de su título.

0.2.7 Reduccion de dimensionalidad de los Embdedings W2V con PCA

Optamos utilizar PaCMAP con PCA para reducir la dimensionalidad de los embeddings de los titulos. Esto nos permitió entrenar los modelos mas rápido, aunque a la larga notamos que era a cuestas de la performance, por lo que se utilizó para poder evaluar otros atributos mas rapido, pero en el modelo final se utilizó el embedding de 300 dimensiones.

Para esto se crearon nuevas columnas con las dimension reducida, Las nuevas columna con las dimensiones reducidas se llaman "pacmap_0," "pacmap_1," y así sucesivamente hasta el número de dimensiones especificado (dims) y se borran las columnas "embeddings_0," "embeddings_1," etc. originalmente creadas.

Las dimensiones de los embeddings y su reducción por PCA hubieran sido buenos hiperparametros a tunear, pero el entrenamiento de W2V era substancialmente grande como preambulo al entrenamiento de los modelos previos, como tambien lo era el modelo de PaCMAP.

Finalmente eliminamos las columnas "title_embs" y "tokenized_title" ya que no se utilizan como atributos.

0.2.8 Fechas

La columna "date" es convertida a un formato "datetime" para poder capturar información sobre aspectos temporales de la misma. De la fecha extrajimos solo el día, mes y día de la semana (1 a 7) para incluir como atributos ya que todo el dataset contaba con fechas entre marzo y abril del mismo año y no contaba con la hora. De haber incluido la hora nos hubiera interesado ver como el momento del día de la visita a MeLi (mañana, tarde y noche) afectaba a la variable conversión.

0.3 Armado de conjunto de validación

Inicialmente planteamos dividir el Dataset en 2, un Set de Test y otro de Train, este holdout set era en proporción 25/75. Luego, para poder validar el modelo, separamos el 20% del set de Train para tener un

sub_conjunto de validación para evaluar rápido como afectaba al modelo cada cambio hecho a los atributos.

Para probar la ingeniería de atributos detallada previamente, entrenabamos un modelo y mediamos su performance haciendo K-Fold Cross Validation con 5 folds sobre el Train Set, si mejoraba, reentrenabamos con todo el Train Set.

Una opción que consideramos fue que nuestra validación tenga la misma cantidad de samples que la cantidad de samples de los datos a predecir en Kaggle. En un comienzo nuestro holdout set era 25/75, cuando planteamos esto lo cambiamos a 20/80.

Otra cuestión que consideramos era el desbalance que había en el Dataset respecto a la variable objetivo, ya que en la mayoría de los samples no ocurria la conversión. Para solucionar esto, se planteo utilizar un método de submuestreo aleatorio para reducir la cantidad de muestras de la clase mayoritaria (no vendidos) a la cantidad de muestras de la clase minoritaria (vendidos). Esto se hizo para evitar que el modelo se sobreajuste a la clase mayoritaria y para mejorar el rendimiento del modelo en la clase minoritaria. Sin embargo observamos una performance substancialmente menor en el modelo, por lo que no se utilizó.

Alternativamente, planteamos oversamplear la clase minoritaria, pero por los distintos splits que le haciamos a los datasets, sobre todo luego cuando utilizamos K-Fold Cross Validation, no queriamos correr el riesgo de Data Leakage y evitar la repetición de datos en los distintos folds. Por lo que tampoco se utilizó.

De todas formas todos estos detalles se encuentran implementados en el código y cuentan con sus parametros para incluirlos o no en su ejecución.

0.4 Exploración de Atributos por sobre nuestro modelo principal

Como modelo principal nos enfocamos en XGBoost. Para estudiar el comportamiento de los atributos por sobre nuestros modelos predictivos planteamos la siguiente metodología:

- Se decidían las modificaciones a realizar sobre los atributos, por ejemplo, eliminar una columna, agregar una nueva, etc.
- Se entrenaba un modelo XGBoost de Clasificación Binaria con sus parametros default, con el 75% del total de datos de Train (75% de 80% del total) y se hacía K-Fold Cross Validation (K=5) con el otro 25% y se evaluaba su performance respecto a la métrica AUC contra los resultados de utilizar el mismo modelo previo a la modificación. Luego se evaluaba tambien por sobre el 20% de Test.
- Si la performance mejoraba en Cross-Val (usualmente observabamos que mejoraba en el 20% de Test mas que en el 25% (del 80% de Train) de Validación):
  - Por un lado se observaba la Gain de cada atributo para ver como el atributo que agregamos o modificamos afectaba al modelo.
  - Si pertenecía al top 10 de atributos mas importantes se reentrenaba un modelo (wrap de atributos) con solo esos 10 atributos y se evaluaba su performance respecto al modelo con todos los atributos, para observar como afectaba al modelo la eliminación de los atributos menos importantes respecto a la inclusíon del nuevo atributo.
  - Luego se entrenaba el modelo con el 80% de Train y se evaluaba su performance respecto a los modelo previo a la modificación sobre el 20% de Test.

Esto lo repetimos para cada modificación que realizamos sobre los atributos.

Luego de este proceso, decidiamos si incluir esta modificación en la evaluación sobre los modelos de Ensamble (explicado luego) mas complejos que demoran mas en entrenar y no siempre podíamos estar reentrenando con distintas modificaciones.

0.5 Otros Modelos

0.5.1 DecisionTrees y RandomForest Classifiers

Inicialmente se probaron modelos de DecisionTrees y RandomForest. Los DecisionTrees se utilizaron solo para hacer un submit inicial y conocer la pipeline de optimización de HyperOpt, sin embargo luego se descartaron.

Aunque los Random Forest por si solos no llegaban a tener la performance de XGBoost (~0.83 vs +0.88 en el resultado final), y eran mas lentos para entrenar que XGB cuando los atributos crecen (para 300 embeddings de Title era extremadamente mas lento que XGB), se optaron por utilizarlos en los modelos de Ensamble junto a XGBoost y los siguientes modelos.

0.5.2 AdaBoost Classifier

Este clasificador nos resulto en performance ROC_AUC mejor que RandomForest acercandose a XGBoost, pero tomaba casi el doble de tiempo en entrenar que XGB (2' ABC vs 1'22" XGB).

0.5.3 HistGradientBoosting Classifiers

Este clasificador nos resulto en performance ROC_AUC mejor que RandomForest los dos modelos anteriores y comparable con XGBoost y por sobre todo, el entrenamiento era mas rápido que XGB (16 segundos HGB vs 1'22" XGB).

0.6 Modelos de Ensamble

0.6.1 Average y Voting

Inicialmente para reducir el bias de nuestros modelos hicimos un ensamble promediando las probas predichas por los 4 modelos anteriores, de por si tuvo mejor performance en nuestra validación holdout pero no en el leaderboard, para ello se construyó la clase AverageClassifier que permitia a partir de modelos ya entrenados, promediar las probabilidades predichas por cada uno de ellos. Para poder entrenar los modelos en simultaneo y "compartir" sus predicciones sobre sus entrenamientos, se utilizó la clase VotingClassifier de sklearn. Luego entonces contabamos con un VotingClassifier compuesto por los modelos, lo llamamos Voting(XGB, ABC, HGB, RFC) que nos daba mejor performance que XGB en nuestra validación holdout y tambien en el leaderboard.

0.6.2 Stacking

Ademas construimos un modelo que a partir de las predicciones de los modelos anteriores, entrenaba un modelo (hicimos uno con XGBoost y uno con LogisticRegression) con las predicciones de los modelos anteriores como atributos y los mismos labels a predecir, para ello utilizamos la clase StackingClassifier de sklearn, lo llamamos Stack(XGB, ABC, HGB, RFC). El modelo que mejor performance obtuvo en el Leaderboard Público fue un Stack(XGB, ABC, HGB) con un XGBoost como modelo final. Armamos tambien otro StackingClassifier (Stack2) sin RFC para acortar en tiempo de entrenamiento pero su performance fue peor.

0.6.3 Ensamble de Ensamble

Otra opción que contemplamos fue promediar las predicciones de los modelos anteriores (Voting(XGB, ABC, HGB, RFC) y Stack(XGB, ABC, HGB, RFC) -> RegLog), a posteriori, este fue nuestro modelo que mejor performance obtuvo en el Leaderboard Privado y fue de los nuestros mejores en el Público.

0.6.4 Bagging

Finalmente, la opción por la que optamos fue hacer Bootstrap Aggregating (Bagging) por sobre XGBoost, para ello utilizamos la clase BaggingClassifier de sklearn, y *baggeamos* 10 estimadores de XGBoost (Mas sobre esta decisión luego), lo llamamos Bagging(XGB). Este fue el modelo que mas tiempo toma en entrenar, ademas fue el que mejor performance obtuvo en nuestra Validación, pero no en el Leaderboard.

0.7 Metodología Búsqueda de Hiperparámetros

En las primeras instancias del desarrollo, optamos por utilizar la librería HyperOpt para el tuneo de Hiperparametros. Lo utilizamos sobre DecisionTrees, RandomForest y XGBoost.

El espació de hiperparametros para XGBoost que nos hayó la mejor performance sobre nuestra validación en su momento fue:

```
    "max_depth": hp.choice("max_depth", [2, 4, 8, 16, 32, 64, 128, None]),
    "learning_rate": hp.uniform("learning_rate", 0.01, 0.2),
    "n_estimators": hp.choice("n_estimators", [10, 50, 100, 200, 500]),
    "colsample_bytree": hp.uniform("colsample_bytree", 0.5, 1),
    "gamma": hp.uniform("gamma", 0, 1),
    "min_child_weight": hp.choice("min_child_weight", [2, 4, 8, 16, 32, 64, 128]),
    "subsample": hp.uniform("subsample", 0.5, 1)
```

Esto podía correr en nuestra máquina por que todavía no contabamos con gran cantidad de atributos (apenas 55).

Sin embargo aquel que mejores resultados obtenía en el Leaderboard era aquel que utilizaba XGBoost con los hiperparametros default, por lo que decidimos replantear la búsqueda de hiperparametros.

En las siguientes instancias del desarrollo, optamos por utilizar la librería RandomizedSearchCV para el tuneo de Hiperparametros, por dos motivos principales: Hiperopt es muy lento en nuestras máquinas, y aunque corrimos todo parelelizado (8 cores de la CPU), luego debimos aprovechar del tiempo de computo de Google Colab sobre una GPU T4 para poder no solo paralelizar si no que entrenar XGBoost con la GPU y tambien HyperOpt, sin embargo Colab cuenta con menos RAM por lo que trabajar con mayor cantidad de atributos sobre Colab fue imposible; a mayor cantidad de atributos, la demanda de memoria RAM se disparó (la primera vez que observamos esto fue al hacer OHE sobre Category_ID) y no pudimos seguir trabajando sobre Colab. Para solucionar ambos problemas decidimos utilizar RandomizedSearchCV que nos permitia explorar de forma mas amplia el espacio de hiperparametros.

El espacio de hiperparametros que utilizamos sobre RandomizedSearchCV fue:

```
    "max_depth": [5, 10, 15, 30, 50],
    "n_estimators": [25, 50, 75, 100, 250, 500],
    "learning_rate": [0.001, 0.01, 0.1, 0.2],
    "colsample_bytree": [0.5, 0.6, 0.7, 0.8, 0.9],
    "gamma": [0, 0.1, 0.2, 0.3, 0.4, 0.5],
    "min_child_weight": [0, 1, 2, 3, 4, 5, 6],
    "subsample": [0.5, 0.6, 0.7, 0.8, 0.9],
```

Por cuestiones de tiempo, comenzamos a hacer RandomizedSearchCV con 100 fittings pero lo interrumpimos luego de haber completado 30'. Los hiperparametros que hayamos fueron los siguientes:

```
{'subsample': 0.8,
  'n_estimators': 500,
  'min_child_weight': 5,
  'max_depth': 5,
  'learning_rate': 0.1,
  'gamma': 0.1,
  'colsample_bytree': 0.7}
```

La busqueda de hiperparametros random la evaluamos sobre un 20% sobre el 80% de Train, esto nos resulto en una performance en nuestra validación de: 0.8923

Con estos hiperparametros reentrenamos con los 80% de train enteros y obtivimos un ROC_AUC de 0.90084

Al Baggear 10 estimadores de XGB sin el tuneo de hiperparametros, obtuvimos un ROC_AUC de 0.90156 con un tiempo de entrenamiento de 24' en nuestra CPU, mientras al baggear 10 estimadores de XGB con

los hiperparametros hayados obtuvimos un ROC_AUC de 0.9025 con un tiempo de entrenamiento de 1h. Nos hubiera gustado considerar la cantidad de estimadores a baggear como parte de la optimización de hiperparametros pero no llegamos a hacerlo.

Otra opción que contemplamos, fue tunear los hiperparametros para el modelo Stack(XGB, AGB, HGB) -> XGB, tambien con RSCV, el espacio de hiperparametros era muy grande:

```
    "final_estimator__n_estimators": [25, 50, 100],
    "final_estimator__max_depth": [5, 10, 25, 50, 100],
    "final_estimator__learning_rate": [0.001, 0.01, 0.1, 0.2, 0.3],
    "final_estimator__gamma": [0, 0.1, 0.2, 0.3, 0.4, 0.5],
    "final_estimator__min_child_weight": [0, 1, 2, 3, 4, 5, 6],
    "final_estimator__subsample": [0.5, 0.6, 0.7, 0.8, 0.9],
    "final_estimator__colsample_bytree": [0.5, 0.6, 0.7, 0.8, 0.9],
    "xgb__n_estimators": [25, 50, 100],
    "xgb__max_depth": [5, 10, 25, 50, 100],
    "xgb__learning_rate": [0.001, 0.01, 0.1, 0.2, 0.3],
    "xgb__gamma": [0, 0.1, 0.2, 0.3, 0.4, 0.5],
    "xgb__min_child_weight": [0, 1, 2, 3, 4, 5, 6],
    "xgb__subsample": [0.5, 0.6, 0.7, 0.8, 0.9],
    "xgb__colsample_bytree": [0.5, 0.6, 0.7, 0.8, 0.9],
    "hgb__max_depth": [2, 3, 4, 5, 6, 7, 8, 9, 10],
    "hgb__learning_rate": [0.001, 0.01, 0.1, 0.2, 0.3],
    "hgb__max_iter": [100, 200, 300, 400, 500],
    "hgb__max_leaf_nodes": [10, 20, 30, 40, 50],
    "hgb__min_samples_leaf": [10, 20, 30, 40, 50],
    "hgb__l2_regularization": [0, 0.1, 0.2, 0.3, 0.4, 0.5],
    "abc_adaboostclassifier_n_estimators": [25, 50, 100],
    "abc_adaboostclassifier_learning_rate": [0.001, 0.01, 0.1, 0.2, 0.3],
```

No llegamos hacer una busqueda muy exhaustiva de hiperparametros, por lo que finalmente descartamos los stacks ya que no lo ibamos a poder hacer un buen Tune de sus hiperparametros, lo mismo para los modelos de ensamble de tipo Votings.

0.8 Importancia de atributos

0.9 Responder

Dada una persona que se encuentra diseñando un anuncio de venta de un producto para publicar en este destacado retailer online, ¿en qué aspectos le recomendarían enfocarse? ¿Ven alguna debilidad en este análisis?

Respuesta: