

Solución Test MeLi












Javier Martínez

Resumen

Los desarrollos se encuentran disponibles en el repositorio público **test_mercado_libre** (https://github.com/esglobe/test_mercado_libre). La solución fue dividida en tres secciones, la primera, destinada a la creación de una base de datos tabular (data_base.ipynb). La segunda, creada para el análisis de datos (data_analysis.ipynb) y finalmente, el proceso para el ajuste o entrenamiento de modelos (ML_process.ipynb). En la búsqueda del mejor modelo (accuracy mayor a 0.86) se aplicaron los algoritmos de Decision Tree, Random Forest, Gradient boosting y eXtreme Gradient Boosting desde la percepción de la clasificación binaria (new=1 y used=0). Para cada uno de los algoritmos se implementó una grilla con el propósito de evaluar el desempeño (performance). En definitiva, **el mejor modelo** encontrado alcanzó un **accuracy de 0.8651** con el uso del algoritmo eXtreme Gradient Boosting (n_estimators = 300). Las tres variables (features) más significativas (importance) fueron el **initial_quantily**, el valor **free** de **listing_type_id** y **accepts_mercadopago**.

El repositorio test_mercado_libre

Se creó el repositorio público **test_mercado_libre** (https://github.com/esglobe/test_mercado_libre), el cual dispone, en su rama master, de la siguiente estructura:

 esglobe actualización 090320234	f4acc8b 5 hours ago	 8 commits
 code	actualizacion_05030223	3 days ago
 documentos	actualizacion 070320233	yesterday
 summary	actualizacion 090320234	5 hours ago
 .gitignore	actualizacion_05030223	3 days ago
 ML_process.ipynb	actualizacion 090320234	5 hours ago
 README.MD	actualizacion_05030223	3 days ago
 data_analysis.ipynb	actualizacion 080320234	17 hours ago
 data_base.ipynb	actualizacion 080320234	17 hours ago
 environment.yml	actualizacion_05030223	3 days ago

Tal que;

1. **code:** Contiene el código **new_or_used.py** para la lectura de los datos.
2. **documentos:** Dispone de los criterios para el desarrollo del test (Opportunities@MeLi - CodeExercise DS_ML.docx).
3. **summary:** Tiene el resumen tras el entrenamiento de los modelos.
4. **ML_process.ipynb:** El proceso creado para el ajuste o entrenamiento de los modelos.
5. **README.MD:** Comandos para la creación del conda environment.
6. **data_analysis.ipynb:** Los desarrollos destinados al estudio de los datos.
7. **data_base.ipynb:** Mecanismo de procesamiento de la información.
8. **environment.yml:** Creación del conda environment (analysis).

El ambiente local dispone de las siguientes librerías y dependencias:

```
name: analisis
channels:
- conda-forge
- nodefaults
dependencies:
- python==3.9
- cython==0.29.32
- matplotlib==3.5.2
- pandas==1.4.3
- numpy==1.23.1
- plotly==5.9.0
- scikit-learn==1.1.1
- jupyter==1.0.0
- pip==22.2.1
- pip:
- tensorflow==2.9.1
- tensorflow-cpu==2.9.1
- pygad==2.17.0
- xgboost==1.7.4
```

Procesamiento de la base de datos

El procesamiento de datos y consolidación de la información es detallado en el notebook **data_base.ipynb** (https://github.com/esglobe/test_mercado_libre/blob/master/data_base.ipynb). Inicialmente, se realizó la lectura del archivo **MLA_100k_checked_v3.jsonlines** modificando la función **build_dataset** para incorporar la marca de training o test

en el conjunto de datos ya que, al ser un json, pueden existir estructuras distintas en los documentos de Training y Test. Para evitar problemáticas a futuro se consolidó una base de datos con los 100,000 artículos y así estudiar los documentos.

En la primera observación se descartaron las siguientes variables:

- * **international_delivery_mode:** Todos los valores nulos.
- * **listing_source:** Todos los valores nulos ({}).
- * **coverage_areas:** Todos los valores nulos. ({}).
- * **differential_pricing:** Todos los valores nulos.
- * **subtitle:** Todos los valores nulos. (None).
- * **descriptions:** Igual al ID.
- * **site_id :** Misma codificación ('MLA').
- * **catalog_product_id:** Solo 11 valores no nulos.
- * **original_price:** Solo 148 valores no nulos.
- * **sub_status:** Solo 986 valores no nulos.

Para usar la mayor información posible se aplicó un proceso que toma la data de los siguientes documentos:

- * **non_mercado_pago_payment_methods_function:** El id del método de pago.
- * **shipping:** La información de las variables **local_pick_up**, **free_shipping** y **mode**.
- * **seller_address:** La información del **seller_address_country**, **seller_address_state** y **seller_address_city**.
- * **tags:** Información dentro de la lista **tags**.

Del mismo modo, se crearon las variables booleanas:

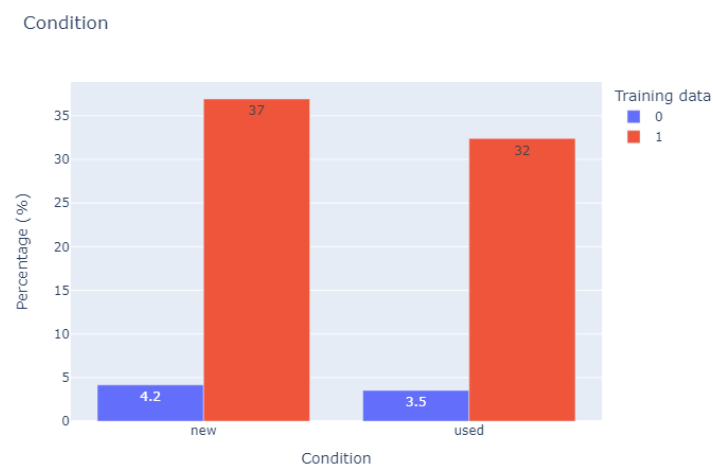
- * **variation_new:** 1 si tiene información en **variations**.
- * **deal_ids_new:** 1 si tiene información en **deal_ids**.
- * **video_id_new:** 1 si tiene información en **video_id**.
- * **attributes_new:** 1 si tiene información en **attributes**.
- * **condition_new:** 1 si es new y 0 si es used.
- * **warranty_yes:** 1 si existe alguna de las palabras OFICIAL, SI, FABRICA, NUEVO, REPUTACION O MESES en **warranty**. Cero en otro caso.

- * **warranty_no**: 1 si existe alguna de las palabras USADO, NO, VIEJO O SIN GARANTIA en **warranty**. Cero en otro caso.
- * **official_store_id_new**: 1 si tiene id de tienda oficial y 0 en otro caso.
- * **title_new**: 1 si existe la palabra NUEVO, NEW O ESTRENO en **title**. 0 en otro caso.
- * **title_usado**: 1 si existe la palabra USADO, VIEJO O ANTIGUO en **title**. 0 en otro caso.

Todo lo anterior permitió la consolidación de la base de datos **./data/data_base.pkl** que consta de 100,000 registros y 51 variables.

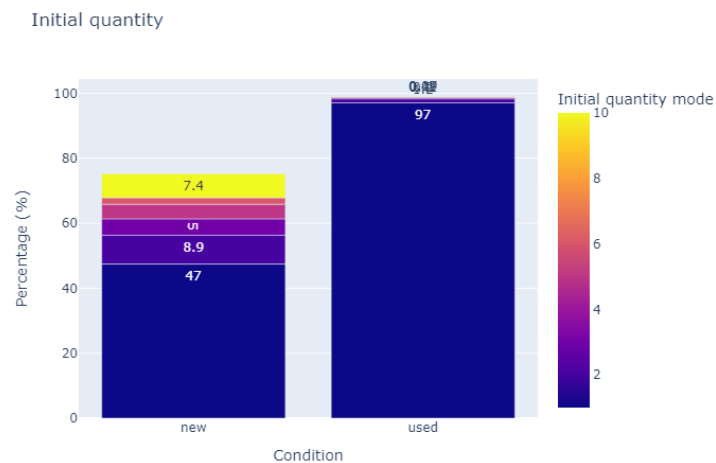
Análisis de datos

El análisis de los datos es realizado en el notebook ***data_analysis.ipynb*** (https://github.com/esglobe/test_mercado_libre/blob/master/data_analysis.ipynb). Al estudiar el número de artículos nuevos y usados en la data de entrenamiento, se pudo concluir que se encuentran relativamente balanceados (37% para el caso de nuevos y 32% para el caso de usados).



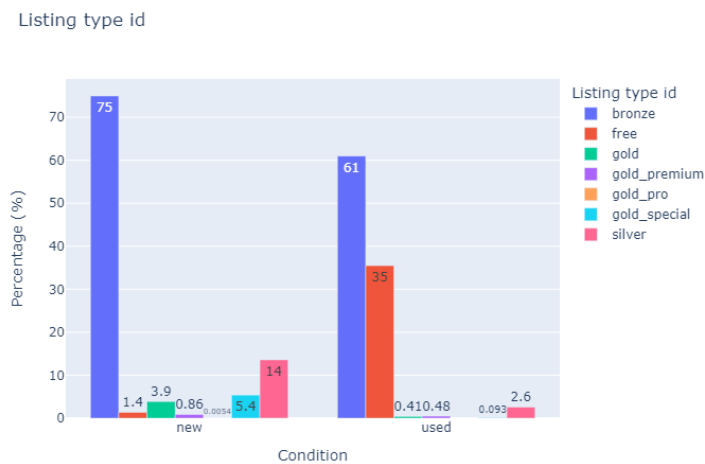
condition	training_data	parent_item_id	percentage
new	0	4153	4.153
new	1	36921	36.921
used	0	3526	3.526
used	1	32389	32.389

Una de las variables más significativas en la clasificación de nuevos y usados fue ***initial_quality***. Donde los productos usados, en el 97% de los casos, mostraron un valor de 1. Es importante señalar que en el 95,5% de los casos los valores de ***available_quantity*** son iguales a ***initial_quality*** por lo que se descarta como feature de modelos.



condition	initial_quantity	parent_item_id	total	percentage
used	1	31429	32389	97.036031
new	1	17512	36921	47.431001
new	2	3274	36921	8.867582
new	10	2729	36921	7.391457
new	3	1864	36921	5.048617
new	5	1661	36921	4.498795
new	4	1364	36921	3.694374

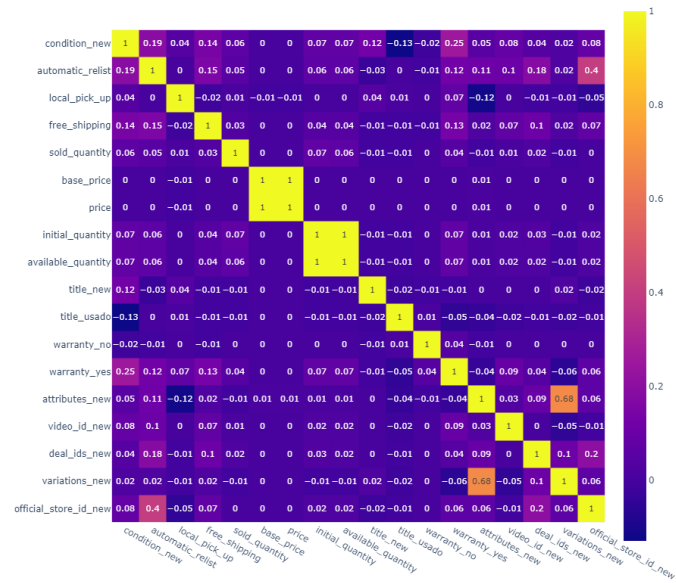
Otra de las variables significativas encontradas fue **list_type_id** en la cual se pudo notar una mayor proporción de casos nuevos en las categorías **bronze**, **gold**, **gold_special** y **silver**. Mientras que, para los casos usados, existe una proporción significativamente mayor en la categoría **free**.



condition	listing_type_id	parent_item_id	total	percentage
new	bronze	27671	36921	74.946507
new	free	500	36921	1.354243
new	gold	1430	36921	3.873135
new	gold_premium	318	36921	0.861298
new	gold_pro	2	36921	0.005417
new	gold_special	1987	36921	5.381761
new	silver	5013	36921	13.577639
used	bronze	19750	32389	60.977492
used	free	11490	32389	35.475007
used	gold	132	32389	0.407546
used	gold_premium	156	32389	0.481645
used	gold_special	30	32389	0.092624
used	silver	831	32389	2.565686

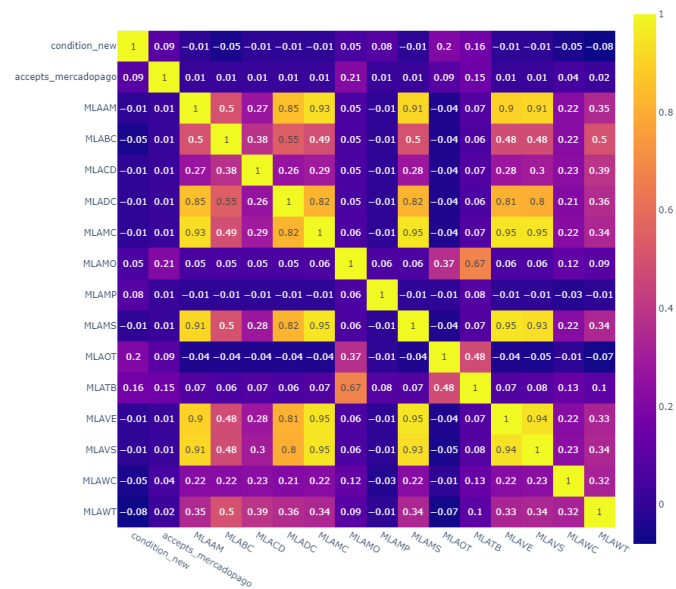
Al estudiar la correlación lineal de las variables numéricas con el target (**condition_new**), se pudo concluir que **price** y **base_price** presentan una baja correlación lineal con **condition_new** (correlación cero). Además se pudo identificar que **price** y **base_price** son iguales en el 99.9% de los casos, en el conjunto de training, por lo que solo será considerada **base_price** en el entrenamiento de ser necesario.

Correlation



Las variables con mayor correlación lineal fueron **warranty_yes** (0.25), **MLAOT** (0.2), **automatic_relist** (0.19) y **MLATB** (0.16). Lo que sugiere que la garantía y los métodos de pago distintos al mercado pago inciden en la presencia de artículos nuevos.

Correlation



En general, dada la meta establecida para el accuracy del modelo, se van a utilizar la mayor cantidad de variables numéricas y booleanas posibles ya que, si bien no es evidente una clasificación marcada con métodos visuales, son fuentes de información valiosa para los algoritmos.

Por último, la variable **sold_quantity** no será utilizada como entrada en el modelado, dado que es obtenida una vez se ha realizado la venta del producto.

Entrenamiento de Modelos

El modelado es detallado en **ML_process.ipynb** (https://github.com/esglobe/test_mercado_libre/blob/master/ML_process.ipynb). Las variables a utilizar son:

Uso	Variable	Tipo
Features	buying_mode	String
	seller_address_state	String
	seller_address_city	String
	status	String
	listing_type_id	String
	tags_no_list	String
	mode	String
	accepts_mercadopago	Boolean
	automatic_relist	Boolean
	training_data	Boolean
	local_pick_up	Boolean
	free_shipping	Boolean
	title_new	Boolean
	title_usado	Boolean
	warranty_no	Boolean
	warranty_yes	Boolean
	attributes_new	Boolean
	video_id_new	Boolean
	deal_ids_new	Boolean
	variations_new	Boolean
	official_store_id_new	Boolean
	non_mercado_pago_payment_methods	String
	base_price	Númérica
	initial_quantity	Númérica
Target	condition_new	Boolean

El proceso programado lleva a dummies las variables categóricas mientras que se aplica la transformación Minimax a las numéricas para mantener la misma escala (entre cero y 1). Lo anterior ayuda en el performance o desempeño.

```
dummies = ['buying_mode',
            #'seller_address_country',
            'seller_address_city',
            'seller_address_state',
            'status',
            'listing_type_id',
            'mode',
            'tags_no_list'
            ]

pd_x_data = pd.get_dummies(pd_model, columns=dummies)
```

MinMaxScaler

```
numbers = ['sold_quantity',
            'base_price',
            'price',
            'seller_id',
            'initial_quantity',
            'available_quantity',
            ]

scaler = MinMaxScaler()
scaler.fit(pd_x_data[numbers])
```

Luego, se constituyen las bases de datos usadas en el entrenamiento (*pandas_x_training* y *pandas_y_training*) y validación (*pandas_x_test* y *pandas_y_test*). Las bases de datos consolidadas para la entrada de los modelos tiene 372 inputs.

```
# Training data
pandas_x_training = pd_x_data.query('training_data==1').copy().fillna(0).astype(float)
pandas_y_training = pandas_x_training[out].astype(int)
pandas_x_training.drop(labels=['training_data', out], axis=1, inplace=True)
pandas_x_training.head(3)
```

pandas_x_training.shape

(90000, 372)

```
# Test Data
pandas_x_test = pd_x_data.query('training_data==0').copy().fillna(0).astype(float)
pandas_y_test = pandas_x_test[[out]].astype(int)
pandas_x_test.drop(labels=['training_data', out], axis=1, inplace=True)
pandas_x_test.head(3)
```

pandas_x_test.shape

(10000, 372)

Se crearon las clases **DecisionTree_MODEL**, **RandomForestClassifier_MODEL**, **GradientBoostingClassifier_Model** y **XGB_MODEL** para realizar experimentos con el uso de los algoritmos Decision Tree, Random Forest, Gradient boosting y eXtreme Gradient Boosting, respectivamente. Cada una de estas clases hizo posible el entrenamiento parametrizado calculando las métricas de *accuracy*, *precisión*, *recall* y *auc* (área bajo la curva ROC).

Para mejorar los tiempos en la ejecución de los experimentos se utilizó la función **map** en lugar del bucle **for**. El primer experimento consistió en entrenar 6 Decision Trees variando los parámetros de **criterion** y **splitter** dejando **max_depth** por defecto. Pero no se alcanzó un accuracy del 0.86.

```
#####
#DecisionTree_MODEL
#####
DecisionTree_models = list(map(lambda x,y: DecisionTree_MODEL.select_model(criterion=x, splitter=y, max_depth=None),\
                               ['gini','gini','entropy','entropy','log_loss','log_loss'],
                               ['best','random','best','random','best','random']
                               ))

# Summary
DecisionTree_summary = pd.concat(list(map(lambda x: x.summary, DecisionTree_models)))
DecisionTree_summary.to_csv('./summary/dt.csv')
DecisionTree_summary.round(2)
```

	accuracy	precision	recall	auc	criterion	splitter	max_depth
0	0.84	0.89	0.80	0.84	gini	best	None
0	0.83	0.87	0.80	0.83	gini	random	None
0	0.84	0.89	0.80	0.84	entropy	best	None
0	0.84	0.88	0.81	0.84	entropy	random	None
0	0.84	0.89	0.80	0.84	log_loss	best	None
0	0.84	0.88	0.81	0.84	log_loss	random	None

Seguidamente se realizaron experimentos utilizando el algoritmo Random Forest. En esta oportunidad se entrenaron 4 modelos dejando fijo el criterio de entropía (**entropy**), la máxima profundidad se dejó por defecto y se aplicó una grilla para **n_estimators**. En ninguno de los casos se logró la meta del accuracy (0.86).

```

: #####
#RandomForestClassifier_MODEL
#####
RandomForest_models = list(map(lambda x,y: RandomForestClassifier_MODEL.select_model(n_estimators=y, criterion=x, max_depth=None),\
                                4*['entropy'],\
                                [100,200,300,400]
                                ))

# Summary
RandomForest_summary = pd.concat(list(map(lambda x: x.summary, RandomForest_models)))
RandomForest_summary.to_csv('./summary/rf.csv')
RandomForest_summary.round(2)

```

	accuracy	precision	recall	auc	criterion	max_depth	n_estimators
0	0.85	0.89	0.83	0.85	entropy	None	100
0	0.85	0.89	0.83	0.85	entropy	None	200
0	0.85	0.89	0.83	0.85	entropy	None	300
0	0.85	0.89	0.83	0.85	entropy	None	400

Dando continuidad, se aplicó el algoritmo Gradient Boosting en la ejecución de experimentos que varían la tasa de aprendizaje (**learning_rate** 0.1 o 0.3) y el **n_estimators** (100, 200 y 300). Los resultados muestran que nuevamente no fue posible alcanzar la meta del performance mínimo (0.86).

```

#####
#GradientBoostingClassifier_Model
#####
GradientBoosting_models = list(map(lambda x,y: GradientBoostingClassifier_Model.select_model(learning_rate=x, n_estimators=y),\
                                    [0.1,0.1,0.1,\
                                    0.3,0.3,0.3],\
                                    [100,200,300,\
                                    100,200,300]
                                    ))

# Summary
GradientBoosting = pd.concat(list(map(lambda x: x.summary, GradientBoosting_models)))
GradientBoosting.to_csv('./summary/gb.csv')
GradientBoosting.round(2)

```

	accuracy	precision	recall	auc	learning_rate	n_estimators
0	0.83	0.89	0.77	0.83	0.1	100
0	0.83	0.89	0.79	0.84	0.1	200
0	0.83	0.89	0.79	0.84	0.1	300
0	0.83	0.89	0.79	0.84	0.3	100
0	0.83	0.89	0.80	0.84	0.3	200
0	0.84	0.89	0.80	0.84	0.3	300

Posteriormente, se ejecutaron 6 experimentos utilizando el algoritmo del eXtreme Gradient Boosting variando el **n_estimators** (50, 100, 200, 300, 400 y 500);

```

#=====
#XGB_MODEL
#=====
xgb_models= list(map(lambda x: XGB_MODEL.select_model(n_estimators=x),\
                    [50,100,200,300,400,500]))

# Summary
xgb_summary = pd.concat(list(map(lambda x: x.summary, xgb_models)))
xgb_summary.to_csv('./summary/xgb.csv')
xgb_summary.round(2)

```

	accuracy	precision	recall	auc	n_estimators
0	0.85	0.87	0.85	0.85	50
0	0.86	0.88	0.86	0.86	100
0	0.86	0.89	0.86	0.86	200
0	0.87	0.89	0.85	0.87	300
0	0.86	0.89	0.85	0.87	400
0	0.86	0.89	0.85	0.86	500

El modelo seleccionando fue el de **n_estimators** = 300 donde se alcanzó **accuracy** de **0.8651** (0.87), una precisión de **0.8912**, un **recall** de 0.8547 y un **AUC** de 0.8660. En el estudio se ha seleccionado a la precisión como segunda métrica ya que esta permite determinar el porcentaje de casos positivos detectados por el modelo. Concretamente:

$$Precision = \frac{TP}{(TP + FP)} = \frac{4,621}{(4,621 + 564)} = 0.8912$$

```

# Best Model
best_model = list(filter(lambda x: x.n_estimators == 300,xgb_models))

```

```

# Confusion Matrix Best Model
best_model[0].confusion_matrix

```

```

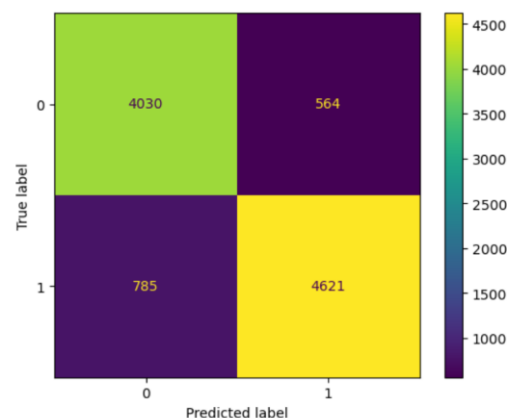
array([[4030, 564],
       [ 785, 4621]])

```

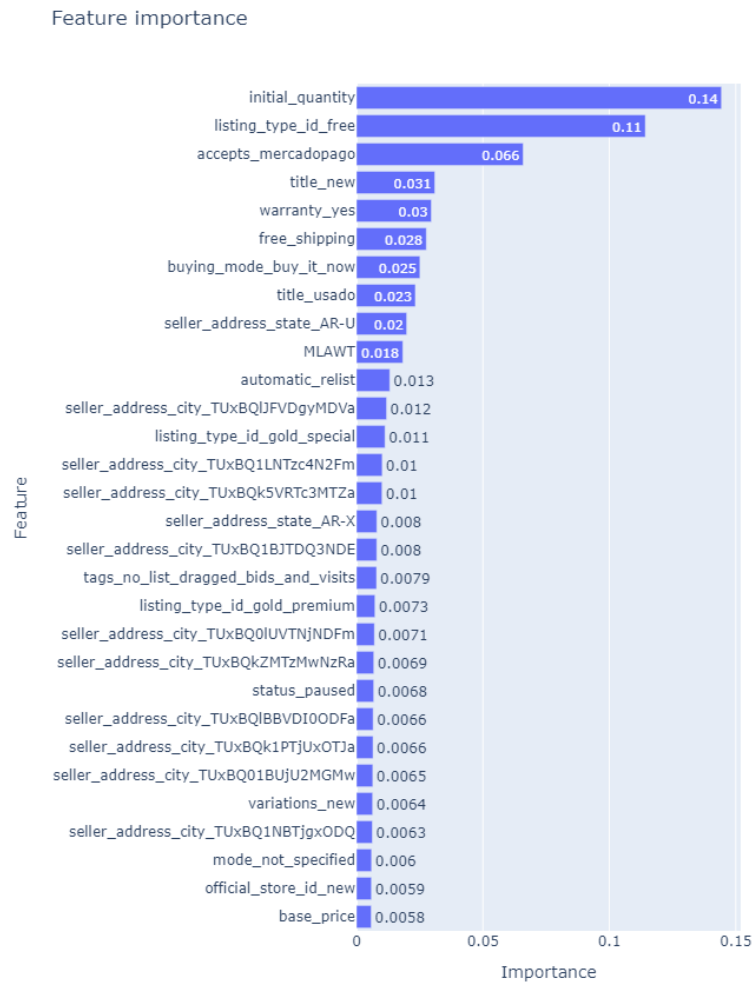
```

# Confusion Matrix Plot Best Model
fig = best_model[0].plot_matrix.plot()

```



Las primeras 30 variables de mayor importancia (**importance**) para el modelo son:



A continuación, se detallan las clases creadas para los algoritmos de Decision Tree (DecisionTree_MODEL), Random Forest (RandomForestClassifier_MODEL), Gradient boosting (GradientBoostingClassifier_MODEL) y eXtreme Gradient Boosting (XGB_MODEL) usadas en la generación de experimentos:

Clase XGB_MODEL:

```
class XGB_MODEL():
    """
    model training
    """

    def __init__(self, pandas_x_training,
                  pandas_y_training,
                  pandas_x_test,
                  pandas_y_test):

        self.pandas_x_training = pandas_x_training
        self.pandas_y_training = pandas_y_training
        self.pandas_x_test = pandas_x_test
        self.pandas_y_test = pandas_y_test

    def training(self, n_estimators):

        self.n_estimators = n_estimators

        np.random.seed(seed)
        # Model
        self.model = XGBClassifier(n_estimators=n_estimators,
                                   verbosity=0)

        # Fit
        self.model.fit(self.pandas_x_training.values,
                       self.pandas_y_training.values
                       )

        # Metrics
        self.prediction = self.model.predict(pandas_x_test.values)
        self.accuracy = accuracy_score(self.pandas_y_test.values, self.prediction)
        self.precision = precision_score(pandas_y_test.values, self.prediction)
        self.recall = recall_score(self.pandas_y_test.values, self.prediction)

        fpr, tpr, thresholds = roc_curve(self.pandas_y_test.values, self.prediction, pos_label=1)
        self.auc = auc(fpr, tpr)

        self.confusion_matrix = confusion_matrix(self.pandas_y_test.values, self.prediction)

        # Plot
        self.plot_matrix = ConfusionMatrixDisplay(confusion_matrix=self.confusion_matrix)

        # Summary
        self.summary = pd.DataFrame({'accuracy': self.accuracy,
                                     'precision': self.precision,
                                     'recall': self.recall,
                                     'auc': self.auc,
                                     'n_estimators': n_estimators}, index=[0])

    @staticmethod
    def select_model(n_estimators):
        xgb_model = XGB_MODEL(pandas_x_training,
                              pandas_y_training,
                              pandas_x_test,
                              pandas_y_test)
        xgb_model.training(n_estimators=n_estimators)
        return xgb_model
```

Clase DecisionTree_MODEL:

```
class DecisionTree_MODEL():
    """
    model training
    """

    def __init__(self, pandas_x_training,
                  pandas_y_training,
                  pandas_x_test,
                  pandas_y_test):

        self.pandas_x_training = pandas_x_training
        self.pandas_y_training = pandas_y_training
        self.pandas_x_test = pandas_x_test
        self.pandas_y_test = pandas_y_test

    def training(self, criterion='gini', splitter='best', max_depth=None):

        np.random.seed(seed)
        # Model
        self.model = DecisionTreeClassifier(criterion=criterion, splitter=splitter, max_depth=max_depth)

        # Fit
        self.model.fit(self.pandas_x_training.values,
                       self.pandas_y_training.values
                       )

        # Metrics
        self.prediction = self.model.predict(pandas_x_test.values)
        self.accuracy = accuracy_score(self.pandas_y_test.values, self.prediction)
        self.precision = precision_score(pandas_y_test.values, self.prediction)
        self.recall = recall_score(self.pandas_y_test.values, self.prediction)

        fpr, tpr, thresholds = roc_curve(self.pandas_y_test.values, self.prediction, pos_label=1)
        self.auc = auc(fpr, tpr)

        self.confusion_matrix = confusion_matrix(self.pandas_y_test.values, self.prediction)

        # Plot
        self.plot_matrix = ConfusionMatrixDisplay(confusion_matrix=self.confusion_matrix)

        # Summary
        self.summary = pd.DataFrame({'accuracy': self.accuracy,
                                     'precision': self.precision,
                                     'recall': self.recall,
                                     'auc': self.auc,
                                     'criterion': criterion,
                                     'splitter': splitter,
                                     'max_depth': max_depth
                                     }, index=[0])

    @staticmethod
    def select_model(criterion='gini', splitter='best', max_depth=None):

        model = DecisionTree_MODEL(pandas_x_training,
                                    pandas_y_training,
                                    pandas_x_test,
                                    pandas_y_test)

        model.training(criterion=criterion, splitter=splitter, max_depth=max_depth)

        return model
```

RandomForestClassifier_MODEL:

```
class RandomForestClassifier_MODEL():
    """
    model training
    """

    def __init__(self, pandas_x_training,
                 pandas_y_training,
                 pandas_x_test,
                 pandas_y_test):

        self.pandas_x_training = pandas_x_training
        self.pandas_y_training = pandas_y_training
        self.pandas_x_test = pandas_x_test
        self.pandas_y_test = pandas_y_test

    def training(self, n_estimators=100, criterion='gini', max_depth=None):

        np.random.seed(seed)
        # Model
        self.model = RandomForestClassifier(n_estimators=n_estimators, criterion=criterion, max_depth=max_depth)

        # Fit
        self.model.fit(self.pandas_x_training.values,
                       self.pandas_y_training.values
                       )

        # Metrics
        self.prediction = self.model.predict(pandas_x_test.values)
        self.accuracy = accuracy_score(self.pandas_y_test.values, self.prediction)
        self.precision = precision_score(pandas_y_test.values, self.prediction)
        self.recall = recall_score(self.pandas_y_test.values, self.prediction)

        fpr, tpr, thresholds = roc_curve(self.pandas_y_test.values, self.prediction, pos_label=1)
        self.auc = auc(fpr, tpr)

        self.confusion_matrix = confusion_matrix(self.pandas_y_test.values, self.prediction)

        # Plot
        self.plot_matrix = ConfusionMatrixDisplay(confusion_matrix=self.confusion_matrix)

        # Summary
        self.summary = pd.DataFrame({'accuracy': self.accuracy,
                                     'precision': self.precision,
                                     'recall': self.recall,
                                     'auc': self.auc,
                                     'criterion': criterion,
                                     'max_depth': max_depth,
                                     'n_estimators': n_estimators
                                     }, index=[0])

    @staticmethod
    def select_model(n_estimators=100, criterion='gini', max_depth=None):

        model = RandomForestClassifier_MODEL(pandas_x_training,
                                             pandas_y_training,
                                             pandas_x_test,
                                             pandas_y_test)

        model.training(n_estimators=n_estimators, criterion=criterion, max_depth=max_depth)

        return model
```


GradientBoostingClassifier_MODEL:

```
class GradientBoostingClassifier_Model():
    """
    model training
    """

    def __init__(self, pandas_x_training,
                 pandas_y_training,
                 pandas_x_test,
                 pandas_y_test):

        self.pandas_x_training = pandas_x_training
        self.pandas_y_training = pandas_y_training
        self.pandas_x_test = pandas_x_test
        self.pandas_y_test = pandas_y_test

    def training(self, learning_rate, n_estimators):

        np.random.seed(seed)
        # Model
        self.model = GradientBoostingClassifier(learning_rate=learning_rate, n_estimators=n_estimators)

        # Fit
        self.model.fit(self.pandas_x_training.values,
                       self.pandas_y_training.values
                       )

        # Metrics
        self.prediction = self.model.predict(pandas_x_test.values)
        self.accuracy = accuracy_score(self.pandas_y_test.values, self.prediction)
        self.precision = precision_score(pandas_y_test.values, self.prediction)
        self.recall = recall_score(self.pandas_y_test.values, self.prediction)

        fpr, tpr, thresholds = roc_curve(self.pandas_y_test.values, self.prediction, pos_label=1)
        self.auc = auc(fpr, tpr)

        self.confusion_matrix = confusion_matrix(self.pandas_y_test.values, self.prediction)

        # Plot
        self.plot_matrix = ConfusionMatrixDisplay(confusion_matrix=self.confusion_matrix)

        # Summary
        self.summary = pd.DataFrame({'accuracy': self.accuracy,
                                     'precision': self.precision,
                                     'recall': self.recall,
                                     'auc': self.auc,
                                     'learning_rate': learning_rate,
                                     'n_estimators': n_estimators
                                     }, index=[0])

    @staticmethod
    def select_model(learning_rate, n_estimators):

        model = GradientBoostingClassifier_Model(pandas_x_training,
                                                  pandas_y_training,
                                                  pandas_x_test,
                                                  pandas_y_test)

        model.training( learning_rate=learning_rate, n_estimators=n_estimators)
        return model
```