

AVANCE PROYECTO INTELIGENCIA ARTIFICIAL

Store sales - Time series

forecasting Inteligencia artificial

INTEGRANTES

**Miguel Ángel Rivas Acevedo
Santiago Lorduy Gil**

**UNIVERSIDAD DE ANTIOQUIA
FACULTAD DE INGENIERÍA
MEDELLÍN
2022**

INTRODUCCIÓN

En el siguiente trabajo encontrará el desarrollo del ejercicio de kaggle titulado “Store Sales - Time Series Forecasting” donde se buscó construir un modelo que prediga con mayor precisión las ventas unitarias de miles de artículos vendidos en diferentes tiendas de una empresa situada en Ecuador de nombre “Favorita”.

En las primeras dos entregas lo que se realizó fue una parte de manipulación de los datos donde se dejó tablas construidas de la manera en la que consideramos adecuada, en estas estaban contenidos todos los datos listos para su uso, después de haber realizado algunas interpolaciones y llenado de casillas vacías.

Para esta entrega final se realiza la implementación del modelo matemático, la métrica de evaluación utilizada fue el error logarítmico medio cuadrático (RMSLE).

Store sales - Time series forecasting

Se inicia buscando con mucho detenimiento y descargando uno a uno los datasets de diversas competencias en kaggle, se revisa y analiza el contenido, hasta que finalmente se encuentra un conjunto de datos que cumple con las características pedidas y una competencia agradable y que ve bastante prometedora e interesante para trabajar, esta se llama Store sales - Time series forecasting.

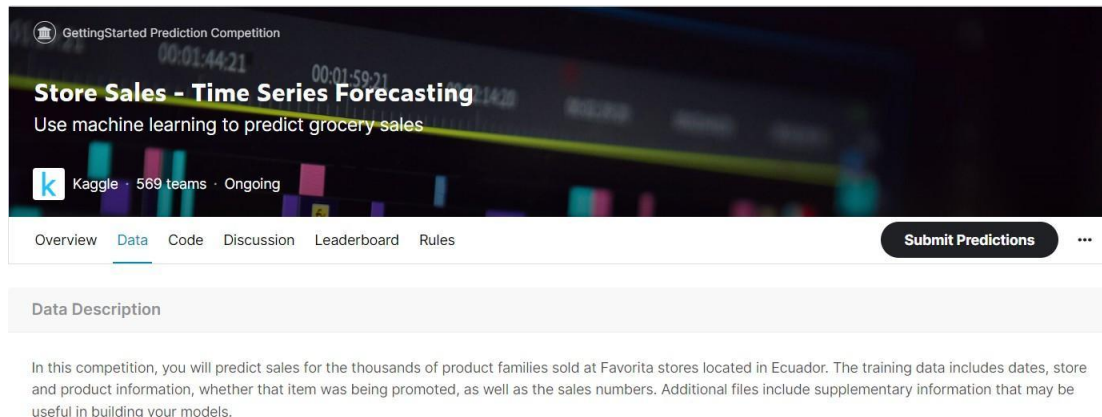


Fig 1. Competencia elegida.

En esta competencia contamos con un dataset de siete archivos, todos son .csv y se tiene uno para entrenar al algoritmo y otro para probarlo.

EXPLORACIÓN DESCRIPTIVA DEL DATASET

train.csv

El primer archivo que encontramos es el train.csv este cuenta con series temporales de características llamadas store_nbr, family y onpromotion así como las ventas objetivas.

store_nbr identifica la tienda en la que se venden los productos, family identifica el tipo de producto vendido y onpromotion proporciona el número total de artículos de una familia de productos que se estaban promocionando en una tienda en una fecha determinada.

test.csv

Este archivo contiene lo mismo que el train.csv y a este será el que se le hará la predicción de las ventas.

sample_submission.csv

Un archivo de envío de muestra en el formato correcto.

stores.csv

Tiene la información de las tiendas, incluyendo city, state, type y cluster, el cluster es un grupo similar de tiendas.

oil.csv

Precio diario del petróleo. Incluye valores durante los períodos de tiempo de los datos del train y del test.

holidays_events.csv

Contiene los días de vacaciones y fechas especiales.

PREPROCESADO DE DATOS

El dataset para esta última entrega, carga directamente desde kaggle a google colab para de este modo agilizar el proceso engorroso de descargar, descomprimir y subir los datos, solo subiendo la API key a nuestro código, ya es posible que se cargue el dataset completo y se descomprima quedando listo para utilizar todo su contenido.

```
[ ] !pip install kaggle

Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/dist-packages (1.5.12)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/dist-packages (from kaggle) (6.1.2)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.24.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages (from kaggle) (4.64.1)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/dist-packages (from kaggle) (1.15.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/dist-packages (from kaggle) (2022.9.24)
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.23.0)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<4,>=3.0.2 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (3.0.4)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->kaggle) (2.10)

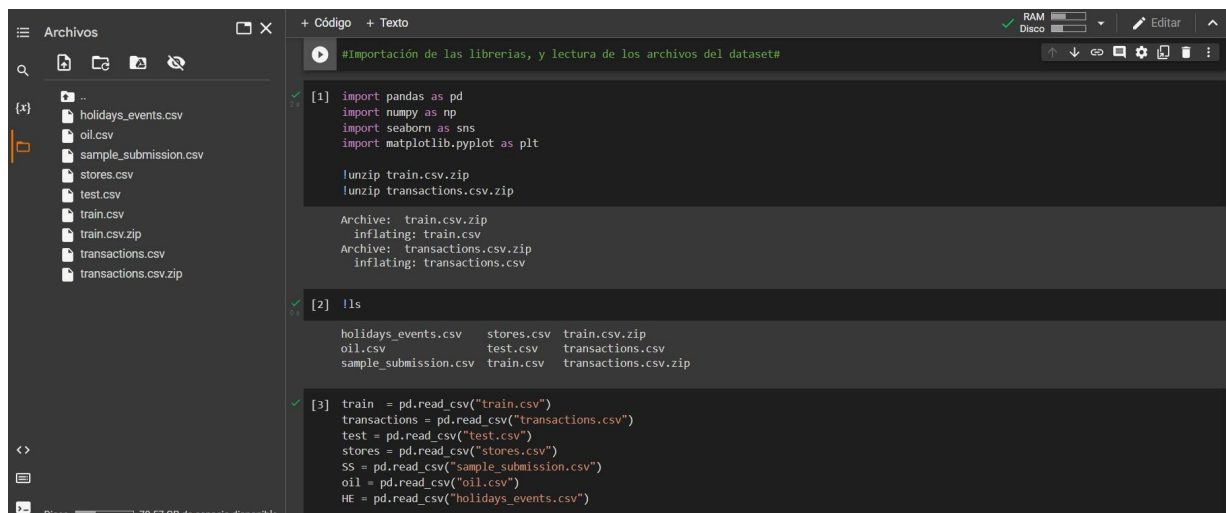
[ ] import os
os.environ['KAGGLE_CONFIG_DIR'] = '/content'

[ ] !kaggle competitions download -c store-sales-time-series-forecasting

Warning: Your Kaggle API key is readable by other users on this system! To fix this, you can run 'chmod 600 /content/kaggle.json'
store-sales-time-series-forecasting.zip: Skipping, found more recently modified local copy (use --force to force download)
```

Fig 3. Importación de los módulos y carga del dataset.

En el colab se importan todas las librerías necesarias y se suben todos los archivos, Se descomprimen los datos .zip de la competencia, posteriormente se realiza una lectura de los archivos usando la librería *pandas* y se imprime los datos de estos para ver su dimensión.



The screenshot shows a Jupyter Notebook interface. On the left is a file explorer showing a directory with files: holidays_events.csv, oil.csv, sample_submission.csv, stores.csv, test.csv, train.csv, train.csv.zip, transactions.csv, and transactions.csv.zip. The main area is the code editor with the following code:

```
#Importación de las librerías, y lectura de los archivos del dataset#

[1] import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt

!unzip train.csv.zip
!unzip transactions.csv.zip

Archive: train.csv.zip
  inflating: train.csv
Archive: transactions.csv.zip
  inflating: transactions.csv

[2] !ls

holidays_events.csv  stores.csv  train.csv.zip
oil.csv              test.csv   transactions.csv
sample_submission.csv train.csv  transactions.csv.zip

[3] train = pd.read_csv("train.csv")
transactions = pd.read_csv("transactions.csv")
test = pd.read_csv("test.csv")
stores = pd.read_csv("stores.csv")
ss = pd.read_csv("sample_submission.csv")
oil = pd.read_csv("oil.csv")
HE = pd.read_csv("holidays_events.csv")
```

The output of the code shows the successful unzipping of the train and transactions datasets, followed by a directory listing and the loading of the datasets into pandas DataFrames.

Fig 4. Importación de los módulos y carga del dataset.

Después de observar el contenido de cada archivo, se utiliza la función `isnull.num` para ver qué columnas tienen valores NaN y así poder reemplazarlos por valores que puedan ser usados posteriormente con mayor eficiencia, para este caso se cambiaron por lo que fue necesario según el dato, debido a que tras hacer un análisis se concluyó que era la manera para que la ausencia no afectará el resultado. Continuando se realiza una concatenación de los archivos creando así un `train` y un `test` completos que facilitan la manipulación de datos y la facilidad para visualizar tablas con ellos e interpretarlos, todo esto para tener la mejor predicción

posible.

```
[ ] train_ = pd.merge(train,HE,how='left',on='date')
    train_ = pd.merge(train_,oil_date,how='left',on='date')
    train_ = pd.merge(train_,stores,how='left',on='store_nbr')

[ ] test_ = pd.merge(test,HE,how='left',on='date')
    test_ = pd.merge(test_,oil_date,how='left',on='date')
    test_ = pd.merge(test_,stores,how='left',on='store_nbr')
```

Fig 5. Train de archivos

Tras esta unión se hace el reemplazo de los valores NaN .

```
▶ train_['type_x'] = train_['type_x'].fillna("Normal")
  train_['locale'] = train_['locale'].fillna("Normal")
  train_['locale_name'] = train_['locale_name'].fillna("Normal")
  train_['description'] = train_['description'].fillna("Normal")
  train_['transferred'] = train_['transferred'].fillna(False)
  train_.isnull().sum()

[ ] #test_["dcoilwtico"] = test_["dcoilwtico"].fillna(method = "bfill")
    test_["type_x"] = test_["type_x"].fillna("Normal")
    test_["locale"] = test_["locale"].fillna("Normal")
    test_["locale_name"] = test_["locale_name"].fillna("Normal")
    test_["description"] = test_["description"].fillna("Normal")
    test_["transferred"] = test_["transferred"].fillna(False)
    #test_["transactions"] = test_["transactions"].fillna(0)
    test_.isnull().sum()
```

Fig 6 y 7. Llenado de celdas sin información.

En este paso lo que se hizo fue cambiar los tipos de dato que tiene las fechas usando la función `to_datetime` de pandas para volverlos objetos `datetime`.

```
▶ train["date"] = pd.to_datetime(train.date)
  test["date"] = pd.to_datetime(test.date)
  HE["date"] = pd.to_datetime(HE.date)
  oil["date"] = pd.to_datetime(oil.date)
  transactions["date"] = pd.to_datetime(transactions.date)
```

Fig 8. Uso de la función `to_datetime`

Con lo realizado hasta este punto, se tiene la información lista para ser introducida a un modelo que me asigne una probabilidad de acierto en base a lo buscado, la cual continuará siendo desarrollada en el marco de este proyecto.

```

from sklearn.preprocessing import LabelEncoder

train_ = train_.drop(['id'],axis=1)

for column in train_:
    if(train_[column].dtype == object):
        le = LabelEncoder()
        le.fit(train_[column])
        train_[column] = le.transform(train_[column])

test_ = test_.drop(['id'],axis=1)

for column in test_:
    if(test_[column].dtype == object):
        le = LabelEncoder()
        le.fit(test_[column])
        test_[column] = le.transform(test_[column])

```

Fig 9. importación del LabelEncoder

Se realiza la importación de LabelEncoder, con esto buscamos que cualquier columna de tipo objeto dentro tanto del dataset de entrenamiento y del dataset de prueba, se encripta cada variable, siendo asignado un número para representar esa variable dentro de nuestros data set, reemplazando así desde tipo float, int y str.

```

train_xt = train_[train_['date'] <= '2016-12-31'].drop(['sales','date'], axis=1)
train_yt = train_.loc[train_['date'] <= '2016-12-31','sales']

train_xv = train_[train_['date'] > '2016-12-31'].drop(['sales','date'], axis=1)
train_yv = train_.loc[train_['date'] > '2016-12-31','sales']

[ ] train_xt = train_xt.drop(['locale_name','description'], axis=1)
    train_xv = train_xv.drop(['locale_name','description'], axis=1)

[ ] train_xv

[ ] test_ = test_.drop(['date','locale_name','description'], axis=1)
    test_

```

Fig 10. Uso de la función .drop

Lo que se realiza aquí es hacer uso de la función .drop la cual devuelve una copia del dataframe luego de excluir las columnas indicadas.


```
[ ] from sklearn.metrics import mean_squared_error
def rmsle(act, pred):
    value = np.sqrt(mean_squared_error(np.log(act+1), np.log(pred+1)))
    return value

from lightgbm import LGBMRegressor
import lightgbm as lgb

model_lgb = LGBMRegressor(objective='regression')

[ ] train_yt1 = np.log(train_yt+1)
model_lgb.fit(train_xt,train_yt1)

LGBMRegressor(objective='regression')

[ ] pred_log = model_lgb.predict(train_xv)
pred = np.exp(pred_log)-1

[ ] rmse = rmsle(train_yv,pred)
rmse

0.7925858286843345
```

Fig 11. Resolución de la métrica y predicción

Se realiza el cálculo del error logarítmico medio cuadrático usado como métrica, ya que desde el inicio así se estableció, luego se realiza la predicción y finalmente se tiene porcentaje de acierto.

CONCLUSIÓN

Gracias al procesamiento de datos que dimos a los datasets, donde buscamos la organización y llenado de todos los datos faltantes necesarios y la implementación del modelo matemático usado, donde obtuvimos un resultado de 0.7925, mostrando como nuestro modelo tiene un acierto en su predicción bastante elevado al ser casi del 80%, siendo bastante fiable sobre la predicción de las ventas unitarias de miles de artículos vendidos en diferentes tiendas de la empresa situada en Ecuador de nombre “Favorita”, pero sin embargo puede buscarse obtener un resultado más alto y así tener mayor confianza, por lo cual este algoritmo no es la solución más óptima a la competencia.