Predicción de ventas unitarias en productos

Corporación Favorita Grocery Sales Forecasting

Por:
Carlos Esteban Cossio Gonzalez

Materia:
Modelos y simulación de sistemas I

Profesor:
Raul Ramos Pollan

Universidad de Antioquia

Facultad de Ingeniería

Medellín 2023

Introducción

La gestión de la demanda es un proceso clave para las empresas que ofrecen productos o servicios a los clientes, ya que les permite optimizar sus recursos, reducir sus costos y mejorar su competitividad. Predecir la demanda no es una tarea fácil, ya que depende de muchos factores internos y externos, como el precio, la calidad, la publicidad, la estacionalidad, la competencia, el clima, las preferencias de los consumidores, etc. Por lo tanto, se requieren métodos avanzados de análisis de datos y modelado matemático para estimar la demanda futura con precisión y eficiencia. En este contexto, los modelos de machine learning y sus algoritmos son herramientas poderosas que permiten representar y analizar el comportamiento conjuntos de datos complejos bajo diferentes condiciones.

Planteamiento del problema

En este sentido, Corporación Favorita se enfrenta a un problema complejo y relevante: cómo estimar la demanda de más de 200.000 productos en más de 1.500 ubicaciones distribuidas en todo el territorio ecuatoriano. Esta tarea implica manejar grandes volúmenes de datos históricos y actuales, así como considerar las características y preferencias de cada mercado local.

Por lo tanto, el problema que se plantea en este proyecto es: ¿cómo desarrollar un modelo predictivo que permita a Corporación Favorita anticipar y gestionar la demanda de productos en sus numerosas ubicaciones, utilizando como variable objetivo la cantidad de ventas unitarias por producto?

Dataset

El conjunto de datos utilizado para este proyecto proviene de la competencia de Kaggle “Favorita Grocery Sales Forecasting”, que consiste en predecir las ventas unitarias de miles de productos vendidos en diferentes tiendas de Corporación Favorita ubicadas en Ecuador. El conjunto de datos contiene los siguientes archivos:

- **train.csv**: Datos de entrenamiento, que incluyen las ventas unitarias objetivo por fecha, store_nbr e item_nbr y un id único para etiquetar las filas. Las ventas unitarias pueden ser enteras (por ejemplo, una bolsa de papas fritas) o decimales (por ejemplo, 1.5 kg de queso). Los valores negativos de ventas unitarias representan devoluciones de ese producto. La columna onpromotion indica si ese item_nbr estaba en promoción para una fecha y store_nbr específicos. Aproximadamente el 16% de los valores de onpromotion en este archivo son NaN.
- **test.csv**: Datos de prueba, con las combinaciones de fecha, store_nbr e item_nbr que se deben predecir, junto con la información onpromotion. Los datos de prueba tienen un pequeño número de productos que no están contenidos en los datos de entrenamiento. Parte del ejercicio será predecir las ventas de un nuevo producto basado en productos similares. El reparto entre el marcador público y el privado se basa en el tiempo. Todos los productos del reparto público también están incluidos en el reparto privado.
- **stores.csv**: Metadatos de las tiendas, incluyendo ciudad, estado, tipo y cluster. El cluster es un agrupamiento de tiendas similares.
- **items.csv**: Metadatos de los productos, incluyendo familia, clase y perecible. NOTA: Los productos marcados como perecibles tienen un peso de puntuación de 1.25; en caso contrario, el peso es 1.0.
- **transactions.csv**: El recuento de transacciones de ventas para cada combinación de fecha y store_nbr. Sólo se incluye para el período de tiempo de los datos de entrenamiento.
- **oil.csv**: Precio diario del petróleo. Incluye valores durante el período de tiempo tanto de los datos de entrenamiento como de los datos de prueba.

(Ecuador es un país dependiente del petróleo y su salud económica es muy vulnerable a los choques en los precios del petróleo.)

- **holidays_events.csv**: Festivos y eventos, con metadatos. NOTA: Preste especial atención a la columna transferred. Un festivo que se transfiere oficialmente cae en ese día del calendario, pero fue movido a otra fecha por el gobierno. Un día transferido se parece más a un día normal que a un festivo.

Métrica de evaluación

La métrica de evaluación utilizada para este proyecto es el NWRMSLE (Normalized Weighted Root Mean Squared Logarithmic Error), que se define como:

$$NWRMSLE = \sqrt{\frac{\sum_{\{i=1\}}^{n} w_i \left(ln\left(\{\widehat{y}\}\imath + 1\right) - ln(y_i + 1)\right)^2}{\sum_{i=1}^{n} w_i}}$$

Por otra parte, los estadísticos de:

- Error Cuadrático Medio (MSE): Esta métrica proporciona una medida de la magnitud promedio de los errores cuadráticos entre las predicciones y los valores reales. Se calcula tanto en el conjunto de entrenamiento como en el conjunto de prueba.
- Error Absoluto Medio (MAE): Mide la magnitud promedio de los errores absolutos entre las predicciones y los valores reales. Similar al MSE, se calcula tanto en el conjunto de entrenamiento como en el conjunto de prueba.

Variable objetivo

La variable objetivo de este proyecto es la cantidad de ventas unitarias de miles de productos vendidos en diferentes tiendas 'unit_sales'. Esta variable representa el número de unidades que se vendieron de cada producto en cada tienda y en cada fecha. La variable objetivo se encuentra en el archivo train.csv, junto con otras variables como la fecha, el store_nbr, el item_nbr y el onpromotion.

Exploración de datos

En este caso, decidí explorar los datos por cada archivo .csv y analizando cada variable de los dataframe. En el caso del dataframe de entrenamiento, este era demasiado grande así que no pude trabajar con la totalidad de su tamaño.
Además, quiero destacar que los archivos que analicé fueron: test.csv, stores.csv, ítems.csv, holidays_events.csv, transactions.csv, oil.csv y train.csv. Esto porque me parecieron los que más podrían influir en el proyecto y en las futuras decisiones.

Análisis de la variable objetivo

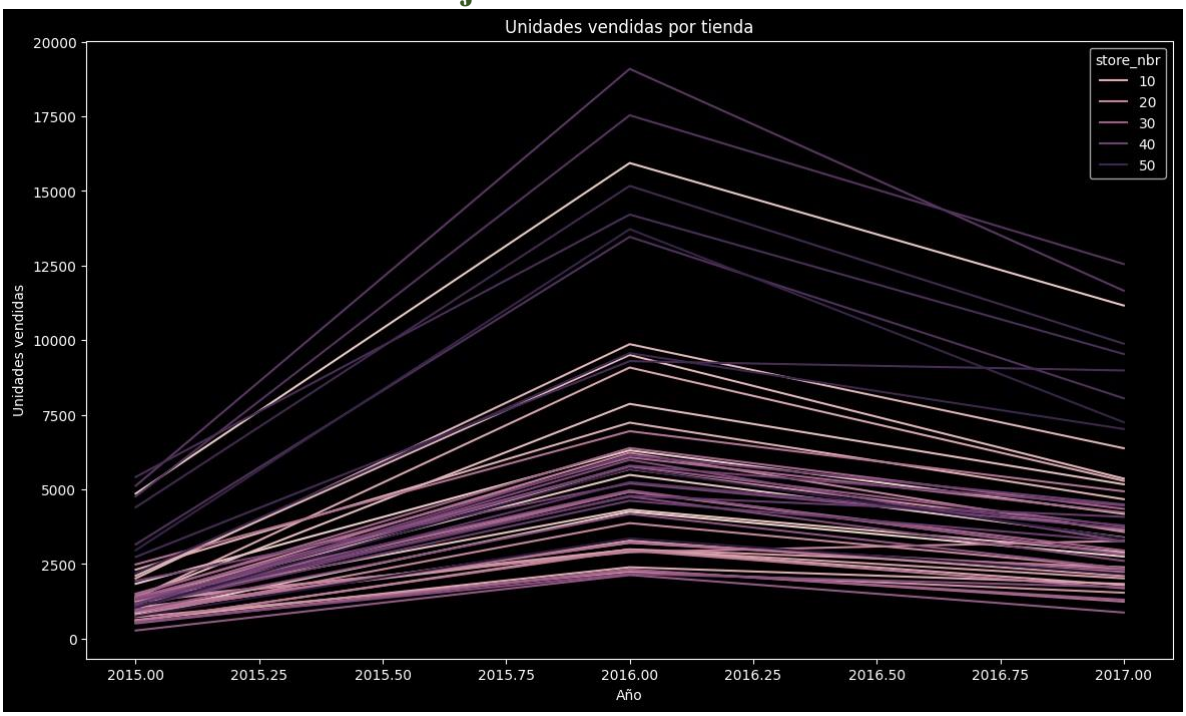

Figura 1. Unidades vendidas por tienda.

En la anterior imagen se puede apreciar que las ventas de productos aumentan en 2015 y este incremento solo duró hasta 2016. En segundo lugar, podemos notar que algunas tiendas son más jóvenes que otras, lo que podría implicar que se abrieron o se incorporaron al conjunto de datos en momentos diferentes.

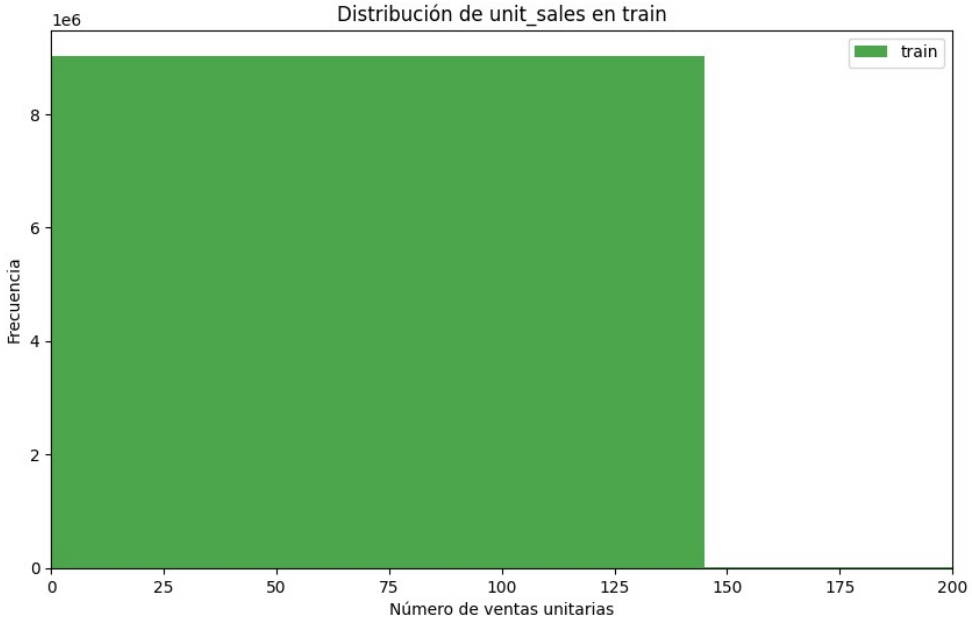

Figura 2. Distribución de la variable unit_sales.

En la figura 2 se muestra la distribución de la variable objetivo. A manera de explicación, el dataframe de entrenamiento fue filtrado para que solo tuviera datos sobre la venta unitaria de productos en el año 2016, con esto dicho, la cantidad de ventas unitarias está dentro de un rango, desde 0 hasta 130 aproximadamente. La mayor cantidad de ventas unitarias se la lleva el producto '2113914' con un número de 198.

Tiendas abiertas por año

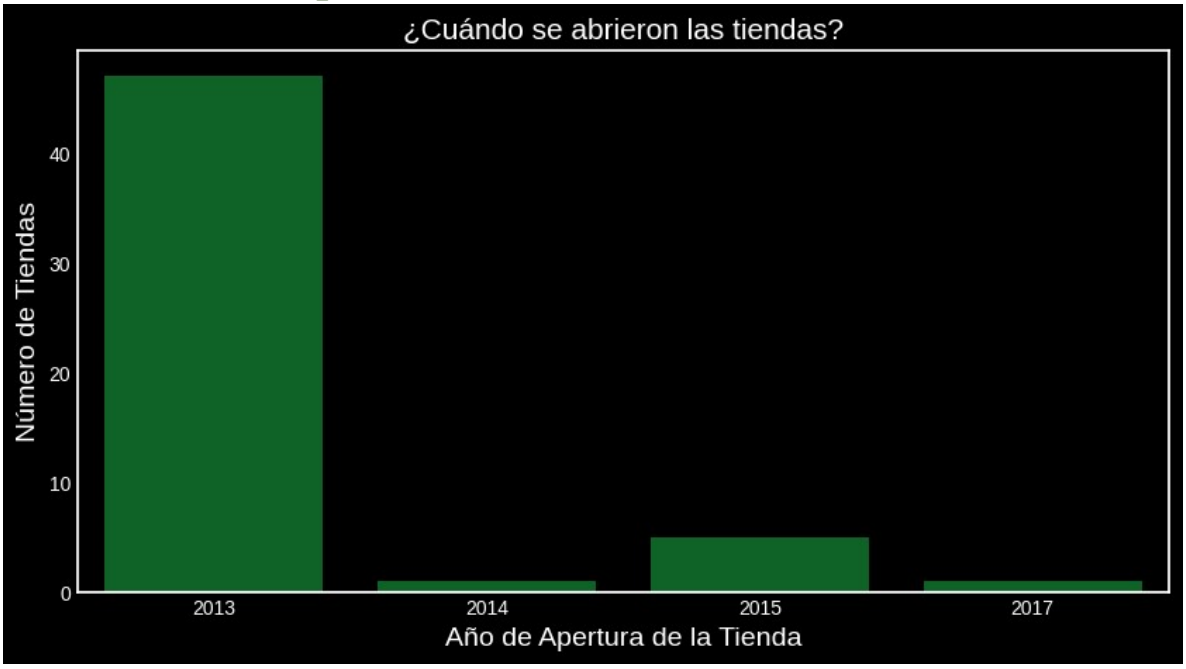

Figura 3. Tiendas abiertas por año

Esta gráfica es importante, en mi caso es porque define la causa de porque filtré finalmente el dataframe de entrenamiento en el año 2016 ya que no se abrieron nuevas tiendas en ese año, por tanto, no influye mucho en el análisis final. Además, a manera de curiosidad

- Más de 40 tiendas se abrieron en el 2013.
- En 2014 y 2017 abrieron 2 tiendas en total.
- Aproximadamente 5 tiendas se abrieron en 2015.

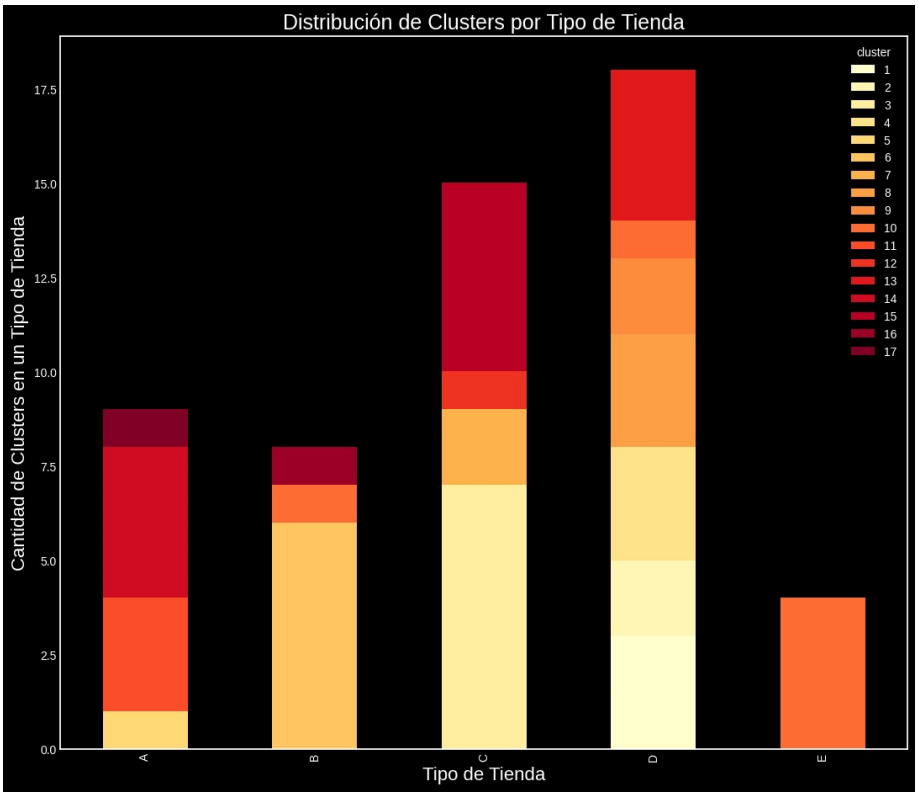

Figura 4. Distribución de clusters y tipo de tiendas

La mayoría de los tipos de tiendas parecen estar clasificados en categorías, especialmente en lo que respecta al tipo de tienda "D". Solo las tiendas de tipo "E" parecen pertenecer al único clúster (grupo) 10. Al pensar en categorías de tiendas, consideraría clasificaciones como tiendas abiertas las 24 horas, tiendas grandes que venden desde comestibles hasta electrodomésticos o tiendas de compra a granel. Pero, no se obtiene más información de los tipos de tiendas y sus clusters.

Clasificación de productos

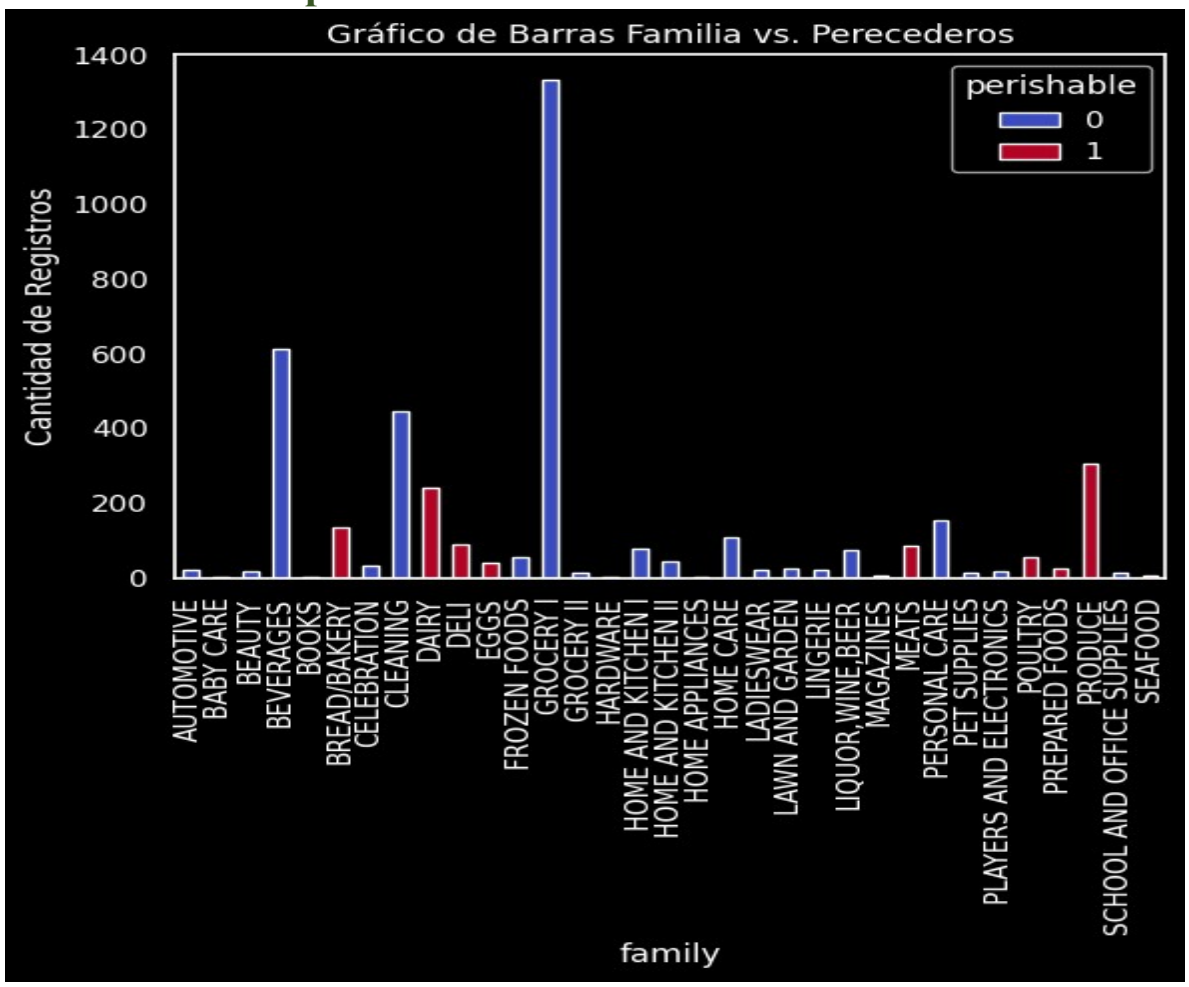

Figura 5. Gráfico de clasificación de los productos

Comestibles, productos de limpieza y bebidas se llevan el podio de productos con más registros/ventas. Algunos productos están clasificados en perecederos o parte de una familia de productos. Por ejemplo, vegetales, panes y periódicos/revistas hacen parte de una familia de productos. Productos para bebé, comida de mar y hardware son productos que se venden muy poco.

Preprocesado de los datos

Se han revisado los diferentes archivos .csv en búsqueda de datos nulos o no asignados. Dentro de esta limpieza de datos, se encontró que el archivo 'oil.csv' fue el único que cuanta con las características de valores 'NaN', en específico, la cantidad fue un 3.5%.

```
[ ]  oil_nan = (oil.isnull().sum() / oil.shape[0]) * 100
     oil_nan

     date          0.000000
     dcoilwtico    3.530378
     dtype: float64
```

Hay un 3.5% de datos faltantes en el archivo (oil.csv)

Figura 6. Búsqueda de datos nulos en oil.csv

En la búsqueda de una comprensión más profunda y precisa de la demanda, se llevó a cabo un proceso para seleccionar tiendas de cada estado ecuatoriano y establecer un periodo de tiempo relevante. El objetivo de prever las ventas unitarias de todos los productos en tiendas específicas durante fechas del año 2016.

state	
Azuay	37
Bolivar	19
Chimborazo	14
Cotopaxi	12
El Oro	40
Esmeraldas	43
Guayas	24
Imbabura	15
Loja	38
Los Rios	31
Manabi	52
Pastaza	22
Pichincha	1
Santa Elena	25
Santo Domingo de los Tsachilas	5
Tungurahua	23

Figura 7. Estado y ID de las tiendas seleccionadas

Por otra parte, se hizo una conversión de fechas en los dataframes respectivos con el motivo de poder trabajar con datos que sean de tipo float y no string. Igualmente se borraron columnas que no son importantes para el análisis del problema, gracias a la exploración de datos se pudo hacer esto.

Pipeline para el flujo de datos

```python
class MergeDataTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Constructor de la clase
        print("Inicializando MergeDataTransformer")

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Unir DataFrames
        train_stores = X[0].merge(X[1], on='store_nbr')
        train_stores_oil = train_stores.merge(X[2], on='date')
        train_stores_oil_items = train_stores_oil.merge(X[3], on='item_nbr')
        train_stores_oil_items_transactions = train_stores_oil_items.merge(X[4], on=['date', 'store_nbr'])
        train_stores_oil_items_transactions_hol = train_stores_oil_items_transactions.merge(X[5], on='date')

        # Copiar DataFrame
        data = train_stores_oil_items_transactions_hol.copy(deep=True)

        # Cambiar booleanos a enteros
        data['onpromotion'] = data['onpromotion'].astype(int)
        data['transferred'] = data['transferred'].astype(int)

        # Cambiar nombres de columnas
        data.rename(columns={'type_x': 'st_type', 'type_y': 'hol_type'}, inplace=True)

        # Eliminar la columna 'id'
        data.drop(['id'], axis=1, inplace=True)

        # Mostrar las primeras filas del DataFrame resultante
        print(data.head())

        # Manipular la columna de fecha
        data['date'] = pd.to_datetime(data['date'])
        data['date'] = data['date'].map(dt.datetime.toordinal)

        return data
```

Figura 8. Clase MergeDataTransformer

Esta clase se encarga de combinar varios DataFrames en uno solo y realizar algunas transformaciones iniciales. El proceso incluye la unión de DataFrames basados en ciertos criterios (como 'store_nbr', 'date', 'item_nbr', etc.), la conversión de valores booleanos a enteros, el cambio de nombres de columnas y la eliminación de la columna 'id'. Además, se muestra una vista previa de las primeras filas del DataFrame resultante. Finalmente, se realiza una manipulación en la columna de fecha, convirtiendo las fechas a un formato ordinal.

```python
class SplitDataTransformer(BaseEstimator, TransformerMixin):
    def __init__(self):
        # Constructor de la clase
        print("Inicializando SplitDataTransformer")

    def fit(self, X, y=None):
        return self

    def transform(self, X):
        # Eliminar la columna 'date' del DataFrame original
        df_without_date = X.drop(['date'], axis=1)

        # Obtener columnas numéricas y categóricas
        all_columns = df_without_date.columns
        numeric_columns = df_without_date._get_numeric_data().columns
        categorical_columns = list(set(all_columns) - set(numeric_columns))

        # Crear DataFrames separados para datos numéricos, categóricos y de fecha
        data_numeric_df = X[numeric_columns]
        data_categorical_df = X[categorical_columns]
        data_date_df = X['date']

        return data_numeric_df, data_categorical_df, data_date_df
```

Figura 9. Clase SplitDataTransformer

Esta clase se encarga de dividir los datos en tres conjuntos distintos: datos numéricos, datos categóricos y datos de fecha. Al principio, se elimina la columna 'date' del DataFrame original. Luego, se identifican las columnas numéricas y categóricas. Finalmente, se crean tres DataFrames separados: uno para datos numéricos, otro para datos categóricos y el último para datos de fecha.

```python
class ProcessData(BaseEstimator, TransformerMixin):
    def __init__(self):
        print("Inicializando ProcessData")

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        # Datos numéricos
        # Imputar valores nulos en atributos numéricos
        imputer = SimpleImputer(strategy="mean", copy=True)
        num_imputed = imputer.fit_transform(X[0])
        data_num_df = pd.DataFrame(num_imputed, columns=X[0].columns, index=X[0].index)

        # Aplicar escalado estándar
        scaler = StandardScaler()
        num_scaled = scaler.fit_transform(data_num_df)
        data_num_df = pd.DataFrame(num_scaled, columns=X[0].columns, index=X[0].index)

        # Datos categóricos
        # One-hot encoder
        cat_encoder = OneHotEncoder(sparse=False)
        data_cat_1hot = cat_encoder.fit_transform(X[1])

        # Convertir a DataFrame con n*99, donde n es el número de filas y 99 es el número de categorías
        data_cat_df = pd.DataFrame(data_cat_1hot, columns=cat_encoder.get_feature_names_out(), index=X[1].index)

        return data_num_df, data_cat_df, X[2]
```

Figura 10. Clase ProcessData

Esta clase se encarga de procesar los datos numéricos y categóricos por separado. Para los datos numéricos, utiliza un imputador para rellenar los valores nulos con la media y luego realiza un escalado estándar. Para los datos categóricos, utiliza un codificador One-Hot para convertir las categorías en columnas binarias. La salida de esta clase son dos DataFrames, uno para datos numéricos y otro para datos categóricos, junto con los datos no transformados.

```python
class JoinDataFrames(BaseEstimator, TransformerMixin):
    def __init__(self):
        print("Inicializando JoinDataFrames")

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        # Unir DataFrames numéricos
        data_df = X[0].join(X[1])
        data_df = data_df.join(X[2])

        return data_df
```

Figura 11. Clase JoinDataFrame

Esta clase se encarga de unir los DataFrames numéricos y categóricos obtenidos después de las transformaciones. Realiza la unión basada en el índice. La salida de esta clase es un único DataFrame resultante de la unión.

```
# Definir la pipeline
pipeline = Pipeline([
    ('merge_data', MergeDataTransformer()),
    ('split_data', SplitDataTransformer()),
    ('process_data', ProcessData()),
    ('join_data', JoinDataFrames())
])
```

Figura 12. Definición del Pipeline

Se define una canalización (pipeline) que combina estas tres clases en un flujo de trabajo coherente. Cada paso de la canalización realiza una transformación específica en los datos, creando una estructura organizada y modular para el procesamiento de datos.

Métodos supervisados

Evaluaré el rendimiento de los diferentes modelos utilizando los estadísticos (MAE, MSE) tanto para los datos de entrenamiento como para los de prueba. Para entrar en contexto el dataframe que llamé 'train_store_state' es el que, recordando palabras anteriores, filtré para que tomara datos del año 2016.

```
# Seleccionar aleatoriamente 70.000 filas de 'train_store_state'
train_sample = train_store_state.sample(n=70000)
```

Figura 13. Código para seleccionar 70mil registros

En el proceso de selección del modelo para la predicción, se evaluaron varios algoritmos de regresión utilizando una partición del conjunto de entrenamiento, que comprendía 70,000 datos, seleccionados al azar. El criterio principal para la elección del modelo fue minimizar el Error Absoluto Medio (MAE), que representa la magnitud promedio de los errores entre las predicciones del modelo y los valores reales.

```
def checkModelPerformance(model):
    model.fit(x_train.values, y_train.values)

    pred = model.predict(x_test.values)

    print("Media cuadrada del error: ", np.sqrt(mean_squared_error(y_test.values, pred)))
    print("Media absoluta del error: ", np.sqrt(mean_absolute_error(y_test.values, pred)))
```

Figura 14. Función para evaluar modelos

Los resultados de la evaluación son los siguientes, tomada de una muestra aleatoria de 70000 registros del dataframe de entrenamiento:

1. **Linear Regression:**
- Error Cuadrático Medio: 5590491.11
- Error Absoluto Medio: 313.57

2. **Lasso Regression (alpha=0.1):**
- Error Cuadrático Medio: 1.04
- Error Absoluto Medio: 0.68

3. **ElasticNet Regression:**
- Error Cuadrático Medio: 1.15
- Error Absoluto Medio: 0.77

4. **Ridge Regression (alpha=1.0):**
- Error Cuadrático Medio: 0.96
- Error Absoluto Medio: 0.61

5. **Random Forest Regression (Random State=42):**
- Error Cuadrático Medio: 0.93
- Error Absoluto Medio: 0.52

Basándonos en la evaluación del MAE, se observa que el modelo que presenta el menor Error Absoluto Medio es el Random Forest Regression con un valor de 0.52. Este resultado indica que, en promedio, las predicciones de este modelo tienen una magnitud de error más baja en comparación con los demás algoritmos evaluados.

Métrica para modelos

En el proceso de evaluación de modelos utilizando la métrica Normalized Weighted Root Mean Squared Logarithmic Error (NWRMSLE), se consideró la importancia de la variabilidad en la precisión de las predicciones para productos perecederos y no perecederos.

```python
# Lista de modelos
modelos = [
    LinearRegression(),
    Lasso(alpha=0.1),
    ElasticNet(),
    Ridge(alpha=1.0),
    RandomForestRegressor(random_state=42)
]

# Entrenar y evaluar cada modelo
for modelo in modelos:
    # Entrenar el modelo
    modelo.fit(x_train, y_train)

    # Hacer predicciones
    y_pred = modelo.predict(x_test.values)

    # Calcular NWRMSLE
    weights_dict = {item_nbr: 1.25 if perishable == 1 else 1.00 for item_nbr, perishable in zip(x_test['item_nbr'], x_test['perishable'])}
    nwrmsle = calculate_nwrmsle(y_test['unit_sales'].values, y_pred, weights_dict)

    # Imprimir nombre del modelo y NWRMSLE
    print(f"{type(modelo).__name__}: NWRMSLE = {nwrmsle}\n")
```

Figura 15. Aplicando NWRMSLE

```python
def calculate_nwrmsle(y_true, y_pred, weights_dict):
    # y_pred es un 2D array quiero seleccionar la primera columna
    y_pred_log = np.log1p(y_pred[:, 0])

    # Aplicar logaritmo natural a los valores reales
    y_true_log = np.log1p(y_true)

    # Calcular el error cuadrático medio ponderado
    squared_errors = (y_true_log - y_pred_log) ** 2

    # Utilizar solo la columna de 'unit_sales' para weights_dict
    weighted_squared_errors = [squared_errors[i] * weights_dict[item_nbr] for i, item_nbr in enumerate(weights_dict)]

    # Calcular la métrica NWRMSLE
    nwrmsle = np.sqrt(np.sum(weighted_squared_errors) / len(y_true))

    return nwrmsle
```

Figura 16. Función para calcular NWRMSLE

Siguiendo con la muestra anterior, los resultados obtenidos son los siguientes:

1. **Linear Regression:**
    - NWRMSLE = 0.3564
2. **Lasso Regression (alpha=0.1):**
    - NWRMSLE = 0.3678
3. **ElasticNet Regression:**
    - NWRMSLE = 0.3678
4. **Ridge Regression (alpha=1.0):**
    - NWRMSLE = 0.3560
5. **Random Forest Regression (Random State=42):**
    - NWRMSLE = 0.3748

El NWRMSLE es una métrica que pondera de manera efectiva la precisión del modelo, considerando las particularidades de productos perecederos y no perecederos. Los modelos de regresión lineal (Linear Regression y Ridge

Regression) muestran un rendimiento similar y mejor en comparación con los modelos de regresión Lasso, ElasticNet y RandomForestRegressor.

Hiperparametros de los modelos

El proceso de optimización de hiperparámetros se realizó utilizando el algoritmo RandomForestRegressor, que emplea múltiples árboles de decisión para realizar predicciones. El objetivo principal fue encontrar la combinación óptima de hiperparámetros que maximizara el rendimiento del modelo en términos de la métrica neg_root_mean_squared_error. Se utilizaron dos conjuntos de hiperparámetros para la búsqueda en la cuadrícula.

Los conjuntos de hiperparámetros evaluados fueron los siguientes:

1. Primer conjunto de hiperparámetros:
    - Número de estimadores (n_estimators): [3, 10, 30]
    - Número máximo de características a considerar para la división de un nodo (max_features): [2, 4, 6, 8]
2. Segundo conjunto de hiperparámetros:
    - Bootstrap: [False] (se deshabilita el muestreo bootstrap)
    - Número de estimadores (n_estimators): [3, 10]
    - Número máximo de características a considerar para la división de un nodo (max_features): [2, 3, 4]

La búsqueda en la cuadrícula se realizó mediante validación cruzada con 5 divisiones (cv=5), y la métrica utilizada fue el error cuadrático medio negativo (neg_root_mean_squared_error).

El mejor conjunto de hiperparámetros identificado por la búsqueda en la cuadrícula fue:

- Número máximo de características (max_features): 8
- Número de estimadores (n_estimators): 30

Estos hiperparámetros se utilizarán para configurar un nuevo modelo RandomForestRegressor, que se espera que ofrezca un rendimiento mejorado en comparación con la configuración predeterminada. La optimización de hiperparámetros ha permitido ajustar el modelo RandomForestRegressor para adaptarse mejor a los datos específicos del conjunto de entrenamiento, mejorando así su capacidad predictiva.

Retos y condiciones para desplegar el modelo

En cuanto a llevar el modelo a producción, este análisis carece de la profundidad en el análisis de curvas de aprendizaje, se recomienda realizar un análisis más detallado de las curvas de aprendizaje para evaluar la capacidad del modelo para generalizar a datos no vistos. Esto podría incluir la variación de parámetros clave y

la evaluación del rendimiento en diferentes conjuntos de entrenamiento y prueba. Además, explorar modelos supervisados más avanzados y técnicas de optimización podría ser beneficioso para mejorar aún más la precisión del pronóstico.

Retos

Gran Variedad de Productos: Corporación Favorita opera con más de 200,000 productos, lo que introduce complejidad en el modelado. La gran cantidad de SKU (Stock Keeping Units) puede requerir estrategias específicas para lidiar con la variabilidad en las demandas de diferentes productos.

Nuevas Ubicaciones y Productos: La apertura de nuevas ubicaciones y la introducción de nuevos productos añade un componente dinámico al desafío. El modelo debe ser lo suficientemente flexible para adaptarse a estos cambios sin requerir ajustes manuales frecuentes.

Condiciones de Despliegue:

Interfaz de Usuario Intuitiva: El modelo debe desplegarse con una interfaz de usuario que sea fácil de entender y utilizar por parte del personal de Corporación Favorita. Debería proporcionar visualizaciones claras de los pronósticos y ser fácil de interpretar.

Mantenimiento Continuo: Dado que el entorno comercial está en constante cambio, se requerirá un plan de mantenimiento continuo. Esto puede incluir actualizaciones periódicas del modelo, reentrenamiento con nuevos datos y ajustes según la evolución de las operaciones de la tienda.

Conclusiones

• Este proyecto de curso ha sido fundamental para mi comprensión de la inteligencia artificial y su aplicación práctica en entornos empresariales.

• A través del análisis e implementación de diversos modelos predictivos, he adquirido una comprensión más profunda de cómo los algoritmos pueden abordar problemas complejos.

• La focalización en la predicción de ventas para Corporación Favorita proporcionó una visión práctica de los desafíos en la gestión de inventarios y la satisfacción del cliente.

• Trabajar con conjuntos de datos del mundo real me permitió enfrentar problemas como la variedad de productos, nuevas ubicaciones y estacionalidad, apreciando la complejidad de estos escenarios.

• Reconozco la necesidad de explorar áreas que requieren una investigación más profunda, incluyendo un análisis detallado de la generalización y rendimiento de los modelos en diversas condiciones.

• La implementación de técnicas más avanzadas y la consideración de interacciones entre productos podrían mejorar la precisión de los pronósticos.