Universidad Nacional Autónoma de México
Facultad de Ingeniería
División de Ingeniería Eléctrica
Ingeniería en Computación

Práctica 3.2
"Clasificación"

Integrantes:
- Castelan Ramos Carlos
- Reyes Esquivel Ana Karen
- Rocandio Montiel Jesús Alexis
- Romero López Alexis Uriel

Materia: Minería de Datos
Grupo: 03
Semestre: 2024-1

Fecha de entrega: 05 de octubre2023

Práctica 3.2 – Clasificación

Instrucciones.

Realizar una práctica de Minería de datos utilizando los algoritmos de clasificación y los datos de Transacciones de ventas que se enviaron anteriormente.

Desarrollo:

Para la siguiente actividad se realizaron varios ejemplos, implementados diferentes modelos de clasificación, donde las variables objetivas y las variables dependientes fueron elegidas a partir de las características por las cuales trabajan los modelos de clasificación.

Caso de Estudio:

- Analizar la cantidad de producto vendida, de acuerdo con el precio y la antigüedad de los clientes.
- Elección predictiva del sector de acuerdo con la descripción y marca en las ventas realizadas.
- Clasificación del sector de acuerdo a precio y cantidad de ventas.

Árbol de decisiones.

Variable objetivo: Cantidad
Variables independientes: Precio y Antigüedad

Algoritmo:
Disponible en:
https://colab.research.google.com/drive/1CjC2ksbS9ec_QJpayGa4ASsNGhn4VWI3?usp=sharing

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
import matplotlib.pyplot as plt
```

Figura #1

- **pandas (import pandas as pd):** pandas es una biblioteca de Python que se utiliza para manipular y analizar datos de manera eficiente. Se usa comúnmente para cargar y preprocesar datos, como leer archivos CSV, crear DataFrames para trabajar con datos tabulares y realizar operaciones de limpieza y transformación de datos.
- **scikit-learn (from sklearn.tree import DecisionTreeClassifier, plot_tree):** scikit-learn es una biblioteca de aprendizaje automático en Python que proporciona herramientas para construir y entrenar modelos de aprendizaje automático. En este caso, has importado DecisionTreeClassifier, que es una implementación de árboles de decisión para clasificación, y plot_tree, que se utiliza para visualizar árboles de decisión.
- **train_test_split (from sklearn.model_selection import train_test_split):** Esta función de scikit-learn se utiliza para dividir un conjunto de datos en conjuntos de entrenamiento y prueba. Ayuda a separar los

datos que se utilizarán para entrenar un modelo y los datos que se utilizarán para evaluar su rendimiento.

- **accuracy_score (from sklearn.metrics import accuracy_score):** accuracy_score es una métrica de evaluación que se utiliza para medir la precisión de un modelo de clasificación. Compara las predicciones del modelo con los valores reales y calcula la proporción de predicciones correctas.
- **classification_report (from sklearn.metrics import classification_report):** classification_report es una función que genera un informe detallado de las métricas de clasificación, incluyendo precisión, sensibilidad, F1-score y más. Proporciona una visión completa del rendimiento del modelo en términos de clasificación.
- **matplotlib.pyplot (import matplotlib.pyplot as plt):** matplotlib es una biblioteca de visualización en Python. pyplot es un módulo de matplotlib que se utiliza para crear gráficos y visualizaciones. En tu código, se utiliza para visualizar el árbol de decisión generado por plot_tree.

```
# Carga los datos desde el archivo CSV
data = pd.read_csv("transacciones.csv", encoding='latin1')

X = data[['Precio', 'Antiguedad']]  # Las características
y = data['Cantidad']                # La variable objetivo

# Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
```

Figura #2

En esta parte del código cargamos los datos y los asignamos a las variables correspondientes (X,Y), así como para generar una división entre los datos de entrenamiento y prueba.

```
# Crea un modelo de árbol de decisión
modelo_arbol = DecisionTreeClassifier()

# Ajusta el modelo a los datos de entrenamiento
modelo_arbol.fit(X_train, y_train)

# Realiza predicciones en el conjunto de prueba
y_pred = modelo_arbol.predict(X_test)
```

Figura #3

Posteriormente, vienen las instrucciones para preparar los datos y poderlos procesar con el árbol de decisiones.

```python
# Calcula la precisión del modelo
precision = accuracy_score(y_test, y_pred)
print(f'Precisión: {precision:.2f}')

# Calcula informe de clasificación que incluye precisión, sensibilidad, F1-score, etc.
informe_clasificacion = classification_report(y_test, y_pred)
print("Informe de Clasificación:")
print(informe_clasificacion)
```

Figura #4

Obtenemos la presición y un informe general de los resultados que arrojó el algoritmo.

```python
# Define un diccionario para mapear las etiquetas numéricas a cadenas de texto
class_labels = {
    0: "Clase0",
    1: "Clase1",
    3: "Clase3",
    2: "Clase2",
    6: "Clase6",
    4: "Clase4",
    7: "Clase7",
    5: "Clase5",
    15: "Clase15",
    25: "Clase25",
    20: "Clase20",
    10: "Clase10",
    12: "Clase12",
    8: "Clase8",
    23: "Clase23",
    24: "Clase24",
    13: "Clase13",
    130: "Clase130",
    9: "Clase9",
    57: "Clase57",
    18: "Clase18",
    11: "Clase11",
    31: "Clase31",
    40: "Clase40",
    42: "Clase42",
    30: "Clase30",
    58: "Clase58",
    14: "Clase14",
    26: "Clase26",
    28: "Clase28",
    19: "Clase19",
    60: "Clase60",
    16: "Clase16",
    45: "Clase45",
    36: "Clase36",
    50: "Clase50",
    140: "Clase140",
    17: "Clase17",
    35: "Clase35",
    75: "Clase75",
    22: "Clase22",
    21: "Clase21",
    34: "Clase34",
    165: "Clase165",
    47: "Clase47"

}
```

Figura #5

Utilizamos scikit-learn para mostrar el árbol de decisión, este modelo de árbol de decisión utiliza valores numéricos como etiquetas de clase por defecto y la biblioteca graphviz espera cadenas de texto (str) para las etiquetas de clase. Por lo tanto, definimos un diccionario de clases que mapea las etiquetas numéricas a cadenas de texto.

```python
# Visualiza el árbol de decisión y usa el diccionario de etiquetas
plt.figure(figsize=(15, 10))
plot_tree(modelo_arbol, feature_names=X.columns, class_names=class_labels, filled=True, rounded=True)
plt.title("Árbol de Decisión")
plt.show()
```

Figura #6

De esta manera visualizamos el árbol.

- Árbol completo:

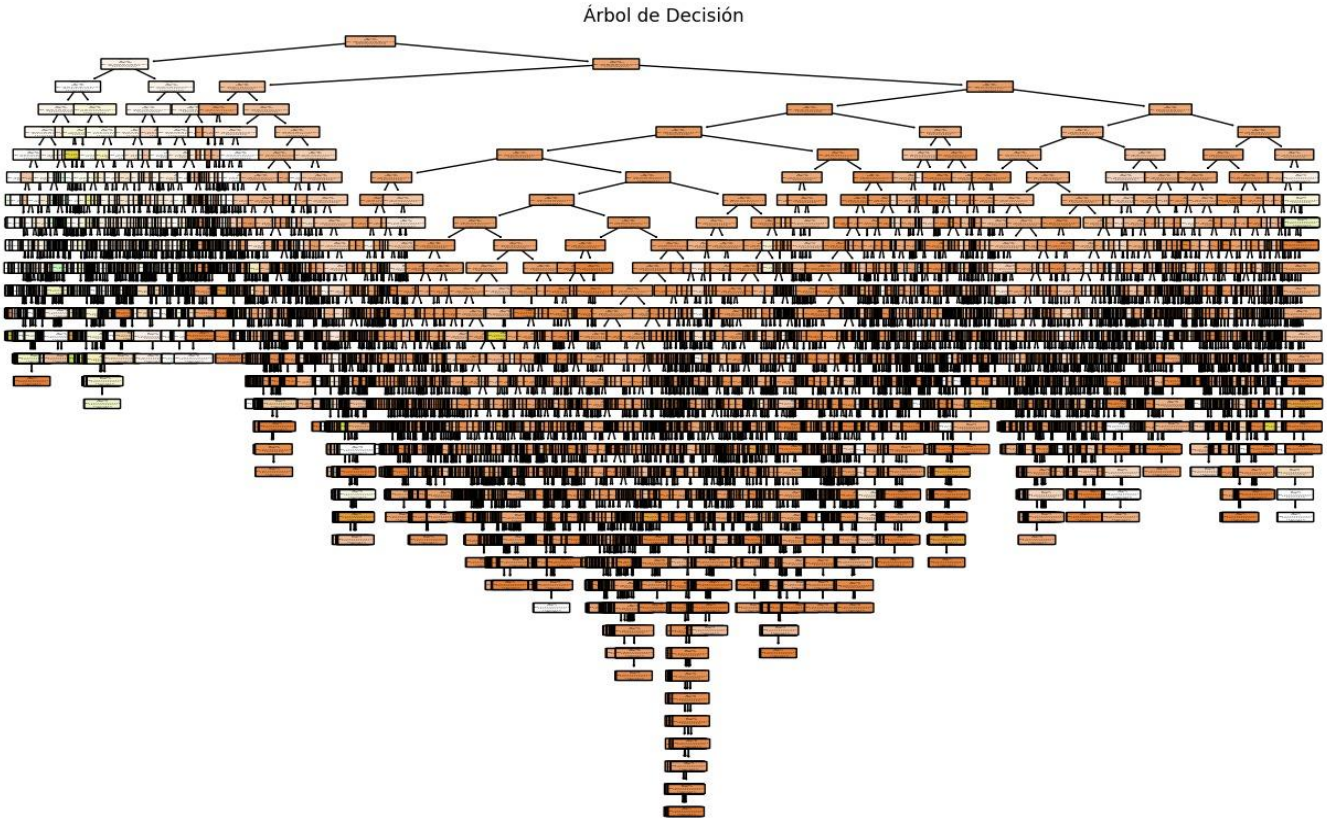

Figura #7

Para obtener una respuesta más clara decidismo segmentar por año:

- 2014

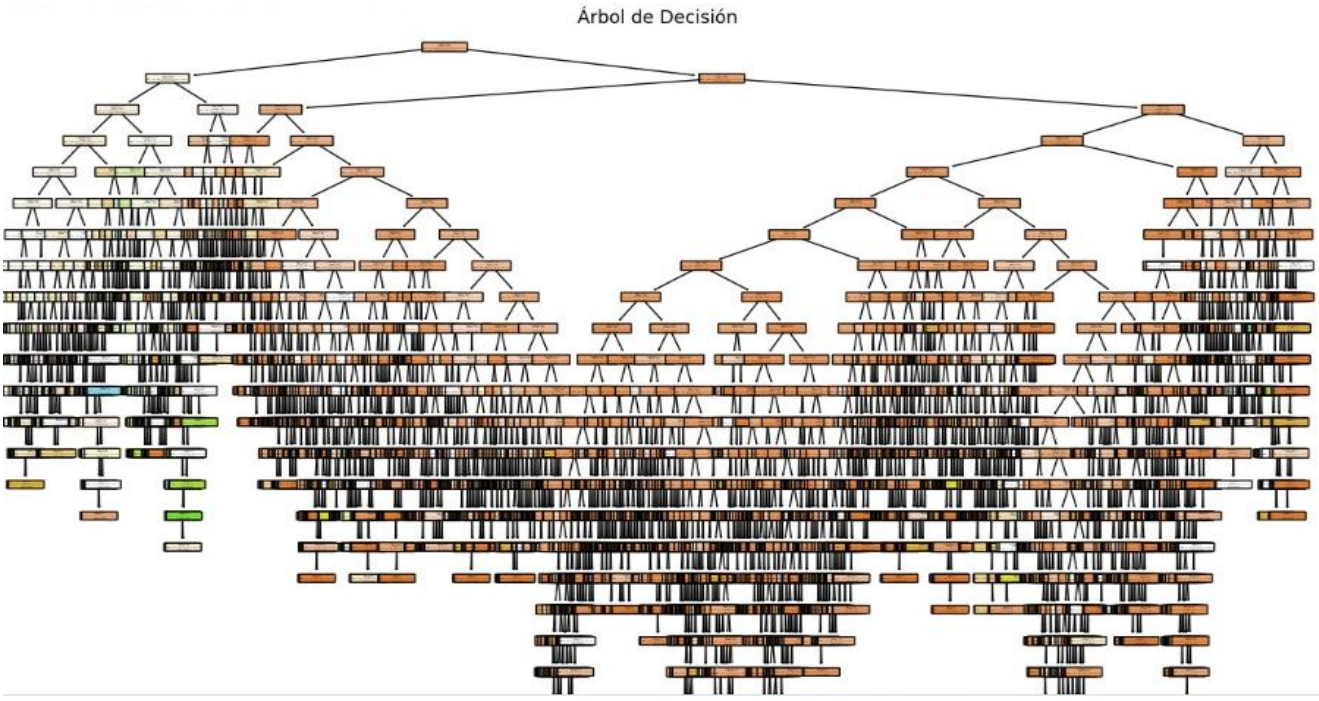

Figura #8

- 2015

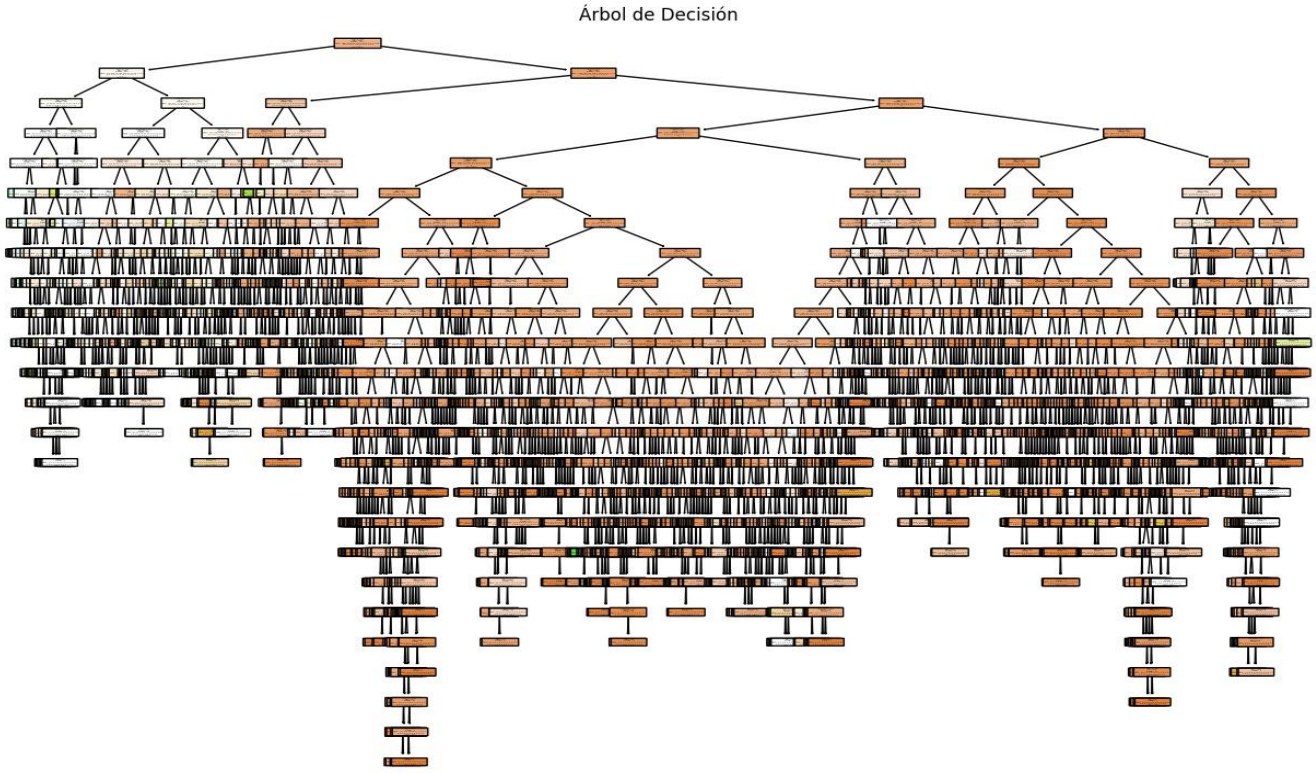

Figura #9

- 2016

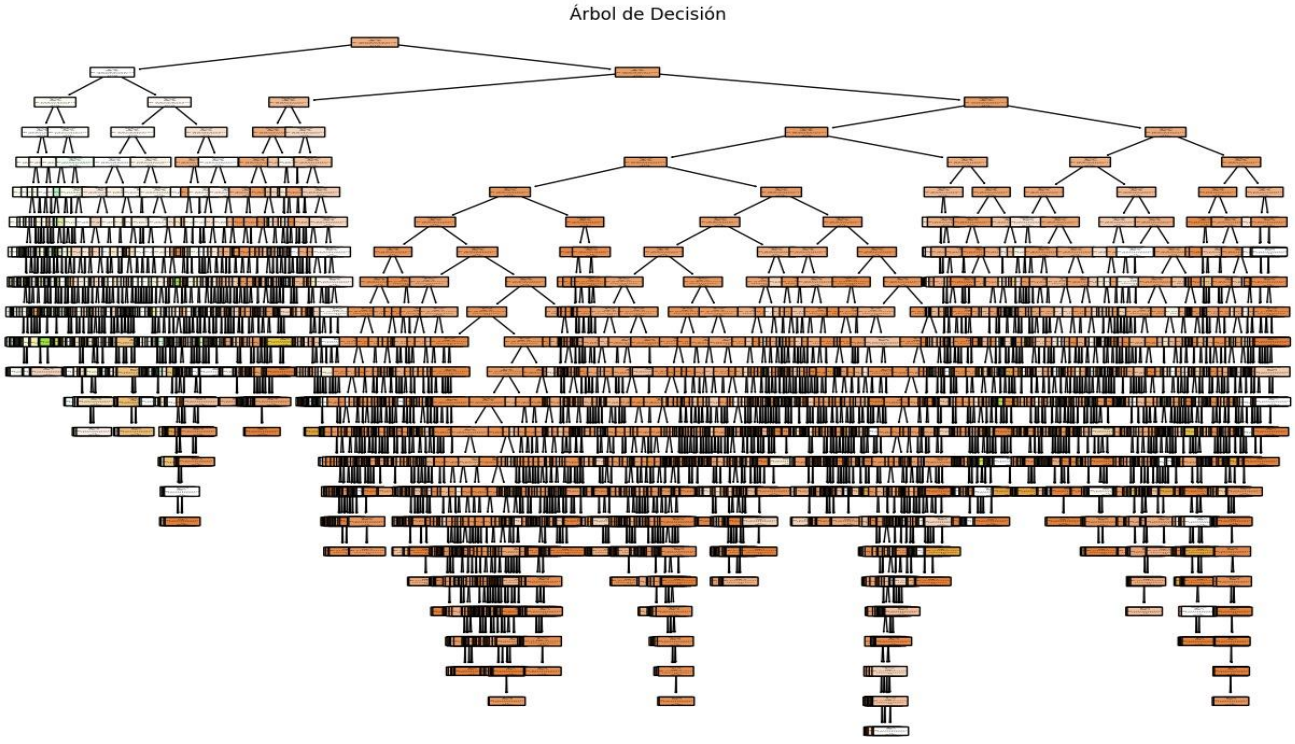

Figura #10

Para realizar la segmentación, basto con delimitar que solo queríamos el año correspondiente de la columa "Año".

```
data_2016 = data[data['Año'] == 2016]

X = data_2016[['Precio', 'Antiguedad']]  # Las características
y = data_2016['Cantidad']                # La variable objetivo
```

Figura #11

A pesar de esto, la cantidad de datos sigue siendo mucha entonces decidimos hacer un corte en los primeros 10 registro de cada año, así poder observar y entender mejor qué es lo que arroja el algoritmo.

- 2014

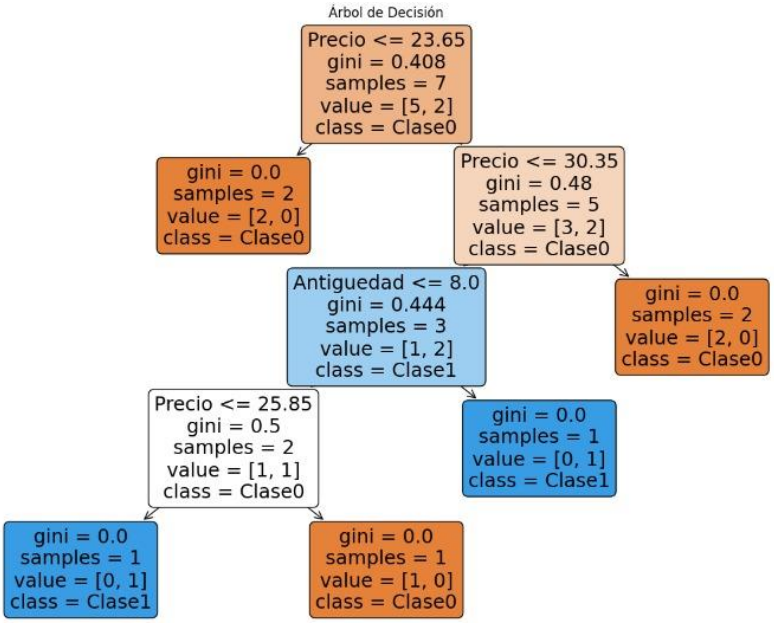

Figura #12

- 2015

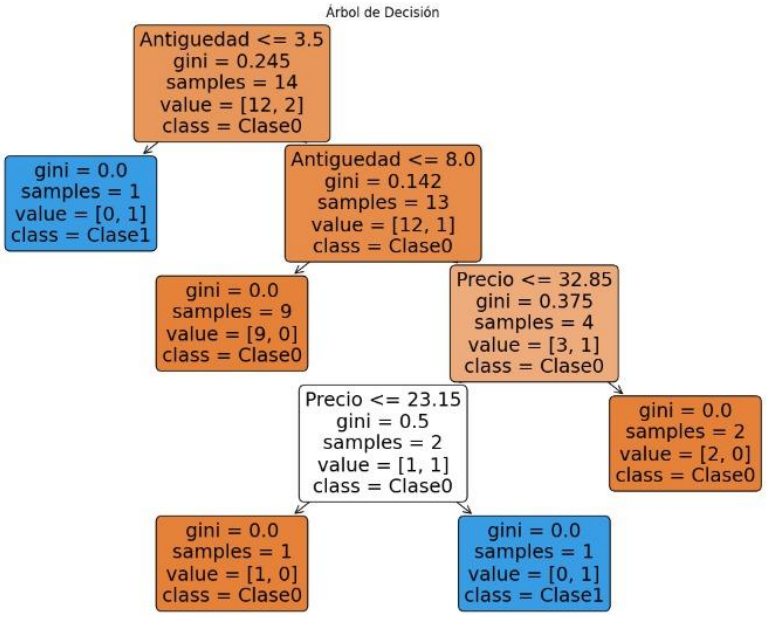

Figura #13

- 2016

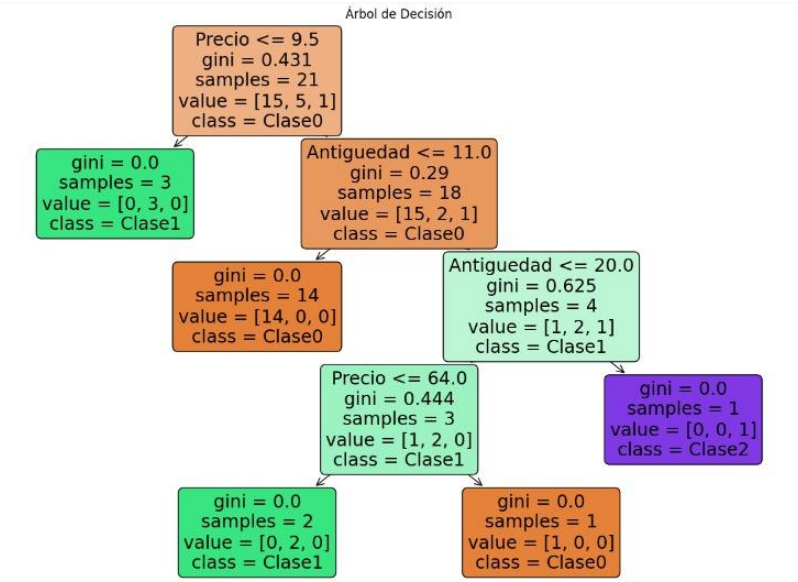

Figura #14

1. Gini (Índice de Gini):
   - El índice de Gini es una medida de impureza que se utiliza en los árboles de decisión para evaluar la pureza de un nodo.
   - En cada nodo del árbol, el índice de Gini se utiliza para calcular la impureza de las muestras en ese nodo.
   - Cuanto más bajo sea el valor del índice de Gini, más puro o homogéneo es el nodo. Un índice de Gini igual a 0 indica que todas las muestras en el nodo pertenecen a una sola clase.
   - Un árbol de decisión busca dividir los nodos de manera que el índice de Gini disminuya en cada división, lo que resulta en nodos más puros.
2. Samples (Muestras):
   - La variable "samples" en un nodo se refiere al número total de muestras (ejemplos de datos) que llegan a ese nodo en particular durante la construcción del árbol.
   - Por ejemplo, si un nodo tiene 100 muestras, significa que 100 ejemplos de datos llegaron a ese nodo antes de que se tomara una decisión sobre cómo dividirlo.
3. Value (Valor):
   - La variable "value" en un nodo indica cómo se distribuyen las muestras en términos de clases en ese nodo.
   - Es una lista que muestra cuántas muestras pertenecen a cada clase en ese nodo en particular.
   - Por ejemplo, si tienes un nodo con "value" igual a [30, 70], significa que en ese nodo hay 30 muestras de una clase y 70 muestras de otra clase.
4. Class (Clase):
   - La variable "class" en un nodo se refiere a la clase que se asigna a ese nodo en función de las muestras que llegan a ese nodo y las decisiones tomadas en el árbol.
   - En muchos árboles de decisión, el nodo final (hoja) se etiqueta con la clase más común entre las muestras en ese nodo.

Naive Bayes.
Variable objetivo: Sector.
Variables independientes: Descripción, Marca

Algoritmo:
Disponible en https://colab.research.google.com/drive/1cu9BPZZGM7g-gpmWCJY5uMTaWIVOQMl7#scrollTo=70xPj8y9Gar2

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.naive_bayes import CategoricalNB
from sklearn.metrics import confusion_matrix, accuracy_score
```

Figura #15

Para este modelo de clasificación hicimos uso de las mismas bibliotecas agregando la de sklearn.naive_bayes para poder implementar el modelo mismo.

```
# Cargar los datos desde un archivo CSV
data = pd.read_csv("transacciones.csv", encoding='latin1')

X = data[['Descripcion', 'Marca']] #Columnas categóricas
y = data['Sector'] #Target a predecir
```

Figura #16

Cargamos los datos del archivo de transacciones de ventas y enseguida definimos las variables independientes de las cuales definiremos los valores predictivos de nuestra variable objetivo sector.

```
# Codificar las características categóricas usando LabelEncoder
label_encoder = LabelEncoder()
X_encoded = X.apply(label_encoder.fit_transform)
```

Figura #17

Se utiliza la clase LabelEncoder de scikit-learn para codificar las características categóricas en el DataFrame X. Esto es necesario porque algunos algoritmos de aprendizaje automático, como el modelo Naive Bayes categórico que se usa aquí, requieren que las características sean numéricas en lugar de categóricas. El método fit_transform del LabelEncoder ajusta y transforma las características, reemplazando las categorías con valores enteros.

```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_encoded,
y,test_size=0.3, random_state=42)
```

Figura #18

Al igual que en el código anterior, se utiliza la función train_test_split de scikit-learn para dividir los datos en conjuntos de entrenamiento (X_train, y_train) y prueba (X_test, y_test). El 30% de los datos se reserva para pruebas, y se establece una semilla aleatoria (random_state=42) para asegurar la reproducibilidad de la división.

```
# Crear y ajustar el modelo Naive Bayes categórico
naive_bayes = CategoricalNB()
naive_bayes.fit(X_train, y_train)
```

Figura #19

Se crea un modelo Naive Bayes categórico utilizando la clase CategoricalNB de scikit-learn y se ajusta (entrena) con los datos de entrenamiento codificados (X_train, y_train). Este tipo de modelo es adecuado cuando se tienen características categóricas.

```
# Realizar predicciones en el conjunto de prueba
y_pred = naive_bayes.predict(X_test)
```

Figura #20

Se utilizan las características codificadas del conjunto de prueba (X_test) para hacer predicciones utilizando el modelo Naive Bayes categórico. Las predicciones se almacenan en y_pred.

```
# Calcular la precisión del modelo
accuracy = accuracy_score(y_test, y_pred)
print(f'Precisión del modelo Naive Bayes: {accuracy:.2f}')
```

Figura #21

Se calcula la precisión del modelo comparando las etiquetas verdaderas (y_test) con las etiquetas predichas (y_pred) utilizando la función accuracy_score de scikit-learn. La precisión es una métrica que mide la proporción de predicciones correctas en el conjunto de prueba.

```
# Crear una matriz de confusión para evaluar el rendimiento del modelo
confusion = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión:')
print(confusion)
```

Figura #22

Se imprime la precisión del modelo Naive Bayes categórico en formato decimal.

```
# Visualizar los resultados (matriz de confusión)
plt.figure(figsize=(8, 6))
plt.imshow(confusion, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.xlabel('Etiquetas Predictivas')
plt.ylabel('Etiquetas Verdaderas')
plt.title('Matriz de Confusión')
plt.show()
```

Figura #23

Conclusiones:

Hemos implementado una matriz de confusión la cual ha generado 12 clases de filas y columnas, en donde las filas representan las clases verdaderas o reales y las columnas son las clases predichas por el modelo. Además, la precisión del modelo de clasificación fue del 100%.

En donde:

- 270 en la celda (1,1): Esto significa que 270 instancias de la clase 1 fueron correctamente clasificadas como clase 1 por el modelo.
- 6189 en la celda (2,2): Indica que 6,189 instancias de la clase 2 fueron correctamente clasificadas como clase 2 por el modelo.
- 1060 en la celda (3,3): Esto significa que 1,060 instancias de la clase 3 fueron correctamente clasificadas como clase 3 por el modelo.
- 39 en la celda (3,8): Indica que 39 instancias de la clase 3 fueron incorrectamente clasificadas como clase 8 por el modelo.
- 609 en la celda (5,5): Esto significa que 609 instancias de la clase 5 fueron correctamente clasificadas como clase 5 por el modelo.
- 84 en la celda (6,6): Indica que 84 instancias de la clase 6 fueron correctamente clasificadas como clase 6 por el modelo.
- 2174 en la celda (12,12): Esto significa que 2,174 instancias de la clase 12 fueron correctamente clasificadas como clase 12 por el modelo.

Es entonces que decimos que la clasificación de acuerdo con las características y marca, definirán en un 100% el sector al que pertenece el producto.

```
Precisión del modelo Naive Bayes: 1.00
Matriz de Confusión:
[[ 270    0    0    0    0    0    0    0    0    0    0    0]
 [   0 6189    0    0    0    0    0    0    0    0    0    0]
 [   0    0 1060    0    0    0    0   39    0    0    0    0]
 [   0    0    0  482    0    0    0    0    0    0    0    0]
 [   0    0    0    0  609    0    0    0    0    0    0    0]
 [   0    0    0    0    0   84    0    0    0    0    0    0]
 [   0    0    0    0    0    0  231    0    0    0    0    0]
 [   0    0    0    0    0   14    0  580    0    0    2    0]
 [   0    0    0    0    0    0    0    0  187    4    0    0]
 [   0    0    0    0    0    0    0    0    0 5219    0    0]
 [   0    0    0    0    0    0    0    0    0    0  527    0]
 [   0    0    0    0    0    0    0    0    0    0    0 2174]]
```

Figura #24

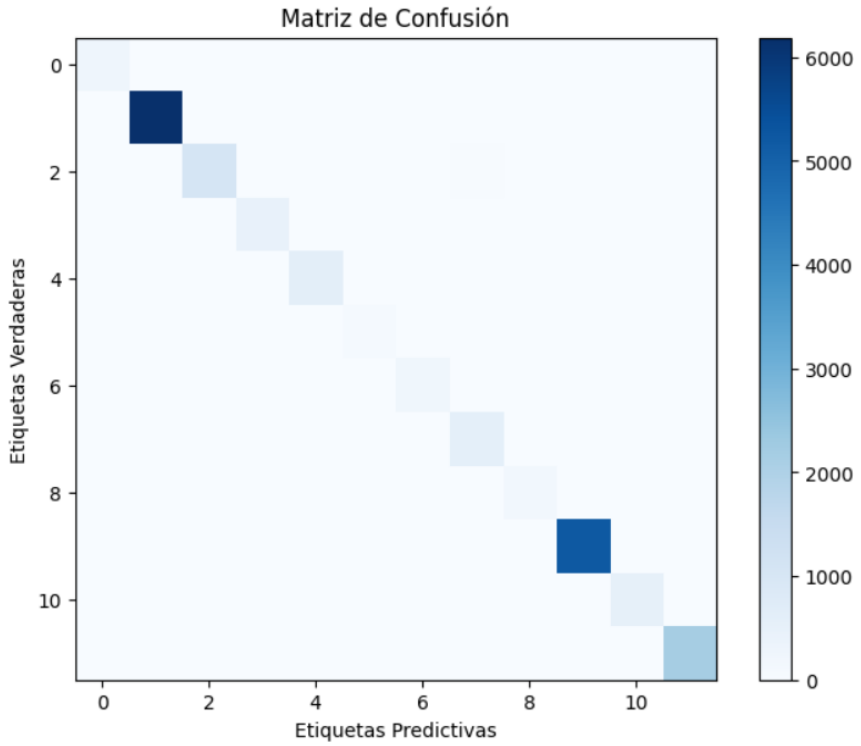

Figura #25

Regresión Lineal.

Variable objetivo: Sector

Variables independientes: Cantidad, Precio

Algoritmo:

Disponible en

https://colab.research.google.com/drive/1zOfxVBF6jFluic5f4oghZbFO4YESq33X#scrollTo=nB6gbnpeVzuF

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, confusion_matrix
```

Figura #26

A continuación, se utiliza la biblioteca pandas (importada como pd) para cargar datos desde un archivo CSV llamado "transacciones.csv" con codificación 'latin1'. Los datos se almacenan en un DataFrame llamado 'data'

```
# Cargar los datos desde un archivo CSV
data = pd.read_csv("transacciones.csv", encoding='latin1')
```

Figura #27

Ahora, vamos a seleccionar características y etiquetas

```
# Supongamos que tienes características en columnas 'Feature1', 'Feature2',
# y una columna 'Target' que deseas predecir
X = data[['Cantidad', 'Precio']]
y = data['Sector']
```

Figura #28

Vamos a dividir los datos en conjuntos de entrenamiento y pruebas:
Se utiliza la función train_test_split de scikit-learn para dividir los datos en conjuntos de entrenamiento (X_train, y_train) y prueba (X_test, y_test). El 30% de los datos se reserva para pruebas, y se establece una semilla aleatoria (random_state=42) para que la división sea reproducible

```
# Dividir los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)
```

Figura #29

Pasemos a la parte de escalar:

```
# Escalar las características (es importante para la Regresión Logística)
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

Figura #30

Es momento de crear y ajustar el modelo de regresión lineal:

```
# Crear y ajustar el modelo de Regresión Logística
logistic_regression = LogisticRegression()
logistic_regression.fit(X_train, y_train)
```

Figura #31

Se utilizan las características escaladas del conjunto de prueba (X_test) para hacer predicciones utilizando el modelo de Regresión Logística. Las predicciones se almacenan en y_pred.

```
# Realizar predicciones en el conjunto de prueba
y_pred = logistic_regression.predict(X_test)
```

Figura #32

Vamos a mostrar el informe de clasificación y la matriz de confusión:

```
# Mostrar el informe de clasificación y la matriz de confusión
print('Informe de Clasificación:')
print(classification_report(y_test, y_pred))

print('Matriz de Confusión:')
```

```
confusion = confusion_matrix(y_test, y_pred)
print(confusion)
```

Figura #33

Se utiliza la biblioteca matplotlib para crear una representación gráfica de la matriz de confusión como una imagen. Esto ayuda a visualizar cómo se distribuyen las predicciones en comparación con las etiquetas verdaderas

```
# Visualizar los resultados (por ejemplo, una matriz de confusión)
plt.figure(figsize=(8, 6))
plt.imshow(confusion, cmap='Blues', interpolation='nearest')
plt.colorbar()
plt.xlabel('Etiquetas Predictivas')
plt.ylabel('Etiquetas Verdaderas')
plt.title('Matriz de Confusión')
plt.show()
```

Figura #34

Conclusiones:

Con el modelo obtuvimos una tabla de resultados clasificada por diversas columnas donde cada una muestra el comportamiento de los objetivos elegidos.

- precision (precisión): La precisión se refiere a la proporción de instancias clasificadas como positivas que son verdaderamente positivas. En este informe, se observa que la precisión varía para diferentes clases. Por ejemplo, para la clase "CARNES FRIAS", la precisión es del 44%, lo que significa que el 44% de las instancias clasificadas como "CARNES FRIAS" por el modelo fueron verdaderamente "CARNES FRIAS".
- recall (sensibilidad o exhaustividad): El recall se refiere a la proporción de instancias verdaderamente positivas que fueron correctamente identificadas por el modelo. Nuevamente, esto varía para diferentes clases. Por ejemplo, para la clase "CARNES FRIAS", el recall es del 90%, lo que significa que el modelo identificó correctamente el 90% de las instancias de "CARNES FRIAS".
- f1-score: El puntaje F1 es una medida que combina precisión y recall en un solo número. Es útil cuando deseas evaluar un equilibrio entre precisión y recall. Valores más altos de F1 indican un mejor rendimiento del modelo.
- support: Esto indica la cantidad de instancias en cada clase en tu conjunto de datos.
- accuracy (exactitud): La exactitud se refiere a la proporción total de instancias clasificadas correctamente por el modelo en todas las clases. En este caso, la exactitud global del modelo es del 44%, lo que significa que el 44% de todas las instancias se clasificaron correctamente.
- macro avg (promedio macro): Estos valores son promedios de las métricas (precisión, recall y f1-score) calculados para cada clase y luego promediados sin tener en cuenta el desequilibrio de clases. Es útil cuando tienes clases con tamaños diferentes.
- weighted avg (promedio ponderado): Estos valores son promedios ponderados de las métricas (precisión, recall y f1-score) calculados para cada clase, ponderados por el soporte de cada clase. Esto da más peso a las clases con más instancias.

	precision	recall	f1-score	support
BEBIDAS	0.00	0.00	0.00	270
CARNES FRIAS	0.44	0.90	0.59	6189
COMIDAS PREPARADAS	0.00	0.00	0.00	1099
COMIDAS REFRIGERADAS	0.00	0.00	0.00	482
CREMAS	0.00	0.00	0.00	609
GENERAL	0.00	0.00	0.00	84
MNTQUILLAS Y MARGARI	0.00	0.00	0.00	231
OTROS	0.00	0.00	0.00	596
POSTRES	0.00	0.00	0.00	191
QUESOS	0.41	0.10	0.16	5219
RES	0.00	0.00	0.00	527
YOGHURT	0.45	0.75	0.56	2174
accuracy			0.44	17671
macro avg	0.11	0.15	0.11	17671
weighted avg	0.33	0.44	0.32	17671

Figura #35

En cuanto a la matriz de confusión, esta no muestra una diagonal como el ejemplo anterior, esto indica que el modelo tiene dificultades para distinguir entre algunas clases, y podría ser útil realizar un análisis adicional para comprender por qué se están produciendo estas confusiones y mejorar el rendimiento del modelo en esas áreas.

```
Matriz de Confusión:
[[   0   66    0    0    0    0    0    0    0  117    0   87]
 [   0 5583    0    0    0    0    0    1    0  375    0  230]
 [   0  725    0    0    0    0    0    1    0   46    0  327]
 [   0  371    0    0    0    0    0    0    0  111    0    0]
 [   0  153    0    0    0    0    0    0    0   14    0  442]
 [   0   83    0    0    0    0    0    0    0    1    0    0]
 [   0   19    0    0    0    0    0    0    0    1    0  211]
 [   0  301    0    0    0    0    0    0    0   40    0  255]
 [   0    4    0    0    0    0    0    1    0    5    0  181]
 [   0 4467    0    0    0    0    0    4    0  516    0  232]
 [   0  518    0    0    0    0    0    0    0    9    0    0]
 [   0  509    0    0    0    0    0    2    0   37    0 1626]]
```

Figura #36

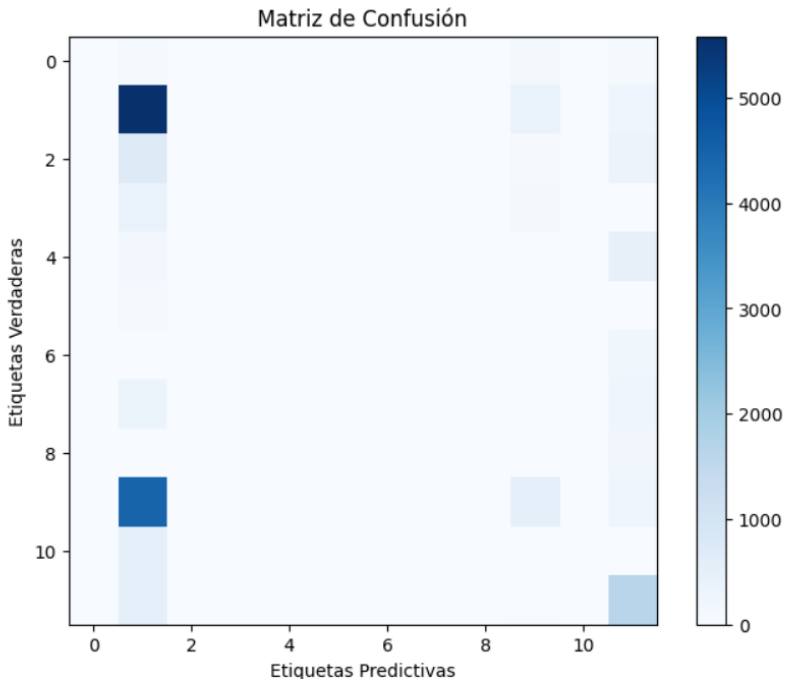

Figura #37

Conclusiones generales.

En esta práctica logramos implementar los algoritmos de clasificación "Árbol de Decisión", "Regresión Lineal" y "Naive Bayes", en donde observamos distintas fortalezas y limitaciones de cada una. Por ejemplom, el Árbol de Decisión demostró ser efectivo para la clasificación, pero su rendimiento dependió de la configuración de hiperparámetros y la prevención del sobreajuste. La Regresión Lineal destacó en problemas de regresión con relaciones lineales, donde se logra desprender información detallada de los objetivos elegidos. Por otro lado, Naive Bayes se destacó en la clasificación de datos categóricos o de texto, ofreciendo un excelente resultado por la elección de objetivos únicos e irrepetibles, aunque esto depende de las columnas elegidas. Por último, podemos decir que la elección del algoritmo más adecuado depende del tipo de datos y del problema específico, resaltando la importancia de la preparación de datos y la optimización de hiperparámetros para obtener un rendimiento óptimo en la minería de datos.

Referencias.

- Su, X., Yan, X., & Tsai, C. L. (2012). Linear regression. *Wiley Interdisciplinary Reviews: Computational Statistics*, *4*(3), 275-294.
- Webb, G. I., Keogh, E., & Miikkulainen, R. (2010). Naïve Bayes. *Encyclopedia of machine learning*, *15*(1), 713-714.
- Cintra, M. E., Monard, M. C., & Camargo, H. A. (2013). A fuzzy decision tree algorithm based on C4. 5. *Mathware & Soft Computing*, *20*(1), 56-62.