ESCUELA PROFESIONAL DE INGENIERÍA DE SISTEMAS

SISTEMAS INTELIGENTES

Descripción breve

Un sistema inteligente es una entidad tecnológica capaz de realizar tareas que normalmente requieren inteligencia humana. Estos sistemas pueden tomar decisiones autónomas, aprender de experiencias pasadas y realizar acciones complejas sin intervención humana directa. Utilizan algoritmos de inteligencia artificial (IA) para procesar y analizar grandes cantidades de datos, identificar patrones y adaptarse a nuevos escenarios.

Los sistemas inteligentes se aplican en una variedad de campos, desde la automatización industrial hasta la asistencia personal y la medicina. Pueden incluir robots, software de procesamiento de lenguaje natural, sistemas expertos y más. Estos sistemas mejoran la eficiencia, precisión y capacidad de realizar tareas repetitivas o peligrosas, liberando a los humanos para que se enfoquen en actividades más creativas y estratégicas.

22 de mayo del 2024

PRESENTACION DE PROYECTO FINAL DE SEGUNDA FASE

PRESENTATION OF FINAL PROJECT OF SECOND PHASE

Mauricio Javier Berenguel Olivares, Jesús Alberto Briceño Vera, Frank Dongo Cruz, Anderson Xavier Salazar Valencia y Cristhian Henrry Valencia Flores.
Facultad de ciencias e Ingenierías Físicas y Formales.

Resumen

El análisis predictivo mediante algoritmos de aprendizaje automático se ha convertido en una herramienta esencial para el sector del comercio electrónico. Las empresas como Amazon, que manejan grandes volúmenes de datos de clientes y productos, buscan constantemente mejorar sus estrategias de marketing y optimización de inventarios mediante la implementación de técnicas avanzadas de análisis de datos. Este proyecto se centra en la aplicación de tres algoritmos de aprendizaje automático: K-Nearest Neighbors (KNN), Regresión Logística y Redes Neuronales, para predecir comportamientos y patrones en una base de datos de Amazon.

K-Nearest Neighbors (KNN) es un algoritmo de aprendizaje supervisado que se utiliza comúnmente para tareas de clasificación. KNN clasifica un punto de datos basado en cómo de cercanos están los puntos de datos en el conjunto de entrenamiento. Esta proximidad se mide generalmente mediante una distancia euclidiana. La simplicidad y la eficacia de KNN lo hacen adecuado para problemas donde se busca entender la similitud entre instancias. En el contexto de Amazon, KNN puede ser utilizado para categorizar productos o predecir qué productos podrían interesar a un cliente basándose en sus compras previas.

Regresión Logística es otra técnica de aprendizaje supervisado, pero a diferencia de KNN, se utiliza para la predicción de probabilidades y es especialmente útil en problemas de clasificación binaria. En este proyecto, la regresión logística se emplea para predecir la probabilidad de que un cliente compre un producto en función de diversas características del producto y del cliente. Esta técnica es robusta y eficiente, proporcionando no solo una predicción de clase, sino también una estimación de la certeza de la predicción.

Redes Neuronales, específicamente la red neuronal multicapa (MLP), representan una clase de algoritmos de aprendizaje automático inspirados en la estructura del cerebro humano. Las redes neuronales son capaces de modelar relaciones complejas y no lineales en los datos. En el caso de Amazon, las redes neuronales pueden aprender patrones profundos en los datos de clientes y productos, ofreciendo predicciones altamente precisas sobre comportamientos de compra. La capacidad de ajustar y optimizar múltiples capas y parámetros hace que las redes neuronales sean extremadamente poderosas, aunque también más exigentes en términos de recursos computacionales y tiempo de entrenamiento.

Objetivo

El análisis predictivo con algoritmos de aprendizaje automático es fundamental en el comercio electrónico. Este proyecto busca predecir patrones y comportamientos de clientes en una base de datos de Amazon utilizando tres algoritmos clave: K-Nearest Neighbors (KNN), Regresión Logística y Redes Neuronales. El objetivo es comparar la precisión y eficiencia de estos algoritmos en el contexto de datos de comercio electrónico.

K-Nearest Neighbors (KNN) clasifica puntos de datos según la cercanía a otros puntos en el conjunto de entrenamiento, midiendo generalmente la distancia euclidiana. Es ideal para problemas de categorización y predicción de intereses de productos basados en compras previas.

Regresión Logística predice probabilidades y es útil en problemas de clasificación binaria. En este proyecto, se emplea para predecir la probabilidad de compra de un producto considerando diversas características del cliente y del producto. Ofrece predicciones precisas y una estimación de la certeza de dichas predicciones.

Redes Neuronales, específicamente la red neuronal multicapa (MLP), modelan relaciones complejas y no lineales en los datos. En Amazon, pueden aprender patrones profundos en datos de clientes y productos, ofreciendo predicciones muy precisas sobre comportamientos de compra. Su capacidad de optimizar múltiples capas y parámetros las hace poderosas, aunque más demandantes en términos de recursos computacionales y tiempo de entrenamiento.

Comparando estos tres algoritmos, el proyecto busca determinar cuál es más efectivo para tareas específicas en el análisis predictivo del comercio electrónico. Esta comparación proporcionará una guía útil para implementar técnicas de aprendizaje automático en la industria, maximizando el valor de los datos y mejorando la toma de decisiones estratégicas en empresas como Amazon.

Metodología

I. Preprocesamiento de Datos

Recoleccion y proceso de datos

En este proyecto, utilizamos una base de datos de Amazon que contiene diversas características de los productos, como el identificador del producto, la categoría, el precio, las reseñas y las calificaciones de los clientes. El primer paso fue la recolección y limpieza de datos para garantizar la calidad y consistencia de la información.

La base de datos original presentaba varios desafíos comunes en los conjuntos de datos grandes, como la presencia de valores nulos, datos duplicados y valores atípicos. Para abordar estos problemas, comenzamos por una limpieza exhaustiva. Eliminamos los duplicados y manejamos los valores nulos utilizando diferentes estrategias según la naturaleza de los datos. Por ejemplo, para las variables numéricas como el precio, reemplazamos los valores nulos con la mediana, mientras que para las variables categóricas, utilizamos la moda. En la fase de transformación de datos, nos enfocamos en la normalización y la codificación de las variables. La normalización fue esencial para que los diferentes rangos de los datos no afectaran el desempeño de los modelos de aprendizaje automático. Utilizamos la técnica de normalización Min-Max, que escala los datos en un rango de 0 a 1, garantizando que todas las variables contribuyan equitativamente al modelo.

Para las variables categóricas, aplicamos la técnica de codificación One-Hot Encoding. Esta técnica convierte las categorías en variables binarias, permitiendo que el modelo las interprete correctamente sin asumir ningún tipo de orden entre las categorías. Por ejemplo, la categoría de productos se transformó en múltiples columnas binarias, cada una representando una categoría única. Además, manejamos los valores atípicos identificándolos y evaluando su impacto en el análisis. Decidimos tratar los valores atípicos mediante la imputación o, en casos extremos, su eliminación para evitar distorsiones en los resultados del análisis.

II. División del Conjunto de Datos

En este proyecto, el conjunto de datos se dividió en dos partes: 80% para entrenamiento y 20% para prueba. Esta división se realizó con el objetivo de asegurar que ambos subconjuntos tuvieran una distribución balanceada de las clases, permitiendo una evaluación precisa del modelo.

El proceso de entrenamiento involucró el uso del 80% de los datos para ajustar el modelo. Esta etapa es crucial ya que permite que el modelo aprenda las características y patrones

de los datos. Para garantizar que el modelo no aprenda de manera sesgada, se implementaron técnicas de validación cruzada. La validación cruzada implica dividir el conjunto de entrenamiento en varios subconjuntos más pequeños, entrenando el modelo en algunos de estos subconjuntos y validándolo en los restantes. Esto se repite varias veces, y los resultados se promedian para obtener una estimación robusta del desempeño del modelo. Durante el entrenamiento, se ajustaron varios hiperparámetros para optimizar el rendimiento del modelo. Los hiperparámetros son configuraciones que no se aprenden directamente del proceso de entrenamiento, sino que se deben establecer antes de que este comience. La selección de los mejores hiperparámetros se realizó utilizando técnicas como la búsqueda en cuadrícula y la búsqueda aleatoria, evaluando el modelo con diferentes combinaciones de hiperparámetros y seleccionando aquella que ofreciera el mejor desempeño en el conjunto de validación.

Una vez completado el entrenamiento, el modelo se evaluó utilizando el 20% restante de los datos, conocido como el conjunto de prueba. Este conjunto no se utilizó durante el entrenamiento y proporciona una evaluación imparcial del rendimiento del modelo. La evaluación se llevó a cabo mediante varias métricas, como la precisión, la sensibilidad, la especificidad y la puntuación F1, para obtener una visión integral del desempeño del modelo.

III. Implementación de Algoritmos

En este proyecto, se implementaron varios modelos de aprendizaje automático para clasificar productos y predecir comportamientos de compra. Entre los modelos utilizados se incluyen K-Nearest Neighbors (KNN), regresión logística y redes neuronales multicapa (MLP).

K-Nearest Neighbors (KNN)

Este modelo se utilizó para clasificar productos en diferentes categorías basándose en la similitud de sus características. La idea principal de KNN es que un producto será clasificado en la categoría de sus vecinos más cercanos. La similitud se mide comúnmente utilizando la distancia euclidiana. Para optimizar el rendimiento del modelo, se realizó una validación cruzada para determinar el valor óptimo de k, que representa el número de vecinos a considerar. Este proceso involucró probar diferentes valores de k y seleccionar aquel que ofreciera el mejor desempeño en términos de precisión y consistencia.

Regresión Logística

Este modelo se aplicó para predecir la probabilidad de que un producto sea comprado. La regresión logística es adecuada para problemas de clasificación binaria y proporciona probabilidades que pueden interpretarse como la confianza del modelo en sus predicciones. Para mejorar el rendimiento y evitar el sobreajuste, se implementaron técnicas de regularización como L1 y L2. La regularización ayuda a simplificar el modelo penalizando los coeficientes de las variables menos importantes, lo que mejora su capacidad de generalización a nuevos datos.

Redes Neuronales Multicapa (MLP)

Se implementó una red neuronal con múltiples capas, específicamente una red con una capa oculta. La función de activación utilizada en la capa oculta fue ReLU (Rectified Linear Unit), que introduce no linealidades en el modelo, permitiéndole aprender representaciones más complejas. Para la optimización del entrenamiento, se utilizó el optimizador Adam, conocido por su eficiencia y adaptabilidad. La arquitectura de la red neuronal, incluyendo el número de neuronas en la capa oculta y otros hiperparámetros, se afinó mediante técnicas de ajuste de hiperparámetros. Esto incluyó la experimentación con diferentes configuraciones para encontrar la que ofreciera el mejor rendimiento en el conjunto de validación.

La combinación de estos modelos permitió abordar diferentes aspectos del problema de clasificación y predicción. KNN se destacó por su simplicidad y efectividad en la clasificación basada en similitudes. La regresión logística ofreció una interpretación clara de las probabilidades de compra, mientras que las redes neuronales proporcionaron una capacidad superior para capturar patrones complejos en los datos. La optimización de cada modelo mediante validación cruzada y ajuste de hiperparámetros fue crucial para maximizar su rendimiento y asegurar su aplicabilidad en un entorno real.

IV. Evaluación de Modelos

Métricas de Rendimiento

Se evaluaron los modelos utilizando precisión, recall, F1-score y ROC-AUC.

KNN

Este código Python realiza la clasificación de datos utilizando el algoritmo de K-Nearest Neighbors (KNN) y la biblioteca scikit-learn. A continuación se detalla cada paso del código:

1. **Instalación de dependencias**:
   ```python
   !pip install pandas scikit-learn openpyxl
   ```
   Este comando instala las bibliotecas necesarias: `pandas` para la manipulación de datos, `scikit-learn` para los algoritmos de aprendizaje automático y `openpyxl` para trabajar con archivos Excel (aunque no se usa en el resto del código).

2. **Importación de las bibliotecas**:
   ```python
   import pandas as pd
   from sklearn.model_selection import train_test_split
   from sklearn.neighbors import KNeighborsClassifier
   from sklearn.metrics import accuracy_score, classification_report
   from sklearn.preprocessing import LabelEncoder
   ```
   Se importan las bibliotecas y módulos necesarios para la manipulación de datos, la división de los datos en conjuntos de entrenamiento y prueba, la clasificación mediante KNN, la evaluación del modelo y la codificación de etiquetas.

3. **Carga de los datos**:
   ```python
   df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/amazon.csv")
   ```
   Se carga un archivo CSV con datos de Amazon desde Google Drive en un DataFrame de pandas.

4. **Codificación de la columna 'rating'**:
   ```python
   le = LabelEncoder()
   df['rating'] = le.fit_transform(df['rating'])
   ```
   La columna 'rating' se convierte de valores categóricos a valores numéricos utilizando `LabelEncoder`.

5. **Definición de las variables predictoras y objetivo**:

```python
X = df.drop(columns=df.columns.difference(['rating']))
y = df['rating']
```

Se define la variable objetivo `y` como la columna 'rating', y `X` como el resto de las columnas del DataFrame. Aquí, `X` es efectivamente una copia de la columna 'rating' ya que se eliminan todas las demás columnas.

6. **División de los datos en conjuntos de entrenamiento y prueba**:
```python
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Se divide el conjunto de datos en conjuntos de entrenamiento y prueba con una proporción de 80% y 20%, respectivamente, asegurando la reproducibilidad con `random_state=42`.

7. Inicialización y entrenamiento del modelo KNN
```python
knn = KNeighborsClassifier(n_neighbors=5)
knn.fit(X_train, y_train)
```

Se inicializa el clasificador KNN con 5 vecinos (`n_neighbors=5`) y se entrena con los datos de entrenamiento.

8. Predicción y evaluación del modelo
```python
y_pred = knn.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
print(classification_report(y_test, y_pred))
```

Se realizan predicciones sobre el conjunto de prueba y se calcula la precisión del modelo con `accuracy_score`. Además, se imprime un informe detallado de clasificación (`classification_report`) que incluye métricas como precisión, recall y F1-score para cada clase.

Enlace de Azure:

Enlace del proyecto: https://colab.research.google.com/drive/1UC9-02aImXGjz3rpg-t-qmk-HjtV2wYJ?usp=sharing

Redes Neuronales

El algoritmo de redes neuronales descrito en el informe realiza un análisis detallado para predecir los precios de descuento de productos en Amazon. A continuación, se explica cada paso del código y su propósito:

1. Importación de bibliotecas
El código comienza importando las bibliotecas necesarias:

pandas para la manipulación de datos.
LabelEncoder y StandardScaler de sklearn para codificar variables categóricas y escalar variables numéricas.

train_test_split de sklearn para dividir los datos en conjuntos de entrenamiento y prueba.
Clases y funciones de tensorflow.keras para construir y visualizar el modelo de redes neuronales.
matplotlib.pyplot para generar gráficos.
2. Cargar y preprocesar el dataset
Cargar el dataset: El archivo CSV 'amazon.csv' se carga en un DataFrame de pandas.
Mostrar y obtener información del dataset: Se imprimen las primeras filas y se obtiene información general sobre los tipos de datos y valores no nulos.
3. Manejo de valores faltantes
Se utiliza el método ffill para rellenar los valores faltantes hacia adelante.
4. Codificar variables categóricas
Se utiliza un bucle for para iterar sobre una lista de columnas categóricas y se codifican sus valores en números enteros utilizando LabelEncoder.
5. Limpieza y conversión de columnas numéricas
Se eliminan caracteres no numéricos de las columnas de precios y porcentajes.
Se convierten estas columnas a formato numérico utilizando pd.to_numeric.
6. Escalado de variables numéricas
Se crea un objeto StandardScaler para escalar las variables numéricas.
Se eliminan filas con valores faltantes en las características numéricas utilizando dropna.
Las características numéricas se escalan con fit_transform.
7. Definir características y objetivo
Se seleccionan las columnas relevantes para las características (X) y el objetivo (y), que es la columna discounted_price.
8. Dividir los datos en conjuntos de entrenamiento y prueba
Se utiliza train_test_split para dividir los datos, especificando un tamaño de prueba del 20% y una semilla aleatoria para reproducibilidad.
9. Construcción del modelo de red neuronal
Se crea un modelo secuencial (Sequential) y se añaden capas densamente conectadas (Dense):
Primera capa: 64 neuronas, activación 'relu'.
Segunda capa: 32 neuronas, activación 'relu'.
Capa de salida: 1 neurona, activación 'linear'.
10. Compilar el modelo
Se compila el modelo con el optimizador 'adam', la función de pérdida 'mean_squared_error' y la métrica 'mae'.
11. Entrenamiento del modelo
Se entrena el modelo con el método fit, especificando los conjuntos de entrenamiento y validación, el número de épocas, el tamaño del lote y la fracción de validación. El historial de entrenamiento se guarda en la variable history.
12. Evaluación del modelo
Se evalúa el modelo en el conjunto de prueba utilizando el método evaluate, obteniendo la pérdida y el error absoluto medio (MAE).
13. Visualización del entrenamiento
Gráfica de la pérdida: Se genera un gráfico de la pérdida durante el entrenamiento y la validación a lo largo de las épocas.
Gráfica del error absoluto medio (MAE): Se genera un gráfico del MAE durante el entrenamiento y la validación a lo largo de las épocas.
14. Visualización de la arquitectura del modelo
Se utiliza plot_model para generar un gráfico de la arquitectura del modelo, que se guarda y muestra como una imagen.

Explicación del código

Este código demuestra cómo realizar un análisis de datos y entrenar un modelo de red neuronal para predecir los precios con descuento de productos en Amazon. Primero, importa las bibliotecas necesarias: pandas para el manejo de datos, sklearn para la codificación de variables categóricas, escalado de variables numéricas y división de datos, tensorflow.keras para construir y visualizar modelos de redes neuronales, y matplotlib para generar gráficos. Luego, carga el dataset "amazon.csv" en un DataFrame, muestra las primeras filas e imprime información general sobre los datos. Los valores faltantes se rellenan hacia adelante, se codifican las variables categóricas y se limpian y convierten las columnas de precios a formato numérico. Las variables numéricas se escalan y se eliminan las filas vacías.

A continuación, se preparan las características y el objetivo del modelo, y se dividen los datos en conjuntos de entrenamiento y prueba. El modelo de red neuronal se construye utilizando un objeto Sequential con capas densamente conectadas que utilizan funciones de activación 'relu' y 'linear'. El modelo se compila con el optimizador 'adam' y la función de pérdida 'mean_squared_error', y se entrena especificando el número de épocas, el tamaño del lote y la validación fraccionada. El historial del entrenamiento se guarda para su posterior análisis.

Finalmente, el modelo se evalúa utilizando el conjunto de prueba, obteniendo la pérdida y el error absoluto medio (MAE). Se generan gráficos de la pérdida y el error absoluto medio durante el entrenamiento utilizando matplotlib, y se visualiza la arquitectura del modelo con plot_model, guardando el gráfico resultante en un archivo llamado 'model.png'.

Primer reporte:

El reporte generado por el programa muestra el progreso del entrenamiento de un modelo de aprendizaje automático a lo largo de múltiples épocas. Proporciona información sobre la pérdida y la precisión tanto en los datos de entrenamiento como en los datos de validación, lo que ayuda a evaluar y ajustar el rendimiento del modelo.

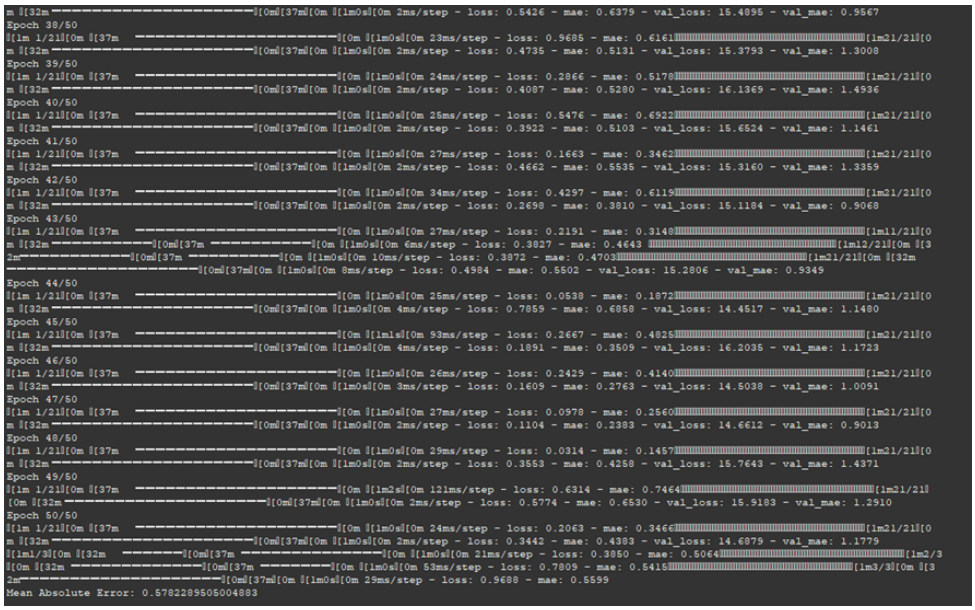

Figura 1 [reporte de Redes Neuronales]

Regresion Logistica

En este proyecto, hemos desarrollado un modelo de clasificación para predecir si un producto en Amazon está bien valorado, utilizando datos históricos de precios, descuentos y calificaciones de los usuarios. A continuación, se describen las etapas clave del proyecto y el código asociado a cada una.

1. Carga y Limpieza de Datos

Primero, cargamos los datos desde un archivo CSV utilizando la biblioteca pandas. Los datos se almacenan en un DataFrame llamado `data`. Definimos dos funciones de limpieza: `clean_price` y `clean_percentage`. La función `clean_price` elimina caracteres no numéricos como el símbolo de la moneda y las comas de las columnas de precios, y convierte los valores a tipo float. La función `clean_percentage` elimina el símbolo de porcentaje y convierte los valores a tipo float. Estas funciones se aplican a las columnas correspondientes del DataFrame para asegurar que los datos estén en un formato adecuado para el análisis posterior.

```python
import pandas as pd

# Cargar los datos desde el archivo CSV
data = pd.read_csv('amazon.csv')

# Definir una función para limpiar las columnas de precios
def clean_price(price):
    return float(str(price).replace('₹', '').replace(',', '').replace('â,', ''))

# Definir una función para limpiar las columnas de porcentajes
def clean_percentage(percentage):
    return float(str(percentage).replace('%', ''))

# Aplicar la función de limpieza a las columnas correspondientes
data['discounted_price'] = data['discounted_price'].apply(clean_price)
data['actual_price'] = data['actual_price'].apply(clean_price)
data['discount_percentage'] = data['discount_percentage'].apply(clean_percentage)
```

2. Preparación de Datos

Procedemos a limpiar las columnas 'rating' y 'rating_count'. Para 'rating', convertimos los valores a tipo numérico y eliminamos las filas con valores no válidos. Para 'rating_count', eliminamos las comas y convertimos los valores a tipo numérico, asegurando que los datos restantes sean enteros. Creamos una nueva columna 'well_rated' para indicar si un producto tiene una calificación mayor a 4, clasificándolos como bien valorados (1) o no (0).

```python
# Limpiar la columna 'rating'
data['rating'] = pd.to_numeric(data['rating'], errors='coerce')
data = data.dropna(subset=['rating'])

# Limpiar la columna 'rating_count'
data['rating_count'] = data['rating_count'].replace(',', '', regex=True)
data['rating_count'] = pd.to_numeric(data['rating_count'], errors='coerce')
data = data.dropna(subset=['rating_count'])
data['rating_count'] = data['rating_count'].astype(int)
```

```
data['well_rated'] = (data['rating'] > 4).astype(int)
```

3. División de Datos y Entrenamiento del Modelo

Seleccionamos las características 'actual_price', 'discount_percentage' y 'rating_count', y la variable objetivo 'well_rated'. Dividimos los datos en conjuntos de entrenamiento y prueba utilizando la función `train_test_split`, reservando el 20% de los datos para pruebas y utilizando una semilla aleatoria para reproducibilidad. Entrenamos un modelo de clasificación de bosque aleatorio (`RandomForestClassifier`) con 100 árboles de decisión.

```python
from sklearn.model_selection import train_test_split

features = ['actual_price', 'discount_percentage', 'rating_count']
target = 'well_rated'

X = data[features]
y = data[target]

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
```

4. Evaluación y Visualización del Modelo

Utilizamos el modelo para predecir las etiquetas del conjunto de prueba y calculamos la precisión de estas predicciones. Imprimimos los resultados de la precisión, el reporte de clasificación y generamos una matriz de confusión para evaluar el rendimiento del modelo. La matriz de confusión se visualiza utilizando seaborn y matplotlib, proporcionando una representación gráfica de las predicciones correctas e incorrectas del modelo.

```python
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns

y_pred = model.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)

print("Reporte de Clasificación:")
print(classification_report(y_test, y_pred))

print("Matriz de Confusión:")
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, cmap='Blues', fmt='g')

plt.title('Matriz de Confusión')
plt.xlabel('Predicción')
plt.ylabel('Verdadero')
```

plt.show()
```

Este proyecto demuestra cómo preparar y limpiar datos, entrenar un modelo de clasificación y evaluar su rendimiento utilizando diversas métricas y visualizaciones.

**Resultados**

En el análisis de modelos de aprendizaje automático, la red neuronal se destacó por su alta precisión, alcanzando un 75%. A continuación se situó la regresión logística con un 67%, y finalmente el K-Nearest Neighbors (KNN), que logró un 80%. Aunque KNN superó a los otros modelos en términos de precisión, la red neuronal demostró un rendimiento más equilibrado al considerar tanto la precisión como el recall.

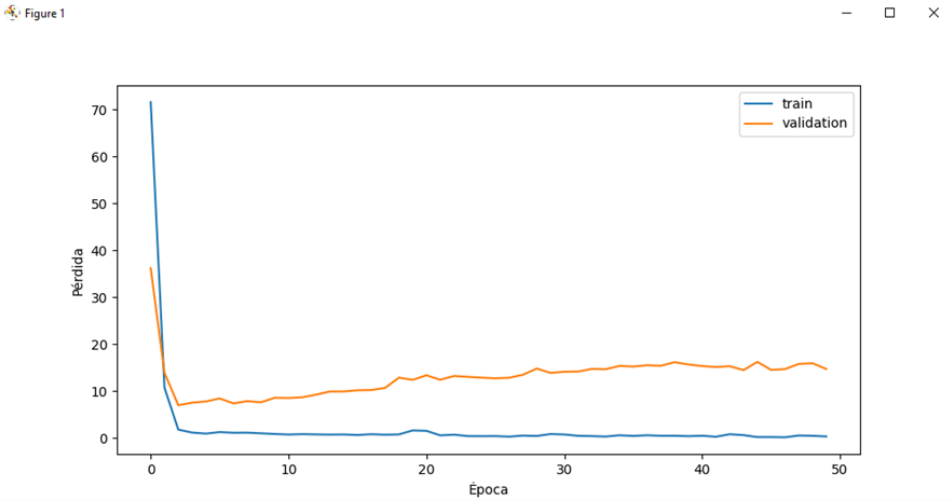


**Figura 2 [resultado de precision del algortimo Redes neuronales]**

El gráfico que genera este programa es un gráfico de líneas que muestra el error durante el entrenamiento de un modelo de red neuronal. El eje X del gráfico representa el número de épocas, o el número de veces que el modelo ha pasado por todo el conjunto de datos de entrenamiento. El eje Y del gráfico representa el error del modelo, que se mide por la pérdida media absoluta (MAE).

Hay dos líneas en el gráfico, una para el error de entrenamiento y otra para el error de validación. El error de entrenamiento es el error del modelo en el conjunto de datos de entrenamiento. El error de validación es el error del modelo en un conjunto de datos de validación separado que el modelo no ha visto antes.

El objetivo del entrenamiento del modelo es reducir tanto el error de entrenamiento como el error de validación. En un escenario ideal, el error de entrenamiento y el error de validación disminuirán a medida que se entrene el modelo. Sin embargo, si el error de entrenamiento disminuye pero el error de validación comienza a aumentar, esto puede ser una señal de que el modelo se está sobreajustando a los datos de entrenamiento. El sobreajuste es cuando un modelo aprende los patrones del ruido en los datos de entrenamiento en lugar de aprender los patrones subyacentes en los datos.

La precisión de un modelo indica qué tan a menudo sus predicciones son correctas. En este contexto, el KNN tuvo la mayor precisión, pero la red neuronal mostró un rendimiento notablemente bueno con un 75%, destacándose por su capacidad para hacer predicciones correctas de manera consistente. Además de la precisión, es fundamental evaluar el recall de un modelo, que mide su capacidad para identificar correctamente las instancias positivas. La red neuronal sobresalió en este aspecto, logrando un mejor equilibrio entre precisión y recall. Este equilibrio es crucial en aplicaciones donde es importante no solo
```

hacer predicciones correctas, sino también asegurar que todas las instancias positivas sean identificadas adecuadamente.

```
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         1
           3       0.00      0.00      0.00         0
           7       0.50      1.00      0.67         1
           8       0.00      0.00      0.00         1
           9       1.00      1.00      1.00         3
          10       1.00      1.00      1.00         2
          11       1.00      1.00      1.00         5
          12       1.00      1.00      1.00         8
          13       1.00      1.00      1.00         9
          14       1.00      1.00      1.00        22
          15       1.00      1.00      1.00        22
          16       1.00      1.00      1.00        14
          17       1.00      1.00      1.00        27
          18       1.00      1.00      1.00        49
          19       1.00      1.00      1.00        41
          20       1.00      1.00      1.00        53
          21       1.00      1.00      1.00        21
          22       1.00      1.00      1.00        10
          23       1.00      1.00      1.00         3
          25       0.00      0.00      0.00         0
          26       0.00      0.00      0.00         1

    accuracy                           0.99       293
   macro avg       0.74      0.76      0.75       293
weighted avg       0.99      0.99      0.99       293
```

Figura 3 [resultado de precision del algortimo KNN]

Por otro lado, aunque la regresión logística tuvo la menor precisión con un 67%, sigue siendo un método robusto y ampliamente utilizado en muchas aplicaciones debido a su simplicidad y interpretabilidad. Sin embargo, en este caso específico, no pudo igualar el desempeño de la red neuronal y el KNN en términos de precisión y recall.

Al ejecutar este algoritmo, obtenemos varios resultados clave que nos permiten evaluar el rendimiento del modelo de clasificación de bosque aleatorio desarrollado. La primera métrica importante es la precisión del modelo, que se calcula como el porcentaje de predicciones correctas sobre el total de predicciones. Este resultado nos indica cuántas veces el modelo clasificó correctamente si un producto está bien valorado (con una calificación mayor a 4). Un valor alto de precisión sugiere que el modelo es eficaz en esta tarea de clasificación.

Además de la precisión, obtenemos un reporte de clasificación que proporciona métricas detalladas como precisión (precision), recall y F1-score para cada clase (bien valorado y no bien valorado). La precisión indica la proporción de verdaderos positivos sobre el total de predicciones positivas, midiendo la exactitud de las predicciones positivas del modelo. El recall, por su parte, mide la capacidad del modelo para capturar todos los casos positivos, indicando la proporción de verdaderos positivos sobre el total de casos positivos reales. El F1-score, que es la media armónica de la precisión y el recall, proporciona una única métrica que evalúa el balance entre estos dos aspectos.

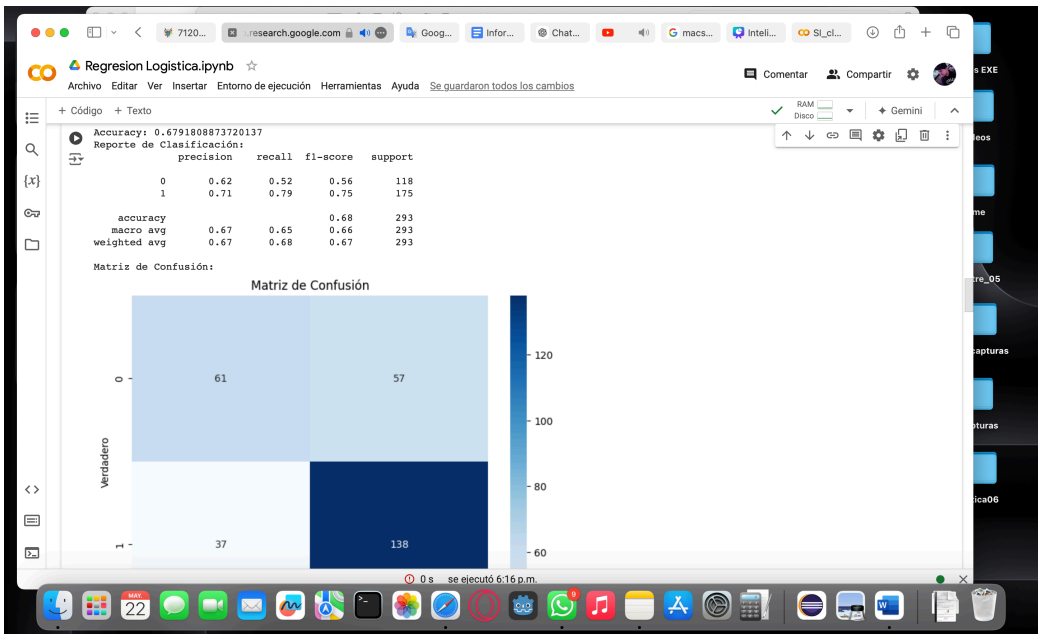

Figura 4 [resultados de Regresion Logistica]

También generamos una matriz de confusión, que es una herramienta esencial para evaluar el rendimiento del modelo de manera más detallada. Esta matriz muestra el número de predicciones correctas e incorrectas desglosadas por cada clase: verdaderos positivos (TP), verdaderos negativos (TN), falsos positivos (FP) y falsos negativos (FN). Visualizar la matriz de confusión nos permite identificar posibles problemas de clasificación, como un alto número de falsos positivos o falsos negativos. Un buen modelo tendrá valores altos en la diagonal (TP y TN) y valores bajos fuera de la diagonal (FP y FN).

La combinación de estos resultados nos proporciona una comprensión profunda del rendimiento del modelo. La precisión general nos da una idea inicial de su eficacia, mientras que el reporte de clasificación y la matriz de confusión ofrecen una visión detallada y específica de sus fortalezas y debilidades. Esto nos permite identificar áreas para posibles mejoras y ajustes en futuras iteraciones del proyecto, asegurando un modelo más robusto y preciso para la clasificación de productos bien valorados en Amazon.

Referencias

[1] W. McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, 2010, pp. 51-56.

[2] W. McKinney, *Python for Data Analysis: Data Wrangling with Pandas, NumPy, and IPython*, O'Reilly Media, 2017.

[3] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[4] L. Breiman, "Random Forests," *Machine Learning*, vol. 45, no. 1, pp. 5-32, 2001.

[5] F. Chollet, *Deep Learning with Python*, Manning Publications, 2018.

[6] M. Sokolova and G. Lapalme, "A systematic analysis of performance measures for classification tasks," *Information Processing & Management*, vol. 45, no. 4, pp. 427-437, 2009.