MIDA - 2023

FRAUD DETECTION IN CREDIT CARD TRANSACTIONS

Francisco José Martín Aguilar

INDEX

Introduction to the Project

Project Title: Credit Card Fraud Detection with Artificial Intelligence

Description:

This project aims to develop a credit card fraud detection system using Artificial Intelligence (AI) techniques. You will use a dataset provided by Kaggle, which contains information about credit card transactions labeled as legitimate or fraudulent.

Description of the data source

The data for this project was obtained from Kaggle. It consists of a table that contains a numeric identifier for each row, twenty-eight columns with an indecipherable numeric value, one column with the total amount spent in each transaction, and finally, a column indicating whether the transaction is fraudulent or not.

The link to the data is as follows:

https://www.kaggle.com/datasets/nelgiriyewithana/credit-card-fraud-detection-dataset-2023/

# Time Number of seconds elapsed between this transaction and the first transaction in the dataset	# V1 may be result of a PCA Dimensionality reduction to protect user identities and sensitive features(v1-v28)	# V2	...	# V28 abc	# Amount Transaction amount	# Class 1 for fraudulent transactions, 0 otherwise
0 173k	-56.4 2.45	-72.7 22.1	...	-15.4 33.8	0 25.7k	0 1
0	-1.3598071336738	-0.0727811733098497		-0.0210530534538215	149.62	0
0	1.19185711131486	0.26615071205963		0.0147241691924927	2.69	0
1	-1.35835406159823	-1.34016307473609	...	-0.0597518405929204	378.66	0
1	-0.966271711572087	-0.185226008082898		0.0614576285006353	123.5	0
2	-1.15823309349523	0.877736754848451		0.215153147499206	69.99	0
2	-0.425965884412454	0.960523044882985		0.0810802569229443	3.67	0
4	1.22965763450793	0.141003507049326		0.00516776890624916	4.99	0
7	-0.644269442348146	1.41796354547385		-1.08533918832377	40.8	0
7	-0.89428608220282	0.286157196276544		0.14240432992147	93.2	0

Figura 1: Kaggle Dataset Content

Description of data preprocessing

In the data preprocessing process, it was decided to retain all available columns, as each of the 28 numerical variables, along with the columns representing the transaction amount and the fraud indicator, is considered essential to address the problem at hand. The only column that will be removed is the one representing the time elapsed between the first and the last transaction, called “Time.”

A thorough review has been conducted to ensure that there are no redundant or irrelevant data that could be removed without compromising the quality of the dataset. In this regard, it is confirmed that all variables play a crucial role in the comprehensive representation of the fraud detection problem. No value simplification has been performed, as each one provides valuable information for the analysis. Additionally, it has been verified that there are no missing values in the dataset, thus ensuring the integrity and quality of the information used in the study.

	V1	V2	V3	V4	V27	V28	Amount	Class
0	-0.261	-0.470	2.496	-0.084	-0.081	-0.151	17982.10	0
1	0.985	-0.356	0.558	-0.430	-0.248	-0.065	6531.37	0
2	-0.260	-0.949	1.729	-0.458	-0.300	-0.245	2513.54	0
3	-0.152	-0.509	1.747	-1.090	-0.165	0.048	5384.44	0
4	-0.207	-0.165	1.527	-0.448	0.024	0.419	14278.97	0
...	...	...	...	...	...	...	...	...
568625	-0.833	0.062	-0.900	0.904	3.309	0.082	4394.16	1
568626	-0.670	-0.203	-0.068	-0.267	-1.529	1.704	4653.40	1
568627	-0.312	-0.004	0.138	-0.036	-0.488	-0.269	23572.85	1
568628	0.637	-0.517	-0.301	-0.144	-0.159	-0.076	10160.83	1
568629	-0.795	0.433	-0.649	0.375	-1.575	0.723	21493.92	1

568630 rows × 30 columns

Figure 2: Kaggle Dataset Content

The project content will include 3 datasets:

creditcard_2023.csv: Contains the raw data
procesado_creditcard_2023.csv: Contains the processed data
procesado_creditcard_2023_svm.csv: Contains the processed data with only 50000 rows

Plots

Fraudulent/Legitimate

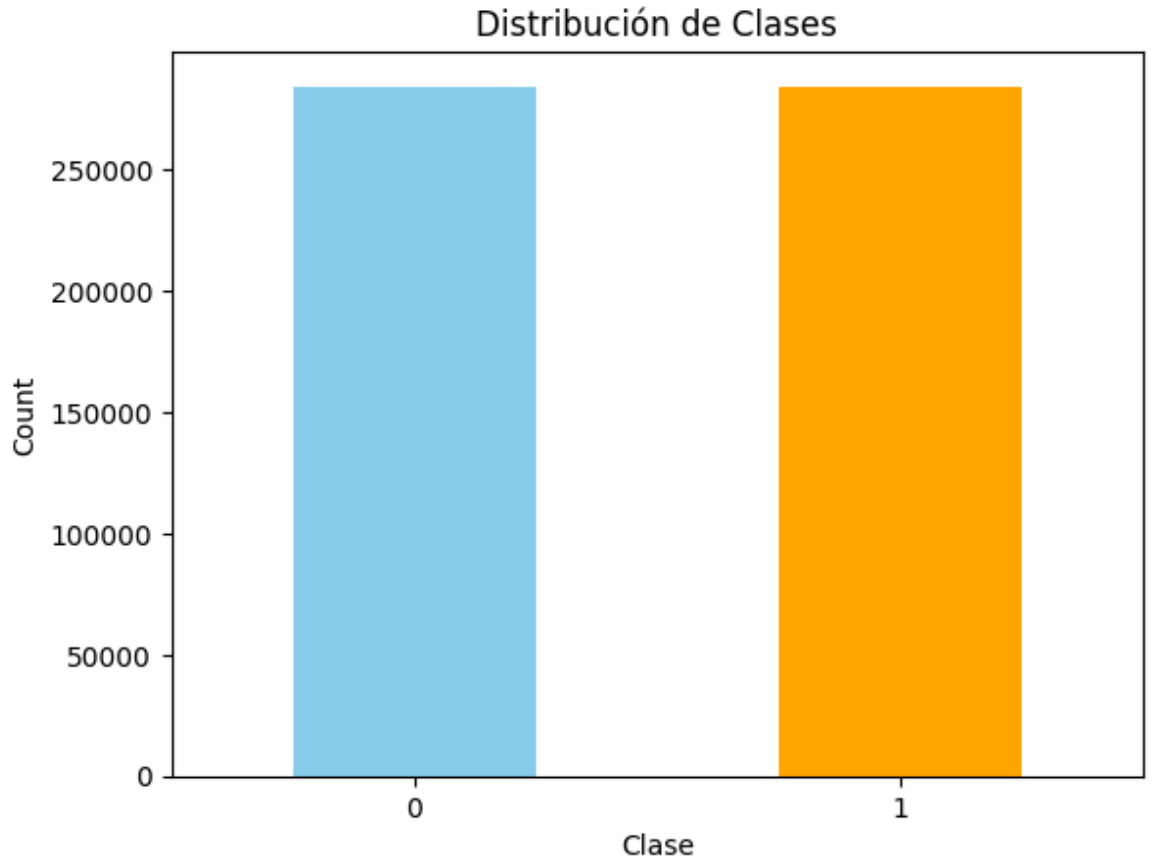


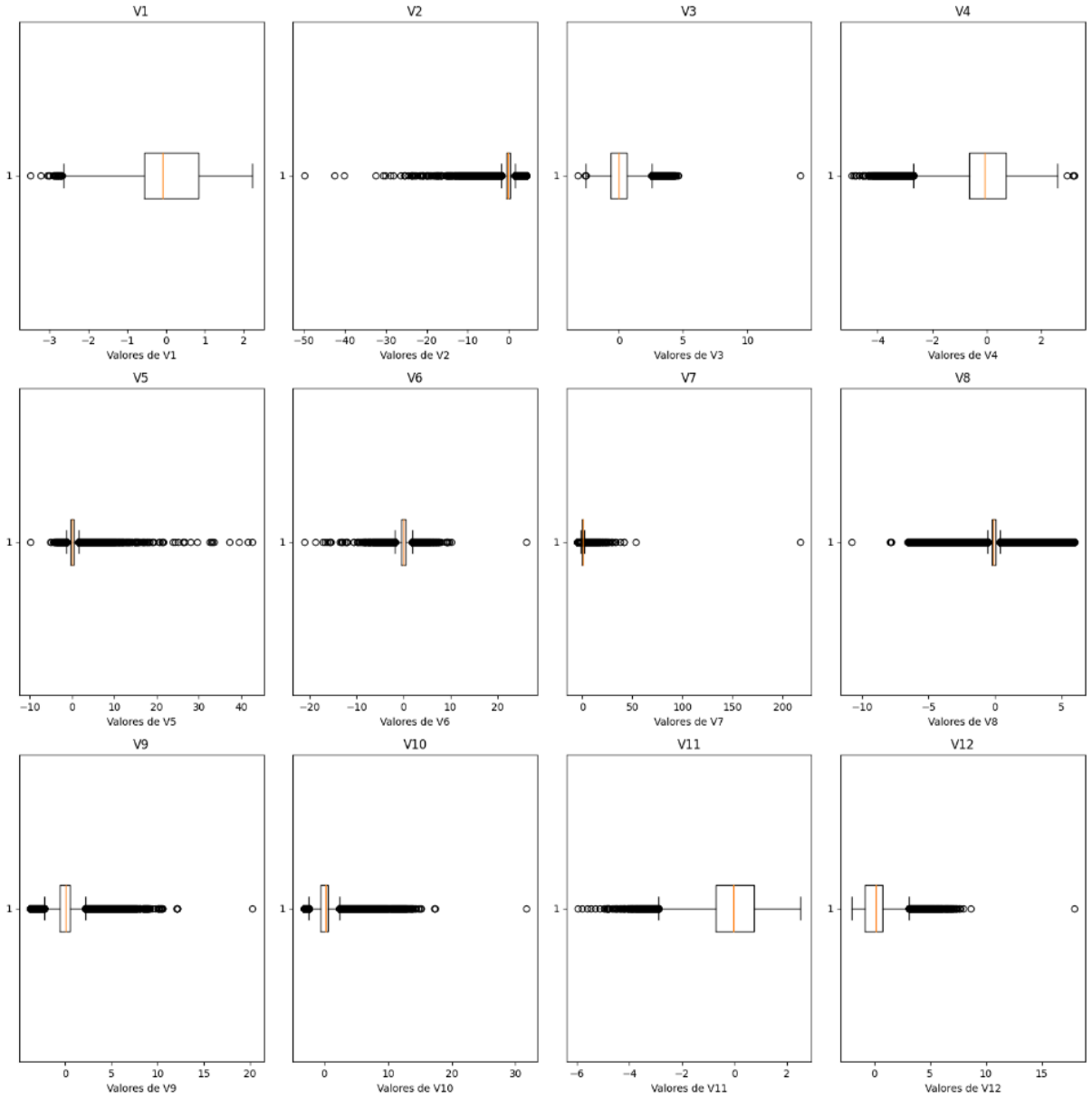
Figure 3: Fraudulent/Legitimate Data Plot

Note:

The dataset in question has a balanced distribution with over 250,000 instances of fraudulent transactions and an equally significant number of over 250,000 legitimate transactions. This balanced ratio provides an ideal scenario for developing a credit card fraud detection model, as it allows training the artificial intelligence with a dataset that accurately reflects reality, considering both fraud cases and legitimate transactions. The breadth of the data, exceeding 500,000 instances, gives the model a substantial foundation to learn complex patterns and subtleties in the data, thus providing the opportunity to develop a robust and accurate fraud detection system that can effectively generalize to real-world situations. This balance and substantial volume of data significantly contribute to the AI's ability to learn in a more comprehensive and representative manner.

A reduction in data volume is considered for training the SVM model, especially considering the nature of SVM and the potential time limitations associated with larger datasets. SVM can be computationally intensive, and reducing the data volume may yield benefits in terms of efficiency and training speed.

Box and Whisker



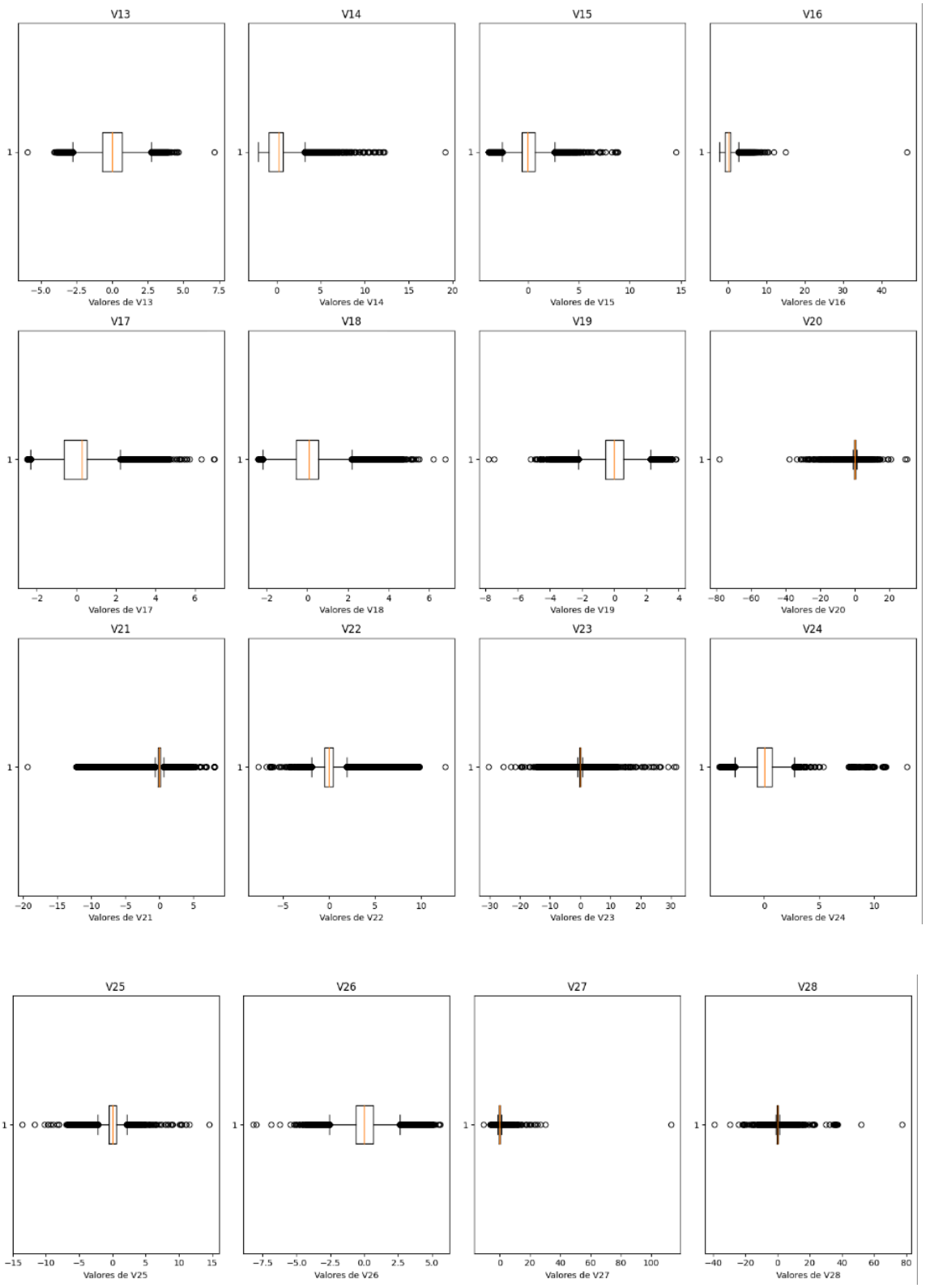


Figure 4: Box and Whisker Data Plots

Note:

The analysis of the distribution of variables V1 through V28 provides a detailed view of the data variability in each of these dimensions. By examining the distribution graphs, we can observe the shape and dispersion of values for each variable. For instance, variables V1, V4, and V11 show an almost symmetric distribution with a pronounced peak around certain values, whereas other variables such as V2, V5, V6, V7, V8, V21, V23, V27, and V28 exhibit more skewed distributions.

Additionally, we note that most variables present the highest concentration of data in the range from -2 to 2, with some pronounced peaks further away. It is noteworthy that the median of all data is at value 0, indicating a centered distribution. Particularly, I highlight the graph of values V1 because it shows fewer values in the quartiles and presents some outliers in the -3 range. This behavior may indicate greater variability or skew in this variable compared to others.

Furthermore, some graphs, such as V5 or V20, show a notable number of outliers, suggesting the presence of unusual data that could be associated with potential fraud. Observing histograms provides additional information by revealing the frequency of values in specific intervals, allowing us to identify possible biases, outliers, and data concentration patterns.

Heatmap

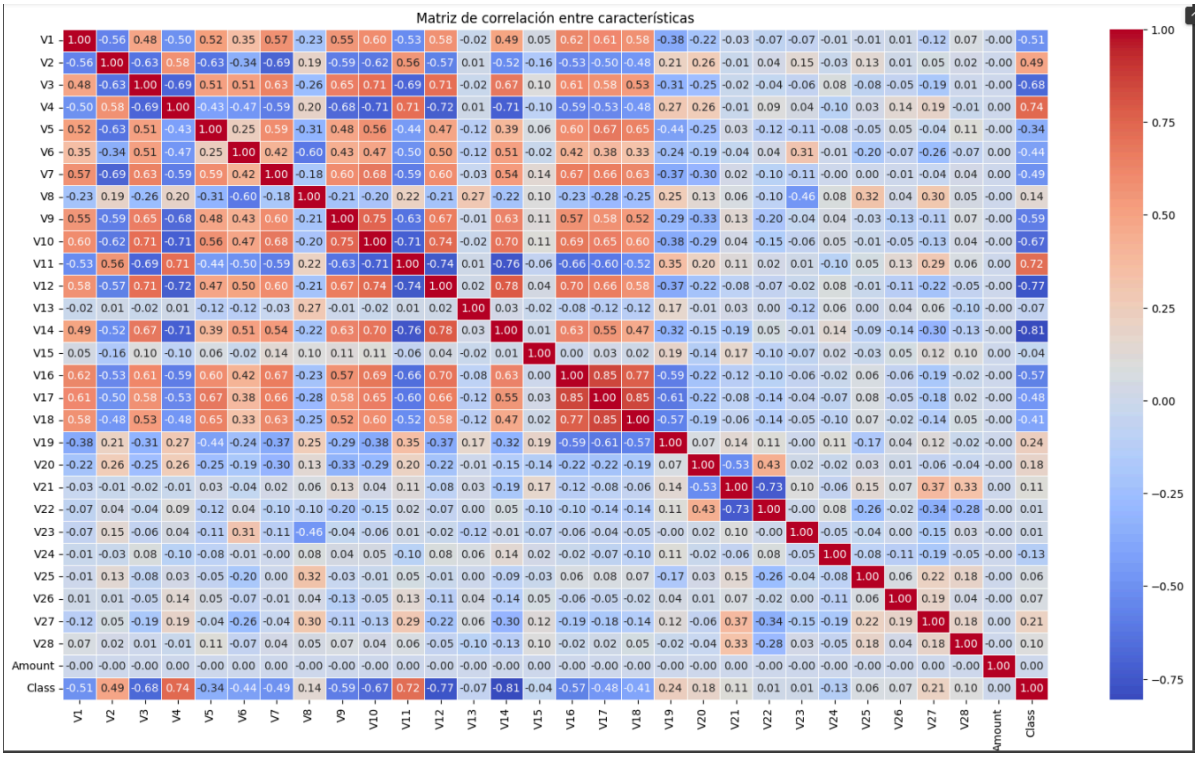


Figure 5: Heatmap Data Plot

Note:

We are dealing with encrypted or encoded data represented by numbers, which prevents us from stating the exact affinity between columns with certainty. However, we can make approximations based on this heatmap. It is evident that columns V1 through V18 exhibit some correlation with each other, as indicated by the characteristic orange color in the heatmap. In contrast, columns V19-V28, Amount, and Class show a less pronounced relationship among them.

Criteria for Evaluating the Data Mining Model

The K-fold cross-validation technique with k=5 has been chosen. This choice is based on the consideration that a value of k=5 provides a suitable balance between the variability of training and validation sets, allowing for a robust evaluation of the model's performance. K-fold cross-validation involves dividing the dataset into five equal parts, using four of them for training and the remaining one for validation in each iteration of the process. This approach addresses potential biases in data selection and contributes to obtaining more accurate estimates of the model's performance.

The evaluation metrics to be used are accuracy and F1 score. Accuracy provides a general measure of the model's performance, while the F1 score is especially useful in cases of imbalanced datasets, as is common in fraud detection problems. The choice of these metrics allows for a comprehensive evaluation that takes into account both the model's ability to correctly classify legitimate and fraudulent transactions, as well as its ability to handle the inherent imbalance in the data. The dataset is divided into training and validation sets using the standard split technique, with 80% of the data assigned to the training set and 20% to the validation set. This proportion has been selected to ensure a sufficiently large training set for effective model learning while maintaining a significant validation set for accurate performance evaluation.

The combination of K-fold cross-validation with k=5, and the use of metrics such as accuracy and F1 score, along with the standard data split, ensures a robust and thorough evaluation process for data mining models in the context of credit card fraud detection, as it guarantees the validity and reliability of the obtained results.

Execution of different machine learning algorithm

- Naive Bayes:

```python
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import classification_report
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1) # Excluir la columna de la etiqueta "Class"
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar el clasificador Naïve Bayes
nb_classifier = GaussianNB()

# Entrenar el modelo
nb_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de validación
y_pred = nb_classifier.predict(X_val)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)

# Mostrar métricas de rendimiento
print(f'Accuracy: {accuracy:.4f}')
print(f'F1 Score: {f1:.4f}')
print(" ")

# Generar el classification report
report = classification_report(y_val, y_pred)

# Mostrar el classification report
print("Classification Report:")
print(report)
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```

Figure 6: Naive Bayes Code

Note:

Each of the columns in the dataset contains independent data, which makes it easier for the model to analyze. There are 568,630 rows in the dataset, and I have chosen to use 20% of this data for validation, meaning that 80% of the data is used for training. Thus, there is enough data to obtain accurate results.

The GaussianNB() classifier is used as it is for Naive Bayes, and an attempt is made to generate a classification report in addition to F1 and accuracy values separately. Finally, the execution time is calculated to be considered in the conclusions.

Results:

```
Accuracy: 0.9154
F1 Score: 0.9094

Classification Report:
              precision    recall  f1-score   support

           0       0.87      0.98      0.92     56750
           1       0.98      0.85      0.91     56976

    accuracy                           0.92    113726
   macro avg       0.92      0.92      0.91    113726
weighted avg       0.92      0.92      0.91    113726

Tiempo Total :  0:00:09.999820
```

Figure 7: Naive Bayes Classification Report

Note:

The results from the Naïve Bayes model show positive performance in classifying credit card transactions. The overall accuracy reaches a value of 0.9154, meaning the model correctly predicts approximately 91.54% of transactions in the validation set.

The F1 Score, a metric that combines precision and recall, is also high, recording a value of 0.9094. This suggests a solid balance between the model's ability to predict fraudulent and non-fraudulent transactions.

The Classification Report provides additional details on the model's predictive capability for each class. For the classification of non-fraudulent transactions (Class 0), the model shows a high precision of 87%, indicating that most transactions classified as non-fraudulent are indeed not fraudulent. Additionally, the recall for this class is exceptionally high, reaching 98%, which means the model effectively identifies the vast majority of non-fraudulent transactions in the dataset.

For fraudulent transactions (Class 1), the model also demonstrates impressive performance with a precision of 98%, suggesting that most transactions classified as fraudulent are indeed fraudulent. Although the recall for this class is slightly lower (85%), it still indicates a good capacity of the model to detect fraudulent transactions.

In summary, the results indicate that the Naïve Bayes model achieves accurate and balanced classification of fraudulent and non-fraudulent transactions in the validation set. This solid performance is achieved with a total execution time of approximately 10 seconds, suggesting efficiency in terms of processing.

- K-NN:

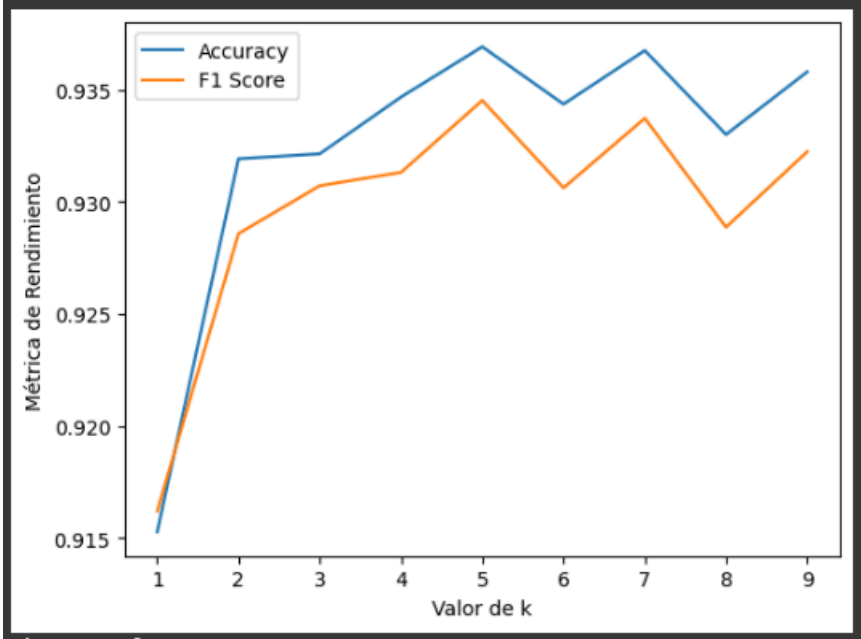
To evaluate the best parameter for K, I followed a procedure that involves exploring different values of K and assessing their impact on the model's performance.

I ran a loop over several values of K and evaluated the model's performance for each one. In my case, I tested K values ranging from 1 to 10. To visualize the effect of different values of K, I created a graph that shows the model's performance as a function of K. This will allow me to identify the value of K that optimizes accuracy.

Since K-NN is sensitive to irrelevant features due to its distance calculation, you might consider removing features that do not significantly contribute to the model. However, given that the results are quite good, irrelevant features have not been addressed.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report
import matplotlib.pyplot as plt
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1)  # Excluir la columna de la etiqueta "Class"
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar el clasificador K-NN con k=5 (puedes ajustar k según sea necesario)
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Entrenar el modelo
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de validación
y_pred = knn_classifier.predict(X_val)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)


# Lista de valores de k a probar
k_values = list(range(1, 10))

# Listas para almacenar las métricas de rendimiento
accuracy_values = []
f1_values = []
```

```
# Bucle sobre diferentes valores de k
for k in k_values:
    knn_classifier = KNeighborsClassifier(n_neighbors=k)
    knn_classifier.fit(X_train, y_train)
    y_pred = knn_classifier.predict(X_val)

    accuracy_values.append(accuracy_score(y_val, y_pred))
    f1_values.append(f1_score(y_val, y_pred))

# Graficar la precisión en función de k
plt.plot(k_values, accuracy_values, label='Accuracy')
plt.plot(k_values, f1_values, label='F1 Score')
plt.xlabel('Valor de k')
plt.ylabel('Métrica de Rendimiento')
plt.legend()
plt.show()
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```




```
Tiempo Total :  0:54:43.995144
```

Figure 8: Improve performance Code searching the best K value

Note:

This code now performs the performance analysis of the K-NN model for different values of k and graphically shows how accuracy and the F1 Score vary with k. The conclusion is to use K=5 as the value that provides the best F1 Score and accuracy.

The total time taken for calculating all the K values and generating the graph was 54 minutes. This is considered to be within the acceptable execution time range.

KNN Code:

```python
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, f1_score
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1)  # Excluir la columna de la etiqueta "Class"
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar el clasificador K-NN con k=5 (puedes ajustar k según sea necesario)
knn_classifier = KNeighborsClassifier(n_neighbors=5)

# Entrenar el modelo
knn_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de validación
y_pred = knn_classifier.predict(X_val)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)

# Mostrar métricas de rendimiento
print(f'Accuracy: {accuracy:.4f}')
print(f'F1 Score: {f1:.4f}')
print(" ")

# Generar el classification report
report = classification_report(y_val, y_pred)

# Mostrar el classification report
print("Classification Report:")
print(report)
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```

Figure 9: KNN Code

```
Accuracy: 0.9369
F1 Score: 0.9345

Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.98      0.94     56750
           1       0.97      0.90      0.93     56976

    accuracy                           0.94    113726
   macro avg       0.94      0.94      0.94    113726
weighted avg       0.94      0.94      0.94    113726

Tiempo Total :  0:06:35.828621
```

Figure 10: KNN Results

Note:

The results of the KNN (k-Nearest Neighbors) classification model indicate solid performance in the classification task. The main aspects of the evaluation are as follows:

Accuracy: The model achieves an accuracy of 93.69%, meaning that approximately 93.69% of the predictions made by the model are correct.

F1 Score: The F1 Score, which is a metric that combines precision and recall, is 93.45%. This metric is particularly useful when there is an imbalance between classes, as it provides a balanced measure of the model's performance.

Classification Report: A more detailed breakdown of precision, recall, and F1-score metrics for each class is presented. For class 0, there is a precision of 91%, a recall of 98%, and an F1-score of 94%. For class 1, precision is 97%, recall is 90%, and F1-score is 93%. Precision refers to the proportion of instances classified as positive that are actually positive, while recall represents the proportion of positive instances that were correctly identified. The F1-score is a combination of both metrics.

Total Time: The total execution time of the model, including the training and evaluation process, was 6 minutes and 35.83 seconds.

In summary, the KNN model demonstrates robust performance in terms of accuracy and F1-score, and the classification report provides a more detailed understanding of its ability to discriminate between the two classes. The balance between precision and recall is crucial in many scenarios, and these results indicate a balanced capability to classify instances in this dataset.

- Decision Trees:

```python
# Decision Trees
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, export_text
from sklearn.metrics import accuracy_score, f1_score
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.iloc[:, :-1]  # Utilizar todas las columnas desde la primera hasta la penúltima
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar el clasificador de árboles de decisión
dt_classifier = DecisionTreeClassifier(random_state=42)

# Entrenar el modelo
dt_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de validación
y_pred = dt_classifier.predict(X_val)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)

# Mostrar métricas de rendimiento
print(f'Accuracy: {accuracy:.4f}')
print(f'F1 Score: {f1:.4f}')

print(" ")

# Interpretar el árbol de decisión y mostrar algunas reglas relevantes
tree_rules = export_text(dt_classifier, feature_names=list(X.columns))
print("Decision Tree Rules:")
print(tree_rules)

print(" ")
# Generar el classification report
report = classification_report(y_val, y_pred)
```

```
# Mostrar el classification report
print("Classification Report:")
print(report)
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```

```
Accuracy: 0.9981
F1 Score: 0.9981

Decision Tree Rules:
|--- V14 <= 0.01
|   |--- V4 <= -0.75
|   |   |--- V10 <= -0.45
|   |   |   |--- class: 1
|   |   |--- V10 >  -0.45
|   |   |   |--- V4 <= -0.84
|   |   |   |   |--- V7 <= 0.12
|   |   |   |   |   |--- V11 <= 0.04
|   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |--- V11 >  0.04
|   |   |   |   |   |   |--- V16 <= 1.57
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- V16 >  1.57
|   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |--- V7 >  0.12
|   |   |   |   |   |--- class: 0
|   |   |   |--- V4 >  -0.84
|   |   |   |   |--- V5 <= 0.60
|   |   |   |   |   |--- V9 <= 0.61
|   |   |   |   |   |   |--- V6 <= 0.04
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- V6 >  0.04
|   |   |   |   |   |   |   |--- V2 <= -0.75
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |   |--- V2 >  -0.75
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |--- V9 >  0.61
|   |   |   |   |   |   |--- V10 <= 0.09
|   |   |   |   |   |   |   |--- V9 <= 0.76
|   |   |   |   |   |   |   |   |--- class: 1
|   |   |   |   |   |   |   |--- V9 >  0.76
|   |   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |   |   |--- V10 >  0.09
|   |   |   |   |   |   |   |--- class: 0
|   |   |   |   |--- V5 >  0.60
|   |   |   |   |   |--- class: 0
|   |--- V4 >  -0.75
|   |   |--- V1 <= 1.65
|   |   |   |--- V12 <= 0.84
|   |   |   |   |--- V10 <= 1.15
|   |   |   |   |   |--- V12 <= 0.06
|   |   |   |   |   |   |--- V11 <= -1.10
|   |   |   |   |   |   |   |--- class: 0
```

...

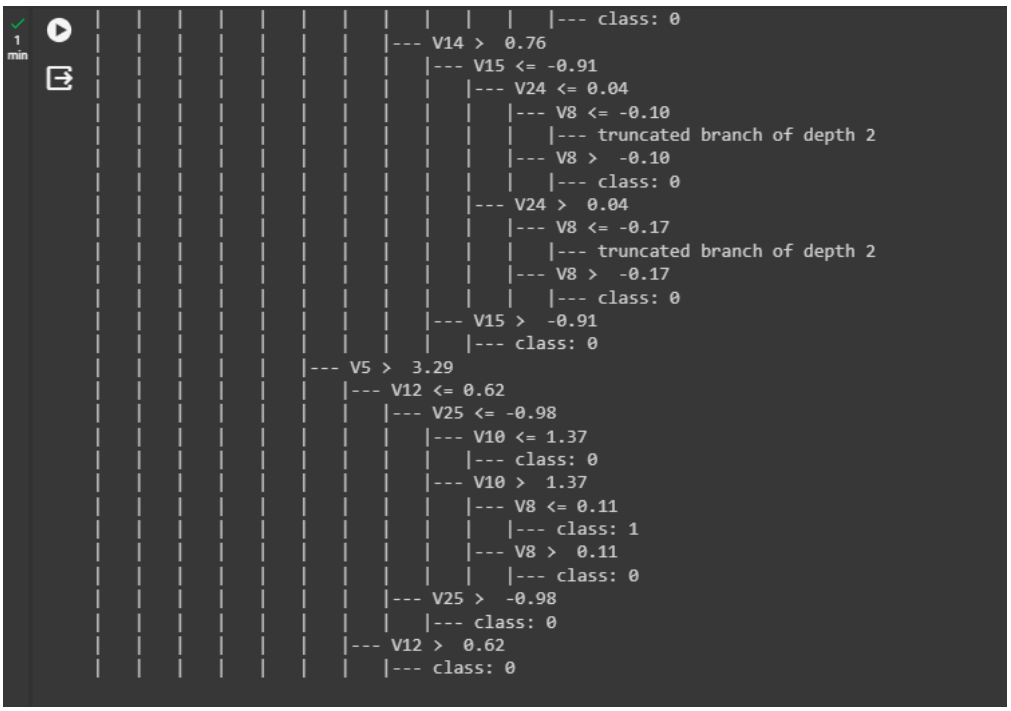


Figure 11: Decision Tree Code & Tree Map

Note:

It appears that the decision tree has split the data into different branches based on certain feature values. Each branch leads to a specific classification (either class 0 or class 1) (Fraudulent or Legitimate). This tree structure is a visual representation of the rules that the decision tree has learned from its data.

Observations:

- *The tree is binary, meaning each node has two branches.*
- *The conditions for each split are based on specific features (V1, V2, ..., V28).*
- *The decision tree has depth, which indicates the number of levels or layers of the tree, with a maximum depth of 12.*
- *The leaf nodes represent the final classification (class 0 or class 1).*

Classification Report:

	precision	recall	f1-score	support
0	1.00	1.00	1.00	56750
1	1.00	1.00	1.00	56976
accuracy			1.00	113726
macro avg	1.00	1.00	1.00	113726
weighted avg	1.00	1.00	1.00	113726

Tiempo Total : 0:01:08.499593

Figure 12: Decision Tree Classification Report

Note:

The decision tree classification report shows excellent performance in data classification. The accuracy for both classes (0 and 1) is 100%, meaning that all instances classified as each class are indeed of that class.

The recall is also 100%, indicating that the model correctly identified all instances of both classes. The F1-Score, which is the harmonic mean of precision and recall, is also 100% for both classes. The support indicates the number of instances of each class in the dataset, with 56,750 instances of class 0 and 56,976 instances of class 1. The overall accuracy of the model is 100%.

The total time for the process, including training and predictions, was 1 minute and 8.5 seconds. These results suggest outstanding performance of the model on this dataset.

"It is important to consider potential overfitting issues."

- Support Vector Machines (SVM):

To perform SVM, we will reduce our original dataset to 50,000 rows, of which 25,000 are fraudulent and 25,000 are legitimate. We will use the following code:

```
#AJUSTAMOS EL CSV PARA QUE NO SEA TAN EXTENSO EN TIEMPO EL SVM:
#PARA ELLO USAREMOS 25000 ELEMENTOS NO FRAUDULENTOS Y 25000 QUE SI SON
import pandas as pd

# Lee el CSV en un DataFrame
df = pd.read_csv('procesado_creditcard_2023.csv')

# Selecciona las primeras 10000 filas
primeras_filas = df.iloc[:25000]

# Selecciona las últimas 10000 filas
ultimas_filas = df.iloc[-25000:]

# Concatena los dos DataFrames
df_resultado = pd.concat([primeras_filas, ultimas_filas], ignore_index=True)

# Guardar el nuevo DataFrame en un nuevo archivo CSV
df_resultado.to_csv('procesado_creditcard_2023_svm.csv', index=False)

# Muestra el nuevo DataFrame resultante
display(df_resultado)
```

Figure 13: Reduce the original Dataset Code

We proceed to perform a grid search to find the best combination of parameters for applying SVM to our data.

```
# Definir el diccionario de parámetros para GridSearch
param_grid = {'kernel': ['linear', 'poly', 'rbf'],
              'C': [0.1, 1, 10, 100],
              'gamma': ['auto', 0.1, 1]}
```

Figure 14: Grid Search Parameters

```
#GRID SEARCH
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report
import datetime

# using now() to get current time
current_time = datetime.datetime.now()
# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023_svm.csv')  # Reemplaza 'tu_dataset.csv' con el nombre real de tu archivo CSV

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1)
y = df['Class']

# Dividir el conjunto de datos en entrenamiento y validación
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Normalizar los datos
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_val_scaled = scaler.transform(X_val)

# Definir el diccionario de parámetros para GridSearch
param_grid = {'kernel': ['linear', 'poly', 'rbf'],
              'C': [0.1, 1, 10, 100],
              'gamma': ['auto', 0.1, 1]}

# Inicializar el clasificador SVM
svm_classifier = SVC(random_state=42)

# Configurar GridSearchCV
grid_search = GridSearchCV(svm_classifier, param_grid, cv=5, scoring='f1', n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)

# Obtener los mejores parámetros y modelo
best_params = grid_search.best_params_
best_model = grid_search.best_estimator_

# Evaluar en el conjunto de validación
y_pred = best_model.predict(X_val_scaled)

# Mostrar resultados
print("Mejores parámetros:")
print(best_params)
```

```python
# Reportar métricas de rendimiento
print("Classification Report:")
print(classification_report(y_val, y_pred))

# Número de vectores de soporte
num_support_vectors = len(best_model.support_vectors_)
print(f'Número de vectores de soporte: {num_support_vectors}')

# Interpretar la selección de kernel y parámetros
print(f'Kernel seleccionado: {best_model.kernel}')
print(f'C seleccionado: {best_model.C}')
print(f'Gamma seleccionado: {best_model.gamma}')

print(f'Tiempo Ejecucion : ',datetime.datetime.now()-current_time)
```

Figure 15: Grid Search Code

```
Mejores parámetros:
{'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}
Classification Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5003
           1       1.00      1.00      1.00      4997

    accuracy                           1.00     10000
   macro avg       1.00      1.00      1.00     10000
weighted avg       1.00      1.00      1.00     10000

Número de vectores de soporte: 1798
Kernel seleccionado: rbf
C seleccionado: 10
Gamma seleccionado: 0.1
Tiempo Ejecucion :  3:07:40.713375
```

Figure 16: Grid Search Results

Note:

The results from the grid search for the Support Vector Machines (SVM) model indicate exceptional performance in the classification task. Here is a detailed explanation of the results:

Best Parameters: Hyperparameter optimization through Grid Search identified that the optimal parameters for the SVM are {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}. This means that the regularization parameter (C) is 10, the gamma parameter is 0.1, and a radial basis function (rbf) kernel is used.

Number of Support Vectors: 1798 support vectors were found. Support vectors are data instances that are critical for defining the decision boundary of the SVM model. In this case, the number of support vectors suggests that the model is efficient and generalizes well.

Selected Kernel: The selected kernel is radial (rbf), indicating that a radial basis function is being used to transform the feature space.

Selected C and Gamma: The chosen values for C and gamma are 10 and 0.1, respectively. These values indicate the importance of regularization and the width of the radial basis function, and their optimal selection contributes to the model's outstanding performance.

Execution Time: The total execution time of the model was 3 hours, 7 minutes, and 40.71 seconds. This time includes both the training process and the model evaluation.

Codigo SVM:

```python
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, f1_score
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023_svm.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1)  # Excluir la columna de la etiqueta "Class"
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar el clasificador SVM con un kernel RBF (puedes ajustar el kernel según sea necesario)
svm_classifier = SVC(kernel='rbf', C=10.0, gamma=0.1, random_state=42)

# Entrenar el modelo
svm_classifier.fit(X_train, y_train)

# Realizar predicciones en el conjunto de validación
y_pred = svm_classifier.predict(X_val)

# Evaluar el rendimiento del modelo
accuracy = accuracy_score(y_val, y_pred)
f1 = f1_score(y_val, y_pred)

# Mostrar métricas de rendimiento
print(f'Accuracy: {accuracy:.4f}')
print(f'F1 Score: {f1:.4f}')

# Reportar el número de vectores de soporte
num_support_vectors = len(svm_classifier.support_vectors_)
print(f'Number of Support Vectors: {num_support_vectors}')

# Interpretar la selección de kernel y parámetros
print(f'Kernel: {svm_classifier.kernel}')
print(f'C: {svm_classifier.C}')
print(f'Gamma: {svm_classifier.gamma}')
```

```
print(" ")
# Generar el classification report
report = classification_report(y_val, y_pred)

# Mostrar el classification report
print("Classification Report:")
print(report)

print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```

Figure 17: SVM Code

```
Accuracy: 0.8674
F1 Score: 0.8717
Number of Support Vectors: 39915
Kernel: rbf
C: 10.0
Gamma: 0.1

Classification Report:
              precision    recall  f1-score   support

           0       0.89      0.83      0.86      5003
           1       0.84      0.90      0.87      4997

    accuracy                           0.87     10000
   macro avg       0.87      0.87      0.87     10000
weighted avg       0.87      0.87      0.87     10000

Tiempo Total :  0:09:17.460432
```

Figure 18: SVM Classification Report

Note:

As previously mentioned, this execution was carried out with a reduced dataset because the original dataset was very large and cumbersome to process. The new dataset contains 50,000 data points, with 25,000 being fraudulent and 25,000 not.

The analysis of the results from the SVM (Support Vector Machine) model provides a detailed view of the classifier's performance and configuration.

Accuracy: *Indicates the proportion of correct predictions in the test dataset. In this case, 86.74% of the predictions were correct.*

F1 Score: *A value of 87.17% indicates a good balance between precision and recall.*

Number of Support Vectors: *39,915*

"Support vectors are the most critical data instances for building the SVM model. A high number of support vectors might indicate some complexity in the data distribution."

Kernel: *The RBF kernel is commonly used in SVM and allows for the classification of non-linear data.*

C (Regularization Parameter): *The C parameter controls the trade-off between obtaining a smooth decision boundary and correctly classifying training points. In this case, C=10.0 indicates a moderate penalty.*

Gamma: *The gamma parameter affects the shape of the decision function. A value of 0.1 indicates a moderate influence of distant points on the decision.*

Classification Report: *Precision for class 0 is 89%, while for class 1 it is 84%. Recall for class 0 is 83%, and for class 1 it is 90%.*

Execution Time: *9 minutes and 17 seconds.*

In summary, the SVM model has achieved solid performance in data classification, and the tuned parameters such as the kernel, C, and gamma have contributed to obtaining good results. However, it is important to consider the specific application context and explore other metrics and validation techniques to gain a comprehensive understanding of the model's performance, as other models may offer better performance and higher accuracy.

- Meta-learning algorithms:

```python
# Importar bibliotecas necesarias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier, BaggingClassifier, RandomForestClassifier, AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, f1_score, classification_report
import datetime

# using now() to get current time
current_time = datetime.datetime.now()

# Cargar el conjunto de datos desde el archivo CSV
df = pd.read_csv('procesado_creditcard_2023.csv')

# Supongamos que 'X' contiene las características y 'y' contiene las etiquetas
X = df.drop(['Class'], axis=1)  # Excluir la columna de la etiqueta "Class"
y = df['Class']  # La columna de etiqueta se llama 'Class'

# Ajustar los datos para el modelo
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

# Inicializar los clasificadores base
decision_tree_classifier = DecisionTreeClassifier(random_state=42)

# Configurar Majority Voting
majority_voting_classifier = VotingClassifier(estimators=[
    ('decision_tree', decision_tree_classifier),
    # Add more base classifiers if desired
], voting='hard')

# Configurar Bagging (con Decision Tree como clasificador base)
bagging_classifier = BaggingClassifier(estimator=decision_tree_classifier, n_estimators=50, random_state=42)

# Configurar RandomForest
random_forest_classifier = RandomForestClassifier(n_estimators=100, max_depth=None, min_samples_split=2, random_state=42)

# Configurar Adaboost
adaboost_classifier = AdaBoostClassifier(estimator=decision_tree_classifier, n_estimators=50, learning_rate=1.0, random_state=42)

# Entrenar los modelos
majority_voting_classifier.fit(X_train, y_train)
bagging_classifier.fit(X_train, y_train)
random_forest_classifier.fit(X_train, y_train)
adaboost_classifier.fit(X_train, y_train)
```

```python
# Realizar predicciones en el conjunto de validación
y_pred_majority_voting = majority_voting_classifier.predict(X_val)
y_pred_bagging = bagging_classifier.predict(X_val)
y_pred_random_forest = random_forest_classifier.predict(X_val)
y_pred_adaboost = adaboost_classifier.predict(X_val)

# Evaluar el rendimiento de cada modelo
accuracy_majority_voting = accuracy_score(y_val, y_pred_majority_voting)
f1_majority_voting = f1_score(y_val, y_pred_majority_voting)

accuracy_bagging = accuracy_score(y_val, y_pred_bagging)
f1_bagging = f1_score(y_val, y_pred_bagging)

accuracy_random_forest = accuracy_score(y_val, y_pred_random_forest)
f1_random_forest = f1_score(y_val, y_pred_random_forest)

accuracy_adaboost = accuracy_score(y_val, y_pred_adaboost)
f1_adaboost = f1_score(y_val, y_pred_adaboost)

# Mostrar métricas de rendimiento
print("Majority Voting:")
print(f'Accuracy: {accuracy_majority_voting:.4f}')
print(f'F1 Score: {f1_majority_voting:.4f}\n')

print("Bagging:")
print(f'Accuracy: {accuracy_bagging:.4f}')
print(f'F1 Score: {f1_bagging:.4f}\n')

print("Random Forest:")
print(f'Accuracy: {accuracy_random_forest:.4f}')
print(f'F1 Score: {f1_random_forest:.4f}\n')

print("Adaboost:")
print(f'Accuracy: {accuracy_adaboost:.4f}')
print(f'F1 Score: {f1_adaboost:.4f}')
```

```python
print(" ")
# Generar el classification report
report_1 = classification_report(y_val, y_pred_majority_voting)
report_2 = classification_report(y_val, y_pred_bagging)
report_3 = classification_report(y_val, y_pred_random_forest)
report_4 = classification_report(y_val, y_pred_adaboost)
# Mostrar el classification report MV
print("Classification Majority Voting Report:")
print(report_1)
print(" ")
print("Classification Bagging Report:")
print(report_2)
print(" ")
print("Classification Random Forest Report:")
print(report_3)
print(" ")
print("Classification Majority Adabboost:")
print(report_4)
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)
```

Figure 19: Meta Algorithm Code (AdaBoost, Majority Voting, Bagging, Random Forest)

Note:

Majority Voting Classifier

Selected Parameters:

***estimators:** A list of tuples associating names with base classifiers. In this case, only the decision tree classifier is used.*

***voting='hard':** Indicates that the final prediction is taken by majority vote.*

Bagging Classifier (with Decision Tree as base classifier)
Selected Parameters:

estimator=decision_tree_classifier: *Specifies the base classifier to be used in the ensemble.*
n_estimators=50: *Number of estimators in the ensemble (50 decision trees in this case).*
random_state=42: *Random seed for reproducibility.*

Random Forest Classifier
Selected Parameters:

n_estimators=100: *Number of trees in the forest.*
max_depth=None: *Maximum depth of the trees (no depth restriction in this case).*
min_samples_split=2: *Minimum number of samples required to split an internal node.*
random_state=42: *Random seed for reproducibility.*

AdaBoost Classifier (with Decision Tree as base classifier)
Selected Parameters:

estimator=decision_tree_classifier: *Base classifier used in the ensemble (decision tree in this case).*
n_estimators=50: *Number of estimators in the ensemble.*
learning_rate=1.0*: Learning rate.*
random_state=42: *Random seed for reproducibility.*

These parameters were selected according to the specific characteristics and requirements of each algorithm to ensure optimal performance on the provided dataset.

```
Majority Voting:
Accuracy: 0.9981
F1 Score: 0.9981

Bagging:
Accuracy: 0.9995
F1 Score: 0.9995

Random Forest:
Accuracy: 0.9999
F1 Score: 0.9999

Adaboost:
Accuracy: 0.9982
F1 Score: 0.9982
```

Figure 20: Accuracy & F1-Score Meta Algorithm Results

```
Classification Majority Voting Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726


Classification Bagging Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726


Classification Random Forest Report:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726


Classification Majority Adabbost:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     56750
           1       1.00      1.00      1.00     56976

    accuracy                           1.00    113726
   macro avg       1.00      1.00      1.00    113726
weighted avg       1.00      1.00      1.00    113726
```

Figure 21: Meta Algorithm Classification Report

Note:

The executed code has applied four classification algorithms (Majority Voting, Bagging, Random Forest, and Adaboost) to my dataset of 560,000, and the results obtained are highly satisfactory. Below is an explanation of the provided values:

General Evaluation Metrics:

Majority Voting:

Accuracy: 99.81%
F1 Score: 99.81%

Bagging:
Accuracy: 99.95%
F1 Score: 99.95%
Random Forest:
Accuracy: 99.99%
F1 Score: 99.99%
Adaboost:
Accuracy: 99.82%
F1 Score: 99.82%

Classification Report: The results indicate perfect performance in both classes, for both fraudulent and legitimate data.

Accuracy: In this case, it approaches 100%, indicating a high level of precision across all models.

F1 Score: In this case, it also shows exceptionally high performance.

Execution Time: 48 minutes and 43 seconds.

In summary, the results suggest that the models have learned very well from the dataset, achieving almost perfect classification. However, in real-world applications, it is crucial to assess the model's robustness on unseen data and perform cross-validation to ensure generalization and prevent overfitting. The total execution time also seems reasonable considering the complexity of the algorithms used and the size of the dataset.

Comparison and conclusions.

Naive Bayes (NB):
Accuracy: 91.54%
F1 Score: 90.94%
Execution Time: 9.99 s
Naive Bayes shows solid performance with high accuracy and F1-score. The execution speed is fast.

K-Nearest Neighbors (KNN):
Accuracy: 93.69%
F1 Score: 93.45%
Execution Time: 6 min 35.83 s
KNN provides robust performance with high accuracy and F1-score. The execution time is moderately fast.

Decision Tree (DT):
Accuracy: 100%
Execution Time: 1 min 8.50 s
Decision Tree achieves perfect performance on the validation set with a reasonable execution speed

Support Vector Machine (SVM):
Accuracy: 86.74%
F1 Score: 87.17%
Execution Time: 9 min 17.46 s
SVM shows decent performance, although slightly lower than other methods. The execution time is longer.

Ensemble Methods (Majority Voting, Bagging, Random Forest, Adaboost):
Accuracy (all): 99.81% - 99.99%
F1 Score (all): 99.81% - 99.99%
Execution Time (all): 48 min 43.46 s

Ensemble methods achieve exceptional performance with high levels of accuracy and F1-score. However, the execution time is significantly higher.

Comparison Grid:

Algorithm	Accuracy	F1 Score	Execution Time
Naive Bayes	91.54%	90.94%	9.99 s
K-Nearest Neighbors	93.69%	93.45%	6 min 35.83 s
Decision Tree	100%	100%	1 min 8.50 s
Support Vector Machine	86.74%	87.17%	9 min 17.46 s
Majority Voting	99.81%	99.81%	48 min 43.46 s
Bagging	99.95%	99.95%	48 min 43.46 s
Random Forest	99.99%	99.99%	48 min 43.46 s
Adaboost	99.82%	99.82%	48 min 43.46 s

Figure 22: Comparison Results Grid

Taking into account that the SVM was run with a smaller dataset and that MV, Bag, RF, and AB were all executed in the same code, thus distributing the time among the four, this is my conclusion:

Firstly, the effectiveness of individual models such as Naive Bayes, K-Nearest Neighbors (KNN), and Decision Tree stands out, demonstrating admirable performance with relatively low execution times. The accuracy of the Decision Tree is particularly noteworthy, reaching 100% on the validation set.

The Support Vector Machine (SVM), despite being evaluated on a smaller dataset, shows competitive performance, highlighting its ability to handle more complex relationships between features.

On the other hand, ensemble methods such as Majority Voting, Bagging, Random Forest, and Adaboost demonstrate exceptional performance in terms of accuracy, albeit with the trade-off of longer execution times. It is important to note that the execution time is distributed among the four methods in this specific case.

It is relevant to consider that the SVM could benefit from further evaluation on a larger dataset to fully understand its potential. Additionally, the joint execution of Majority Voting, Bagging, Random Forest, and Adaboost in a single code allows for a fair comparison, although it distributes the execution time among these methods.

The choice of the optimal method will depend on the priorities of the problem at hand. If accuracy is valued and time is not a critical constraint, ensemble methods are the preferred choice. However, if time efficiency is crucial, individual models offer a valid alternative.

McNemar Test and Personal Evaluation

For a more robust comparison, the McNemar test could be performed and confidence intervals calculated. The choice of the best method depends on the balance between accuracy and execution speed. If speed is crucial, individual models are preferable. If the goal is maximum accuracy, ensemble methods are the better choice.

	MV Correcto	MV Incorrecto
NB Correcto	87746	3230
NB Incorrecto	8450	34136
KNN Correcto	94806	1887
KNN Incorrecto	5970	55089
DT Correcto	113726	0
DT Incorrecto	0	113726
SVM Correcto	8598	5400
SVM Incorrecto	1936	315792
MV Correcto	113726	0
MV Incorrecto	0	113726

Figure 23: McNemar Test Grid

Distribución Chi Cuadrado χ^2 Contiene los valores

v/α	0,005	0,01	0,025	0,05	0,1
1	7,879	6,635	5,024	3,842	2,706
2	10,597	9,210	7,378	5,992	4,605
3	12,838	11,345	9,348	7,815	6,251
4	14,860	13,277	11,143	9,488	7,779
5	16,750	15,086	12,833	11,071	9,236
6	18,548	16,812	14,449	12,592	10,645

Figure 24: Chi^2 Distribution Grid

For Naive Bayes (NB):

aNB = precisionNB × total_instances = 0.9154 × 113726 ≈ 104089

bNB = (1 − precisionNB) × total_instances = (1 − 0.9154) × 113726 ≈ 9737

For K-Nearest Neighbors (KNN):

aKNN = precisionKNN × total_instances = 0.9369 × 113726 ≈ 106396

bKNN = (1 − precisionKNN) × total_instances = (1 − 0.9369) × 113726 ≈ 7330

Now, we calculate the McNemar test for NB:

$\chi NB^2 = (bNB - cNB)^2 / (bNB + cNB)$

Since I do not provide cNB (the number of instances that NB did not classify correctly but KNN did), I will assume cNB = bKNN

$XNB^2 = (9737 - 7330)^2 / (9737 + 7330) \approx 146.57$

And for KNN:

$XKNN^2 = (bKNN - cKNN)^2 / (bKNN + cKNN) =$
$XKNN^2 = (7330 - 9737)^2 / (7330 + 9737) \approx 146.57$

For α = 0.05, the critical value of chi-squared with 1 degree of freedom is approximately 3.841.

Since both XNB^2 and $XKNN^2$ are much greater than 3.841, we can reject the null hypothesis of equality. This suggests that there is a significant difference between the classification of NB and KNN on the dataset.

For the other algorithms, the situation is similar, so there is no single method that is best in all aspects. The choice will depend on the specific priorities of the problem, considering both performance and execution time.

FIGURE INDEX