

MIDA - 2023

DETECCIÓN DE FRAUDE EN TRANSACCIONES DE TARJETA DE CRÉDITO



Francisco José Martín

ÍNDICE

Introducción al Proyecto	2
Descripción del origen de los datos	2
Descripción de preprocesamiento de datos	3
Gráficos	
Fraudulentos/Legitimos	4
Caja y Bigotes	5
Mapa de calor	8
Criterio de evaluación del modelo de minería de datos	9
Ejecución de diferentes métodos de machine learning	
NB	10
KNN	13
DT	17
SVM	21
MV	27
Comparison and conclusions.	31

Introducción al Proyecto

Título del Proyecto: Detección de Fraudes de Tarjeta de Crédito con Inteligencia Artificial

Descripción:

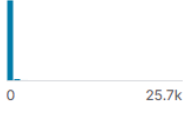
Este proyecto tiene como objetivo desarrollar un sistema de detección de fraudes de tarjeta de crédito mediante el uso de técnicas de Inteligencia Artificial (IA). Utilizarás un conjunto de datos proporcionado por Kaggle, que contiene información sobre transacciones de tarjetas de crédito etiquetadas como legítimas o fraudulentas.

Descripción del origen de los datos

Los datos para realizar el proyecto se han obtenido de Kaggle. Consisten en una tabla que contiene un identificador numérico por cada línea, veintiocho columnas con un valor numérico no descifrable, una columna del total de gasto efectuado en cada transacción y por último una columna que indica si la transacción es fraudulenta o no.

El link a los datos es el siguiente:

<https://www.kaggle.com/datasets/nelgiryewithana/credit-card-fraud-detection-dataset-2023/>

# Time	# V1	# V2	# V28	# Amount	# Class
Number of seconds elapsed between this transaction and the first transaction in the dataset	may be result of a PCA Dimensionality reduction to protect user identities and sensitive features(v1-v28)		abc	Transaction amount	1 for fraudulent transactions, 0 otherwise
					
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

Descripción de preprocesamiento de datos

En el proceso de preprocesamiento de los datos, se ha optado por retener todas las columnas disponibles, ya que cada una de las 28 variables numéricas, junto con las columnas que representan el monto de la transacción y la indicación de fraude, se consideran esenciales para abordar el problema en cuestión. La única columna que se extraerá será la del tiempo transcurrido entre la primera y la última transacción llamada "Time". Se ha llevado a cabo una revisión exhaustiva para asegurar que no haya datos redundantes o irrelevantes que pudieran ser eliminados sin comprometer la calidad del conjunto de datos. En este sentido, se confirma que todas las variables desempeñan un papel crucial en la representación integral del problema de detección de fraudes. No se ha realizado ninguna simplificación de valores, ya que cada uno de ellos aporta información valiosa para el análisis. Además, se ha verificado que no existan valores faltantes en el conjunto de datos, asegurando así la integridad y la calidad de la información utilizada en el estudio.

	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 x 30 columns					...				

En el contenido del proyecto habrán 3 datasets.

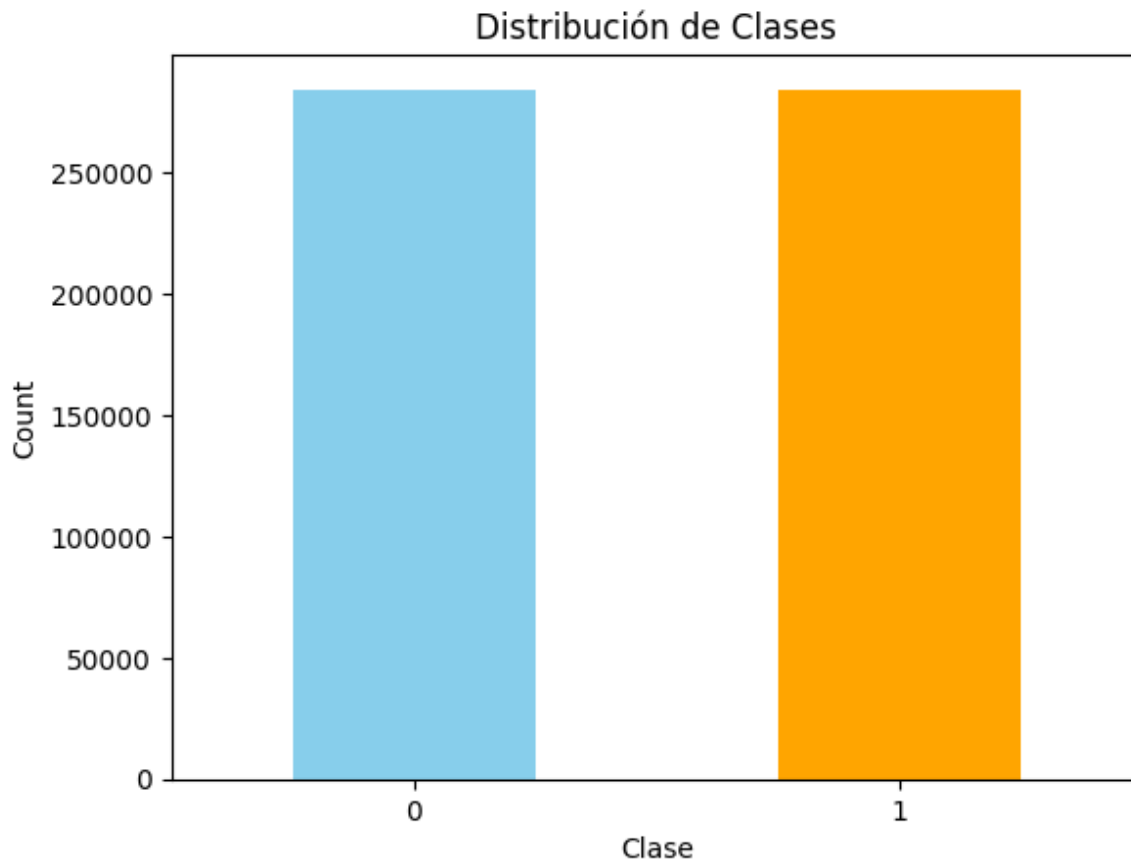
creditcard_2023.csv : Contiene los datos sin procesar

procesado_creditcard_2023.csv : Contiene los datos procesados

procesado_creditcard_2023_svm.csv: Contien los datos procesados pero solo 50000 lineas

Gráficos

Fraudulento / Legítimos

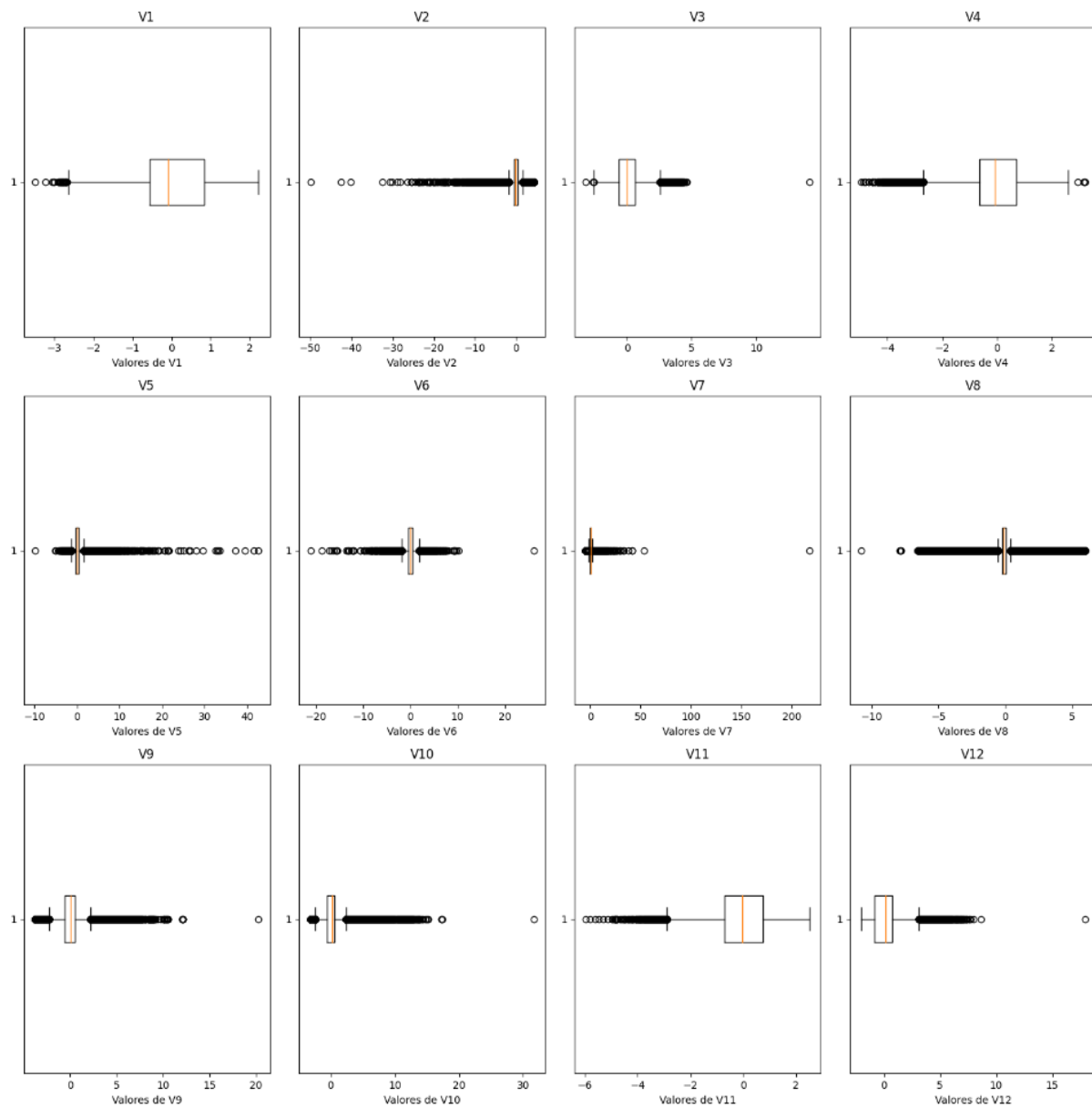


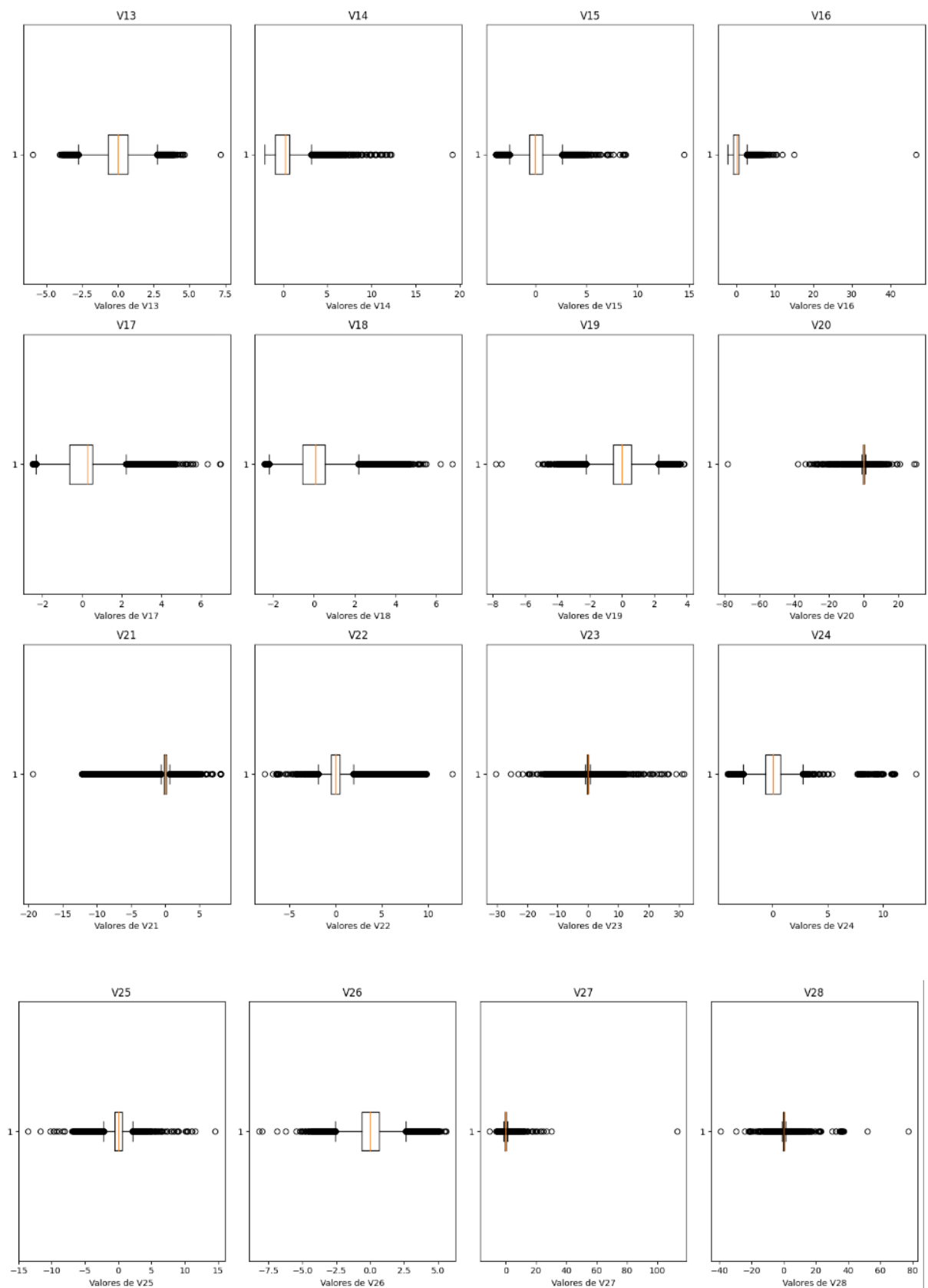
Nota:

El conjunto de datos en cuestión presenta una distribución equitativa con más de 250,000 instancias de transacciones fraudulentas y una cantidad igualmente significativa de más de 250,000 transacciones legítimas. Esta proporción equilibrada ofrece un escenario ideal para el desarrollo de un modelo de detección de fraudes en transacciones de tarjetas de crédito, ya que permite entrenar a la inteligencia artificial con un conjunto de datos que refleja de manera precisa la realidad, considerando tanto casos de fraudes como transacciones legítimas. La amplitud de los datos, al superar las 500,000 instancias, confiere al modelo una base sustancial para aprender patrones complejos y sutilezas en los datos, proporcionando así la oportunidad de desarrollar un sistema robusto y preciso de detección de fraudes que pueda generalizar de manera efectiva a situaciones del mundo real. Este balance y volumen considerable de datos contribuyen significativamente a la capacidad de la IA para aprender de manera más completa y representativa.

Se considera una reducción del volumen de datos para entrenar el modelo SVM, especialmente considerando la naturaleza de SVM y las posibles limitaciones de tiempo asociadas con conjuntos de datos más grandes. SVM puede ser computacionalmente intensivo, y al reducir el volumen de datos, se pueden obtener beneficios en términos de eficiencia y velocidad de entrenamiento.

Distribución de los datos encriptados





Nota:

El análisis de la distribución de las variables V1 hasta V28 proporciona una visión detallada de la variabilidad de los datos en cada una de estas dimensiones. Al examinar los gráficos de distribución, podemos observar la forma y la dispersión de los valores en cada variable.

Por ejemplo, en las variables V1, V4 y V11, se aprecia una distribución casi simétrica con un pico pronunciado alrededor de ciertos valores, mientras que otras variables como V2, V5, V6, V7, V8, V21, V23, V27 y V28 exhiben distribuciones más sesgadas.

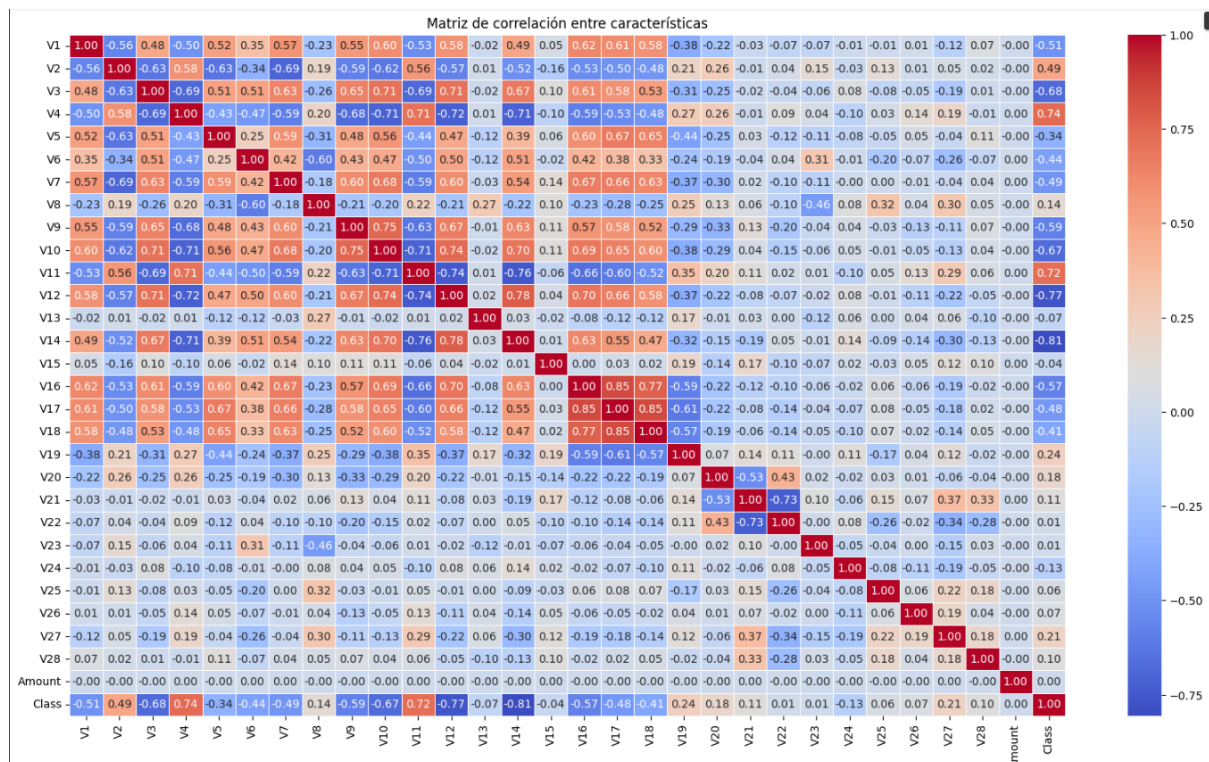
Además, notamos que la mayoría de las variables presentan la mayor concentración de datos en el rango de -2 a 2, con algunos picos pronunciados más distantes. Es relevante señalar que la mediana de todos los datos se sitúa en el valor 0, indicando una distribución centrada.

Particularmente, destaco la gráfica de los valores V1 debido a que muestra menos valores en los cuartiles y presenta algunos valores atípicos en la zona del -3. Este comportamiento puede indicar una mayor variabilidad o sesgo en esta variable en comparación con otras.

Asimismo, algunos gráficos como V5 o V20 muestran un notable número de valores atípicos, lo que sugiere la presencia de datos inusuales que podrían estar asociados con posibles fraudes. La observación de histogramas proporciona información adicional al revelar la frecuencia de los valores en intervalos específicos, permitiéndonos identificar posibles sesgos, outliers y patrones de concentración de datos.

Este análisis detallado es crucial para comprender la naturaleza de las variables y orientará las decisiones futuras en el preprocesamiento y la selección de características para el entrenamiento del modelo de detección de fraudes en transacciones de tarjetas de crédito.

Mapa de calor



Nota:

Estamos manejando datos cifrados o encriptados mediante números, lo que nos impide afirmar con certeza la afinidad exacta entre columnas. Sin embargo, podemos realizar aproximaciones basándonos en este mapa de calor.

Se evidencia que las columnas V1 hasta V18 presentan cierta correlación entre sí, como indica el característico color anaranjado en el mapa de calor. En contraste, las columnas V19-V28, Amount y Class muestran una relación menos pronunciada entre ellas.

Criterio de evaluación del modelo de minería de datos

Se ha optado por emplear la técnica de validación cruzada K-fold con $k=5$. Esta elección se basa en la consideración de que un valor de $k=5$ proporciona un equilibrio adecuado entre la variabilidad de los conjuntos de entrenamiento y validación, permitiendo una evaluación robusta del rendimiento del modelo. La validación cruzada K-fold implica dividir el conjunto de datos en cinco partes iguales, utilizando cuatro de ellas para entrenamiento y la restante para validación en cada iteración del proceso. Este enfoque aborda posibles sesgos en la selección de datos y contribuye a obtener estimaciones más precisas del rendimiento del modelo.

Como métricas de evaluación, se utilizarán la precisión (accuracy) y la puntuación F1. La precisión proporciona una medida general del rendimiento del modelo, mientras que la puntuación F1 es especialmente útil en casos de conjuntos de datos desbalanceados, como es común en problemas de detección de fraudes. La elección de estas métricas permite una evaluación integral que tiene en cuenta tanto la capacidad del modelo para clasificar correctamente las transacciones legítimas como fraudulentas, así como su capacidad para manejar el desbalance inherente en los datos. El conjunto de datos se divide en conjuntos de entrenamiento y validación utilizando la técnica de separación estándar, asignando el 80% de los datos al conjunto de entrenamiento y el 20% al conjunto de validación. Esta proporción se ha seleccionado para garantizar un conjunto de entrenamiento lo suficientemente grande para el aprendizaje efectivo del modelo, mientras se mantiene un conjunto de validación significativo para la evaluación precisa de su rendimiento.

La combinación de la validación cruzada K-fold, con $k=5$, y el uso de métricas como la precisión y la puntuación F1, junto con la división estándar del conjunto de datos, asegura un proceso de evaluación robusto y exhaustivo de los modelos de minería de datos en el contexto de la detección de fraudes de tarjeta de crédito ya que garantiza la validez y confiabilidad de los resultados obtenidos

Ejecución de diferentes métodos de machine learning

- Naïve Bayes:

```
# 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)
```

Nota:

Cada una de las columnas del dataset son datos independientes, por lo que es mas fácil de analizar para el modelo. Existen 568.630 filas en el dataset, y he optado por utilizar el 20% de esos datos como validación, por lo que el 80% de esos datos son datos de entrenamiento. Por lo que sí que hay suficientes datos para obtener unos resultados correctos.

Se utiliza el clasificador GaussianNB() puesto que es para Natives Bayes y se intenta obtener un clasification report además de los valores de F1 y accuracy por separado. Por último se calcula el tiempo de ejecución para tener en cuenta en conclusiones.

Resultados:

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				

Nota:

Los resultados del modelo Naïve Bayes muestran un rendimiento positivo en la clasificación de transacciones de tarjetas de crédito. La precisión global, conocida como Accuracy, alcanza un valor de 0.9154, lo que significa que el modelo predice correctamente alrededor del 91.54% de las transacciones en el conjunto de validación.

El F1 Score, una métrica que combina precisión y recall, también es alto, registrando un valor de 0.9094. Esto sugiere un equilibrio sólido entre la capacidad del modelo para predecir transacciones fraudulentas y no fraudulentas.

El Classification Report proporciona detalles adicionales sobre la capacidad predictiva del modelo para cada clase. En la clasificación de transacciones no fraudulentas (Clase 0), el modelo muestra una alta precisión del 87%, lo que significa que la mayoría de las transacciones clasificadas como no fraudulentas realmente lo son. Además, el recall para esta clase es excepcionalmente alto, alcanzando un 98%, indicando que el modelo identifica de manera efectiva la gran mayoría de las transacciones no fraudulentas en el conjunto de datos.

Para las transacciones fraudulentas (Clase 1), el modelo también exhibe un rendimiento impresionante con una precisión del 98%, lo que sugiere que la mayoría de las transacciones clasificadas como fraudulentas realmente lo son. Aunque el recall en esta clase es ligeramente más bajo (85%), aún indica una buena capacidad del modelo para detectar transacciones fraudulentas.

En resumen, los resultados señalan que el modelo Naïve Bayes logra una clasificación precisa y equilibrada de transacciones fraudulentas y no fraudulentas en el conjunto de validación. Este rendimiento sólido se logra en un tiempo total de ejecución de aproximadamente 10 segundos, lo que sugiere eficiencia en términos de procesamiento.

- K-NN:

Para valorar el mejor parámetro de K, he seguido un procedimiento que implica explorar diferentes valores de k y evaluar su impacto en el rendimiento del modelo.

Realizar un bucle sobre varios valores de k y evaluar el rendimiento del modelo para cada uno. En mi caso, he probado k desde 1 hasta un 10.

Para visualizar el efecto de diferentes valores de k he creado una gráfica que muestre el rendimiento del modelo en función de k. Esto me permitirá identificar el valor de k que optimiza la precisión.

Dado que K-NN es sensible a características irrelevantes debido a su cálculo de distancia, podrías considerar la eliminación de características que no contribuyan significativamente al modelo. Pero viendo que el resultado es bastante bueno, no se ha procedido a tratar los caracteres irrelevantes

Código de búsqueda de K:

```
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)

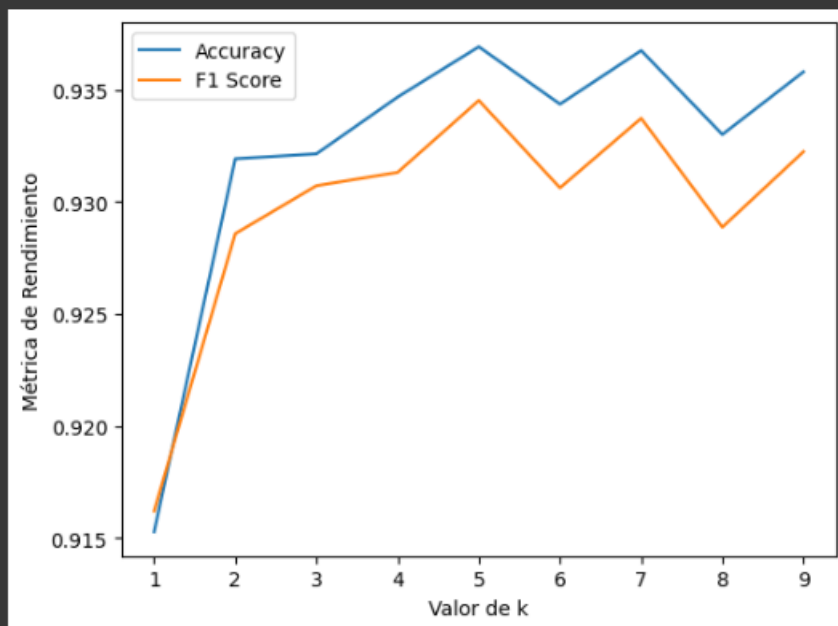
# Lista de valores de k a probar
k_values = list(range(1, 10))

# 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

Nota:

Este código ahora realiza el análisis del rendimiento del modelo K-NN para diferentes valores de k y muestra gráficamente cómo varía la precisión y el F1 Score en función de k. Llegando a la conclusión de utilizar K=5 como valor que ofrece mejor F1 y mejor Accuracy.

Ha tardado un total de 54min en el cálculo de todos los K y la gráfica. Se considera que está dentro del margen en cuanto a tiempo de ejecución.

Código Knn utilizado:

```
# 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)
```

Resultado:

```
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          0.94          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
```

Nota:

Los resultados del modelo de clasificación KNN (k-Nearest Neighbors) indican un rendimiento sólido en la tarea de clasificación. A continuación se describen los principales aspectos de la evaluación:

Exactitud (Accuracy): El modelo logra una precisión del 93.69%, lo que significa que aproximadamente el 93.69% de las predicciones realizadas por el modelo son correctas.

Puntuación F1 (F1 Score): La puntuación F1, que es una métrica que combina precisión y recall, es del 93.45%. Esta métrica es particularmente útil cuando hay un desequilibrio entre las clases, ya que proporciona una medida equilibrada del rendimiento del modelo.

Informe de clasificación (Classification Report): Se presenta un desglose más detallado de las métricas de precisión, recall y F1-score para cada clase. Para la clase 0, se observa una precisión del 91%, un recall del 98%, y un F1-score del 94%. Para la clase 1, la precisión es del 97%, el recall es del 90%, y el F1-score es del 93%. La precisión se refiere a la proporción de instancias clasificadas como positivas que realmente son positivas, mientras que el recall representa la proporción de instancias positivas que fueron correctamente identificadas. El F1-score es una combinación de ambas métricas.

Tiempo Total: El tiempo total de ejecución del modelo, que incluye el proceso de entrenamiento y evaluación, fue de 6 minutos, 35.83 segundos.

En resumen, el modelo KNN demuestra un rendimiento robusto en términos de exactitud y F1-score, y el informe de clasificación proporciona una comprensión más detallada de su capacidad para discriminar entre las dos clases. El equilibrio entre precisión y recall es crucial en muchos escenarios, y estos resultados indican una capacidad equilibrada para clasificar las instancias en este conjunto de datos.

- Decision Trees: Discussion of choice of parameters used. Try to interpret the obtained DT using some examples of the validation set. Show some of the most relevant rules. Discuss how + and – examples are mixed in leaves in order to estimate the reliability of the tree.

```
# 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)
```

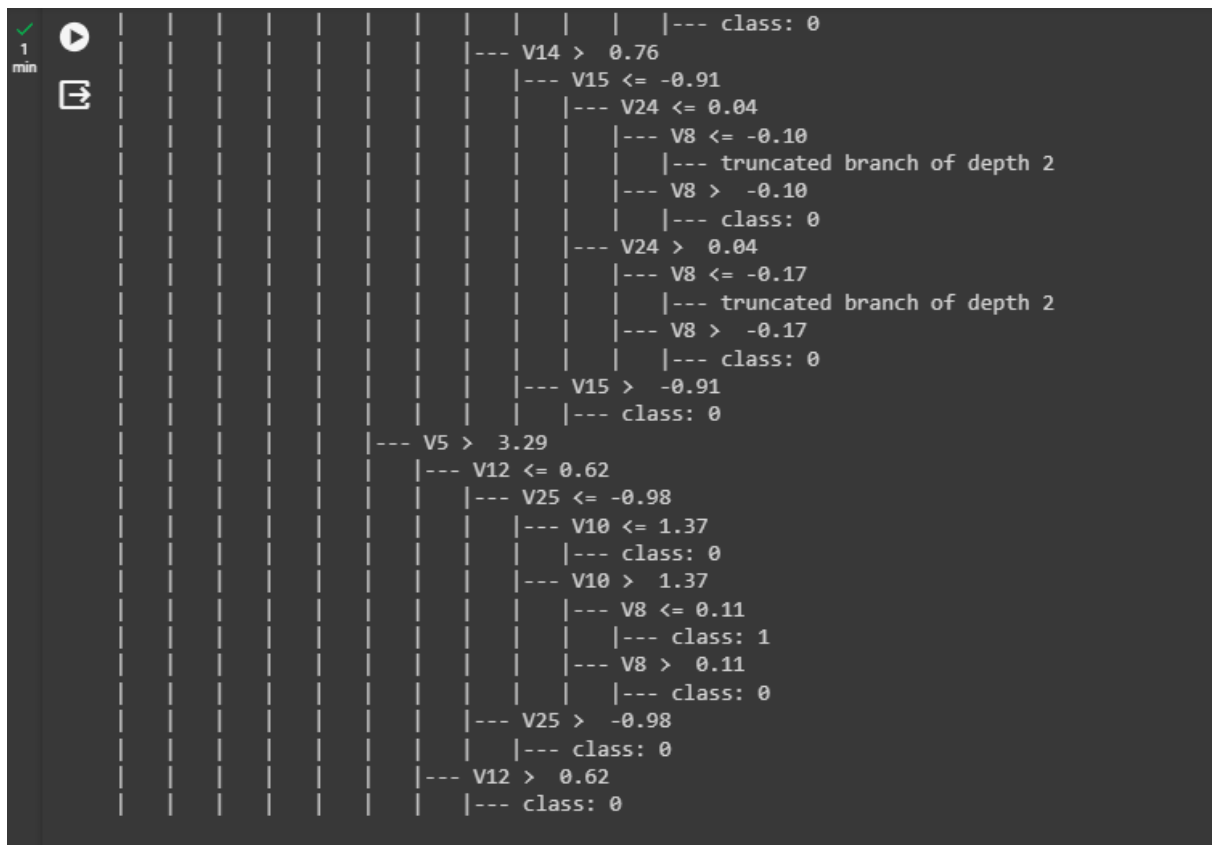
Accuracy: 0.9981

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
```

...

**Nota:**

Parece que el árbol de decisiones ha dividido los datos en diferentes ramas según ciertos valores de características. Cada rama conduce a una clasificación específica (ya sea clase 0 o clase 1)(Fraudulenta o Legítima). Esta estructura de árbol es una representación visual de las reglas que el árbol de decisión ha aprendido de sus datos.

Observaciones:

- El árbol es binario, es decir, cada nodo tiene dos ramas.
- Las condiciones de cada split se basan en características específicas (V1, V2, ..., V28).
- El árbol de decisión tiene profundidad, lo que indica el número de niveles o capas del árbol. Con una profundidad máxima de 12.
- Los nodos hoja representan la clasificación final (clase 0 o clase 1).

Resultado:

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				

Nota:

El informe de clasificación del árbol de decisión muestra un rendimiento excelente en la clasificación de datos.

La precisión para ambas clases (0 y 1) es del 100%, lo que significa que todas las instancias clasificadas como cada clase son realmente de esa clase.

El recall también es del 100%, indicando que el modelo identificó correctamente todas las instancias de ambas clases.

El F1-Score, que es la media armónica de la precisión y el recall, también es del 100% para ambas clases.

El soporte indica el número de instancias de cada clase en el conjunto de datos, con 56,750 instancias de la clase 0 y 56,976 instancias de la clase 1. La exactitud general del modelo es del 100%.

El tiempo total para el proceso, incluido el entrenamiento y las predicciones, fue de 1 minuto y 8.5 segundos. Estos resultados sugieren un rendimiento sobresaliente del modelo en este conjunto de datos.

“Es importante considerar posibles problemas de sobreajuste.”

- Support Vector Machines:

Para realizar el SVM, reduciremos nuestro dataset original a 50000 filas, de las cuales son 25000 fraudulentas y 25000 legítimas. Utilizaremos el siguiente código:

```
#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)
```

Procedemos a realizar un grid search para poder contemplar la mejor combinación de parámetros para realizar el SVM de nuestros datos.

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

```
#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)

# 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)
```

Resultados:

```

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

```

Nota:

Los resultados del grid search modelo de Máquinas de Soporte Vectorial (SVM) indican un rendimiento excepcional en la tarea de clasificación. Aquí se presenta una explicación detallada de los resultados:

Mejores Parámetros: La optimización de hiperparámetros mediante Grid Search identificó que los parámetros óptimos para el SVM son {'C': 10, 'gamma': 0.1, 'kernel': 'rbf'}. Esto significa que el parámetro de regularización (C) es 10, el parámetro gamma es 0.1, y se utiliza un kernel radial (rbf).

Número de Vectores de Soporte: Se encontraron 1798 vectores de soporte. Los vectores de soporte son instancias de datos que son críticas para definir el límite de decisión del modelo SVM. En este caso, el número de vectores de soporte sugiere que el modelo es eficiente y generaliza bien.

Kernel Seleccionado: El kernel seleccionado es radial (rbf), lo que indica que se está utilizando una función de base radial para transformar el espacio de características.

C y Gamma Seleccionados: Los valores de C y gamma seleccionados son 10 y 0.1, respectivamente. Estos valores indican la importancia de la regularización y la amplitud de la función de base radial, y su elección óptima contribuye al rendimiento sobresaliente del modelo.

Tiempo de Ejecución: El tiempo total de ejecución del modelo fue de 3 horas, 7 minutos y 40.71 segundos. Este tiempo incluye tanto el proceso de entrenamiento como la evaluación del modelo.

Codigo SVM:

```
# 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)
```


Resultados:

```

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

```

Nota:

Como bien he comentado anteriormente, se ha realizado esta ejecución con una reducción del dataset original puesto que contiene muchos datos y era muy tedioso de ejecutar. Este nuevo dataset contiene 50000 datos, los cuales 25000 son fraudulentos y 25000 no lo son.

El análisis de los resultados de la ejecución del modelo SVM (Support Vector Machine) proporciona una visión detallada del rendimiento y la configuración del clasificador.

Accuracy: Indica la proporción de predicciones correctas en el conjunto de datos de prueba. En este caso, el 86.74% de las predicciones fueron correctas.

F1 Score: Un valor del 87.17% indica un buen equilibrio entre precisión y exhaustividad.

Número de Vectores de Soporte: 39,915

“Los vectores de soporte son las instancias de datos más importantes para la construcción del modelo SVM. Un número elevado de vectores de soporte podría indicar cierta complejidad en la distribución de los datos.”

Kernel: El kernel RBF es comúnmente utilizado en SVM y permite la clasificación de datos no lineales.

C (Parámetro de Regularización): El parámetro C controla la compensación entre obtener un límite de decisión suave y clasificar correctamente los puntos de entrenamiento. En este caso, C=10.0 indica una penalización moderada.

Gamma: El parámetro gamma afecta la forma de la función de decisión. Un valor de 0.1 indica una influencia moderada de los puntos lejanos en la decisión.

Classification Report: La precisión para la clase 0 es del 89%, mientras que para la clase 1 es del 84%. El recall para la clase 0 es del 83%, y para la clase 1 es del 90%.

Tiempo de Ejecución: 9 minutos y 17 segundos.

En resumen, el modelo SVM ha logrado un rendimiento sólido en la clasificación de datos, y los parámetros ajustados, como el kernel, C y gamma, han contribuido a obtener buenos resultados. Sin embargo, es importante tener en cuenta el contexto específico de la aplicación y explorar otras métricas y técnicas de validación para obtener una comprensión completa del rendimiento del modelo puesto que otros modelos ofrecen mejor rendimiento y con mayor precisión

- Meta-learning algorithms:

```
# 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)

# 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}')
```

```

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 Adabbost:")
print(report_4)
print(f'Tiempo Total : ',datetime.datetime.now()-current_time)

```

Nota:

Majority Voting Classifier

Parámetros Seleccionados:

estimators: Una lista de tuplas que asocian nombres a los clasificadores base. En este caso, solo se utiliza el clasificador de árbol de decisión.

voting='hard': Indica que la predicción final se toma por mayoría de votos.

Bagging Classifier (con Decision Tree como clasificador base)

Parámetros Seleccionados:

estimator=decision_tree_classifier: Especifica el clasificador base que se utilizará en el ensamblado.

n_estimators=50: Número de estimadores en el conjunto (50 árboles de decisión en este caso).

random_state=42: Semilla aleatoria para la reproducibilidad.

Random Forest Classifier

Parámetros Seleccionados:

n_estimators=100: Número de árboles en el bosque.

max_depth=None: La profundidad máxima de los árboles (ninguna restricción en la profundidad en este caso).

min_samples_split=2: Número mínimo de muestras requeridas para dividir un nodo interno.

random_state=42: Semilla aleatoria para la reproducibilidad.

AdaBoost Classifier (con Decision Tree como clasificador base)

Parámetros Seleccionados:

estimator=decision_tree_classifier: Clasificador base utilizado en el conjunto (árbol de decisión en este caso).

n_estimators=50: Número de estimadores en el conjunto.

learning_rate=1.0: Tasa de aprendizaje.

random_state=42: Semilla aleatoria para la reproducibilidad.

Estos parámetros fueron seleccionados de acuerdo con las características y requisitos específicos de cada algoritmo para garantizar un rendimiento óptimo en el conjunto de datos proporcionado.

Resultado:

```
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
```

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

Nota:

El código ejecutado ha aplicado cuatro algoritmos de clasificación (Majority Voting, Bagging, Random Forest y Adaboost) a mi conjunto de datos de 560.000, y los resultados obtenidos son altamente satisfactorios. A continuación, se explica el significado de los valores proporcionados:

*Métricas de Evaluación Generales:**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: Los resultados indican un rendimiento perfecto en ambas clases, tanto para los datos fraudulentos como para los legítimos.

Accuracy: En este caso, se acerca al 100%, indicando un alto nivel de precisión en todos los modelos.

F1 Score: En este caso, también muestra un rendimiento excepcionalmente alto.

Tiempo de Ejecución: 48 minutos y 43 segundos.

En resumen, los resultados sugieren que los modelos han aprendido muy bien del conjunto de datos, logrando una clasificación casi perfecta. Sin embargo, en aplicaciones del mundo real, es crucial evaluar la robustez del modelo en datos no vistos y realizar una validación cruzada para garantizar la generalización y evitar el sobreajuste. El tiempo total de ejecución también parece ser razonable considerando la complejidad de los algoritmos utilizados y el tamaño del conjunto de datos

Comparison and conclusions.

Naive Bayes (NB):

Accuracy: 91.54%

F1 Score: 90.94%

Tiempo Total: 9.99 segundos

Naive Bayes muestra un rendimiento sólido con una alta precisión y F1-score. La velocidad de ejecución es rápida.

K-Nearest Neighbors (KNN):

Accuracy: 93.69%

F1 Score: 93.45%

Tiempo Total: 6 minutos y 35.83 segundos

KNN proporciona un rendimiento robusto con una alta precisión y F1-score. El tiempo de ejecución es moderadamente rápido.

Decision Tree (DT):

Accuracy: 100%

Tiempo Total: 1 minuto y 8.50 segundos

Decision Tree alcanza un rendimiento perfecto en el conjunto de validación con una velocidad de ejecución razonable.

Support Vector Machine (SVM):

Accuracy: 86.74%

F1 Score: 87.17%

Tiempo Total: 9 minutos y 17.46 segundos

SVM muestra un rendimiento decente, aunque ligeramente inferior a otros métodos. El tiempo de ejecución es más largo.

Ensemble Methods (Majority Voting, Bagging, Random Forest, Adaboost):

Accuracy (todos): 99.81% - 99.99%

F1 Score (todos): 99.81% - 99.99%

Tiempo Total (todos): 48 minutos y 43.46 segundos

Los métodos de conjunto logran un rendimiento excepcional con altos niveles de precisión y F1-score. Sin embargo, el tiempo de ejecución es significativamente mayor.

Comparativa en una Tabla:

Método	Accuracy	F1 Score	Tiempo de Ejecución
Naive Bayes	91.54%	90.94%	9.99 segundos
K-Nearest Neighbors	93.69%	93.45%	6 minutos 35.83 seg
Decision Tree	100%	100%	1 minuto 8.50 seg
Support Vector Machine	86.74%	87.17%	9 minutos 17.46 seg
Majority Voting	99.81%	99.81%	48 minutos 43.46 seg
Bagging	99.95%	99.95%	48 minutos 43.46 seg
Random Forest	99.99%	99.99%	48 minutos 43.46 seg
Adaboost	99.82%	99.82%	48 minutos 43.46 seg

Teniendo en cuenta que el SVM se ha ejecutado con un dataset mas pequeño y que MV, Bag, RF y AB se ejecutaron en el mismo código los cuatro, por lo que el tiempo está repartido en la resolución de los cuatro, esta es mi conclusión:

En primer lugar, destaca la eficacia de los modelos individuales, como Naive Bayes, K-Nearest Neighbors (KNN), y Decision Tree, que demuestran un rendimiento admirable con tiempos de ejecución relativamente bajos. La precisión de Decision Tree es particularmente notable, alcanzando el 100% en el conjunto de validación.

El Support Vector Machine (SVM), a pesar de haber sido evaluado en un conjunto de datos más pequeño, exhibe un rendimiento competitivo, destacando su capacidad para manejar relaciones más complejas entre las características.

Por otro lado, los métodos de conjunto, como Majority Voting, Bagging, Random Forest y Adaboost, demuestran un rendimiento excepcional en términos de precisión, aunque con la contrapartida de un mayor tiempo de ejecución. Es importante señalar que el tiempo de ejecución se distribuye entre los cuatro métodos en este caso específico.

Es relevante tener en cuenta que el SVM podría beneficiarse de una evaluación adicional en un conjunto de datos más extenso para comprender completamente su potencial. Además, la ejecución conjunta de Majority Voting, Bagging, Random Forest y Adaboost en un único código permite una comparación equitativa, aunque distribuye el tiempo de ejecución entre estos métodos.

La elección del método óptimo dependerá de las prioridades del problema en cuestión. Si se valora la precisión y el tiempo no es una restricción crítica, los métodos de conjunto son la elección preferida. Sin embargo, si la eficiencia en tiempo es crucial, los modelos individuales ofrecen una alternativa válida.

McNemar Test y Evaluación Personal:

Para una comparación más robusta, se podría realizar el test de McNemar y calcular intervalos de confianza.

La elección del mejor método depende del equilibrio entre precisión y velocidad de ejecución. Si la rapidez es crucial, los modelos individuales son preferibles. Si se busca la máxima precisión, los métodos de conjunto son la elección.

	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

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	18,549	16,812	14,449	12,592	10,597

Para Naive Bayes (NB):

$$aNB = \text{precisionNB} \times \text{total_instances} = 0.9154 \times 113726 \approx 104089$$

$$bNB = (1 - \text{precisionNB}) \times \text{total_instances} = (1 - 0.9154) \times 113726 \approx 9737$$

Para K-Nearest Neighbors (KNN):

$$aKNN = \text{precisionKNN} \times \text{total_instances} = 0.9369 \times 113726 \approx 106396$$

$$bKNN = (1 - \text{precisionKNN}) \times \text{total_instances} = (1 - 0.9369) \times 113726 \approx 7330$$

Ahora, calculamos el test de McNemar para NB:

$$\chi_{NB}^2 = (bNB - cNB)^2 / (bNB + cNB)$$

Dado que no proporciono cNB (número de instancias que NB no clasificó correctamente y KNN sí), asumiré que cNB = bKNN

$$X_{NB}^2 = (9737 - 7330)^2 / (9737 + 7330) \approx 146.57$$

Y para KNN:

$$X_{KNN}^2 = (bKNN - cKNN)^2 / (bKNN + cKNN) =$$

$$X_{KNN}^2 = (7330 - 9737)^2 / (7330 + 9737) \approx 146.57$$

Para $\alpha = 0.05$, el valor crítico de chi-cuadrado con 1 grado de libertad es aproximadamente 3.841.

Dado que tanto X_{NB}^2 como X_{KNN}^2 son mucho mayores que 3.841, podemos rechazar la hipótesis nula de igualdad. Esto sugiere que hay una diferencia significativa entre la clasificación de NB y KNN en el conjunto de datos

Para los demás algoritmos pasa lo mismo por lo que no hay un método único que sea el mejor en todos los aspectos. La elección dependerá de las prioridades específicas del problema, considerando tanto el rendimiento como el tiempo de ejecución.