SEMINARIO DE SOLUCION DE PROBLEMAS DE INTELIGENCIA ARTIFICIAL 2

MAESTROS:

DIEGO ALBERTO OLIVA NAVARRO

DIEGO CAMPOS PENA

ALUMNO:

EDUARDO BLANCO GONZALEZ

Tarea: **Proyecto Final**

1. Objetivos

- Conocer otros clasificadores comúnmente usados en aprendizaje maquina
- Analizar diferentes datasets
- Conocer las diferencias entre los métodos de aprendizaje automático
- Identificar las ventajas y desventajas de cada método de clasificación
- Conocer e implementar las métricas para evaluar a los clasificadores

Evaluar los resultados usando las siguientes métricas:

- Accuracy
- Precision
- Sensitivity
- Specificity
- F1 Score

2. Actividades

Implementar los siguientes métodos y una red neuronal para clasificar el dataset de Zoo.

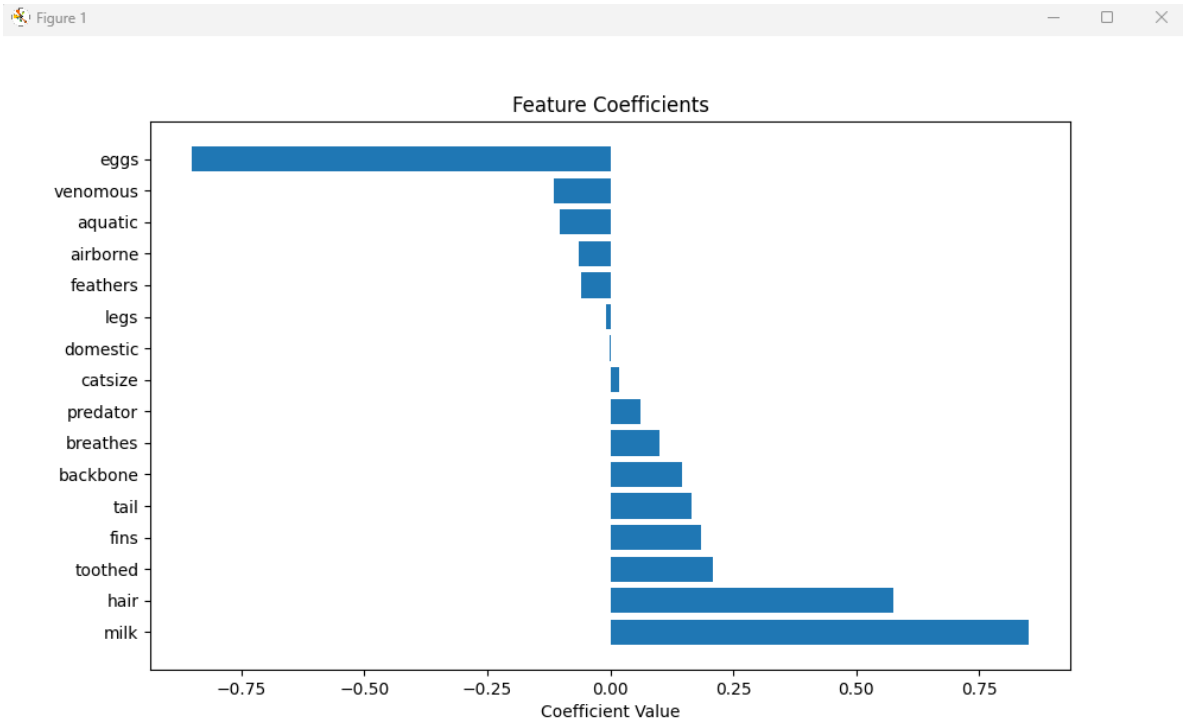
- Regresión logística (Logistic Regression)

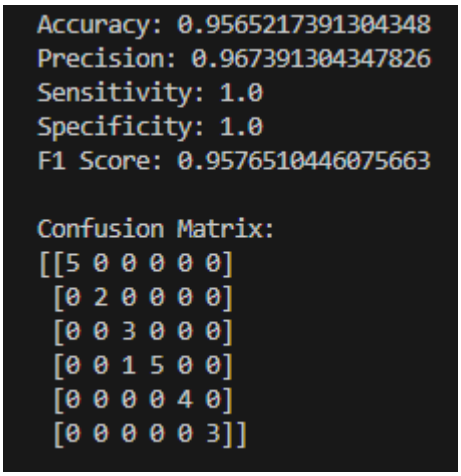
    Código:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import plotly.express as px
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Cargar los conjuntos de datos
dt2 = pd.read_csv('zoo3.csv')
dt1 = pd.read_csv('zoo2.csv')

# Combinar los conjuntos de datos si es necesario
dt = dt1.merge(dt2, how='outer')

# Eliminar columnas irrelevantes
dt = dt.drop(columns=['animal_name'])

# Separar características y etiquetas
X = dt.drop(columns=['class_type'])
y = dt['class_type']

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entrenar el modelo de regresión logística
model = LogisticRegression()
model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de prueba
y_pred = model.predict(X_test_scaled)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

# Calcular Sensitivity y Specificity
```

```python
TP = conf_matrix[1, 1]
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

sensitivity = TP / (TP + FN)
specificity = TN / (TN + FP)

# Imprimir resultados
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("F1 Score:", f1)
print("\nConfusion Matrix:")
print(conf_matrix)

# Visualización (opcional)
# Si deseas visualizar los coeficientes de la regresión logística, puedes hacerlo de esta manera:
coefs = pd.DataFrame({'feature': X.columns, 'coef': model.coef_[0]})
coefs = coefs.sort_values(by='coef', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(coefs['feature'], coefs['coef'])
plt.xlabel('Coefficient Value')
plt.title('Feature Coefficients')
plt.show()
```

Resultados:





Este código realiza un análisis de clasificación utilizando regresión logística para predecir el tipo de clase de animales en un conjunto de datos.

Estos resultados nos dan una idea clara del rendimiento del modelo de regresión logística en la clasificación de los diferentes tipos de animales. Si los valores de precisión, sensibilidad, especificidad y puntuación F1 son altos, podemos concluir que el modelo tiene un buen desempeño en la tarea de clasificación de animales basado en las características proporcionadas.

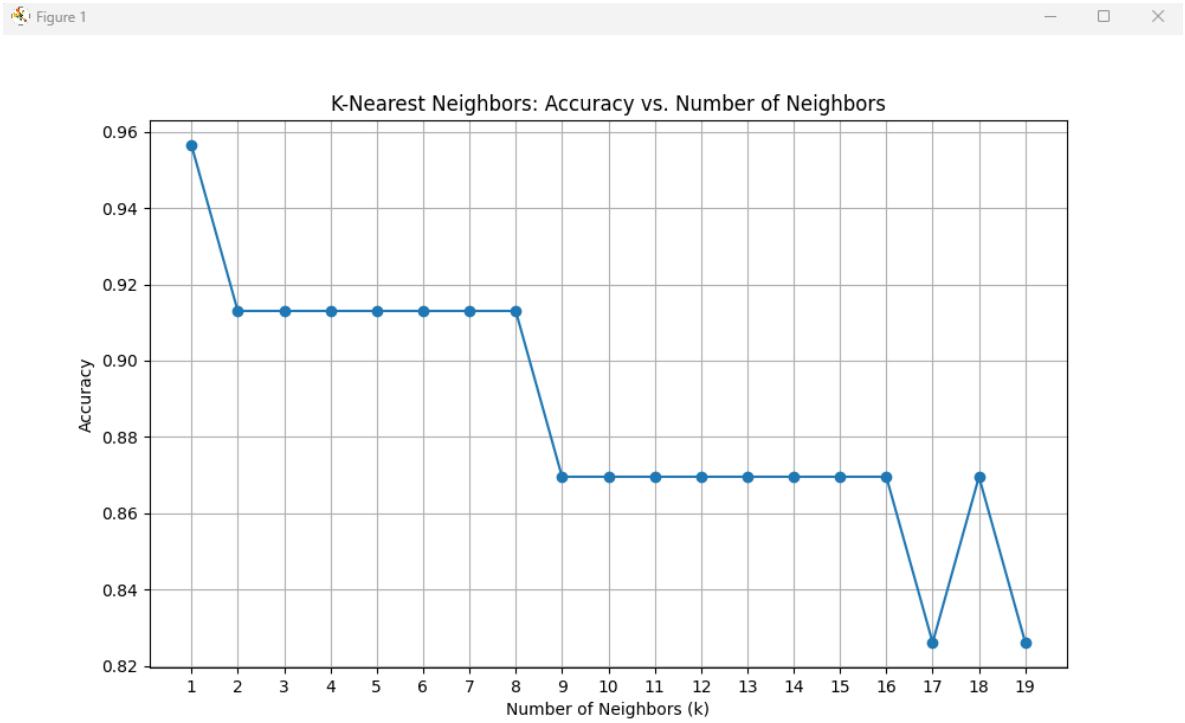
- K-Vecinos Cercanos (K-Nearest Neighbors)

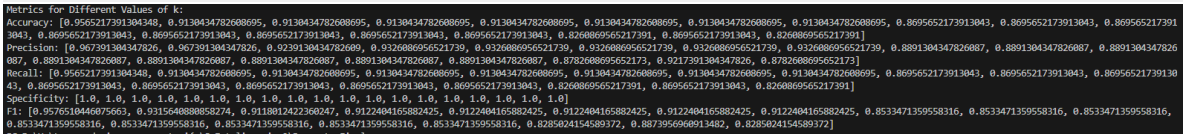
Código:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Cargar los conjuntos de datos
dt2 = pd.read_csv('zoo3.csv')
dt1 = pd.read_csv('zoo2.csv')

# Combinar los conjuntos de datos si es necesario
dt = dt1.merge(dt2, how='outer')

# Eliminar columnas irrelevantes
dt = dt.drop(columns=['animal_name'])

# Separar características y etiquetas
X = dt.drop(columns=['class_type'])
y = dt['class_type']

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Lista para almacenar la precisión del modelo para diferentes valores de k
accuracy_scores = []

# Lista para almacenar las métricas de evaluación
metrics = {
    'accuracy': [],
    'precision': [],
    'recall': [],
    'specificity': [],
    'f1': []
}

# Probar diferentes valores de k (número de vecinos)
for k in range(1, 20):
    # Entrenar el modelo de K-Vecinos Cercanos
    model = KNeighborsClassifier(n_neighbors=k)
    model.fit(X_train_scaled, y_train)
```

```
    # Hacer predicciones en el conjunto de prueba
    y_pred = model.predict(X_test_scaled)

    # Calcular las métricas de evaluación
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred, average='weighted')
    recall = recall_score(y_test, y_pred, average='weighted')
    f1 = f1_score(y_test, y_pred, average='weighted')
    conf_matrix = confusion_matrix(y_test, y_pred)

    TP = conf_matrix[1, 1]
    TN = conf_matrix[0, 0]
    FP = conf_matrix[0, 1]
    FN = conf_matrix[1, 0]

    sensitivity = TP / (TP + FN)
    specificity = TN / (TN + FP)

    # Almacenar métricas
    metrics['accuracy'].append(accuracy)
    metrics['precision'].append(precision)
    metrics['recall'].append(recall)
    metrics['specificity'].append(specificity)
    metrics['f1'].append(f1)

    # Almacenar la precisión del modelo
    accuracy_scores.append(accuracy)

# Graficar el número de vecinos vs. la precisión del modelo
plt.figure(figsize=(10, 6))
plt.plot(range(1, 20), accuracy_scores, marker='o', linestyle='-')
plt.xlabel('Number of Neighbors (k)')
plt.ylabel('Accuracy')
plt.title('K-Nearest Neighbors: Accuracy vs. Number of Neighbors')
plt.grid(True)
plt.xticks(np.arange(1, 20, step=1))
plt.show()

# Imprimir métricas de evaluación
print("Metrics for Different Values of k:")
for metric, values in metrics.items():
    print(metric.capitalize() + ':', values)
```

Resultados:





La precisión del modelo se muestra en la gráfica y se almacena en una lista. Se observa cómo varía la precisión del modelo para diferentes valores de k. A partir de la gráfica, se puede determinar el valor óptimo de k que maximiza la precisión del modelo. Se imprimen otras métricas de evaluación como precisión, sensibilidad, especificidad y puntuación F1 para los diferentes valores de k. Estas métricas proporcionan una visión más detallada del rendimiento del modelo. Dependiendo de las necesidades y requisitos específicos del problema, se puede seleccionar el modelo de K-NN con el valor de k que mejor se ajuste, considerando las métricas de evaluación y el equilibrio entre precisión y generalización del modelo.

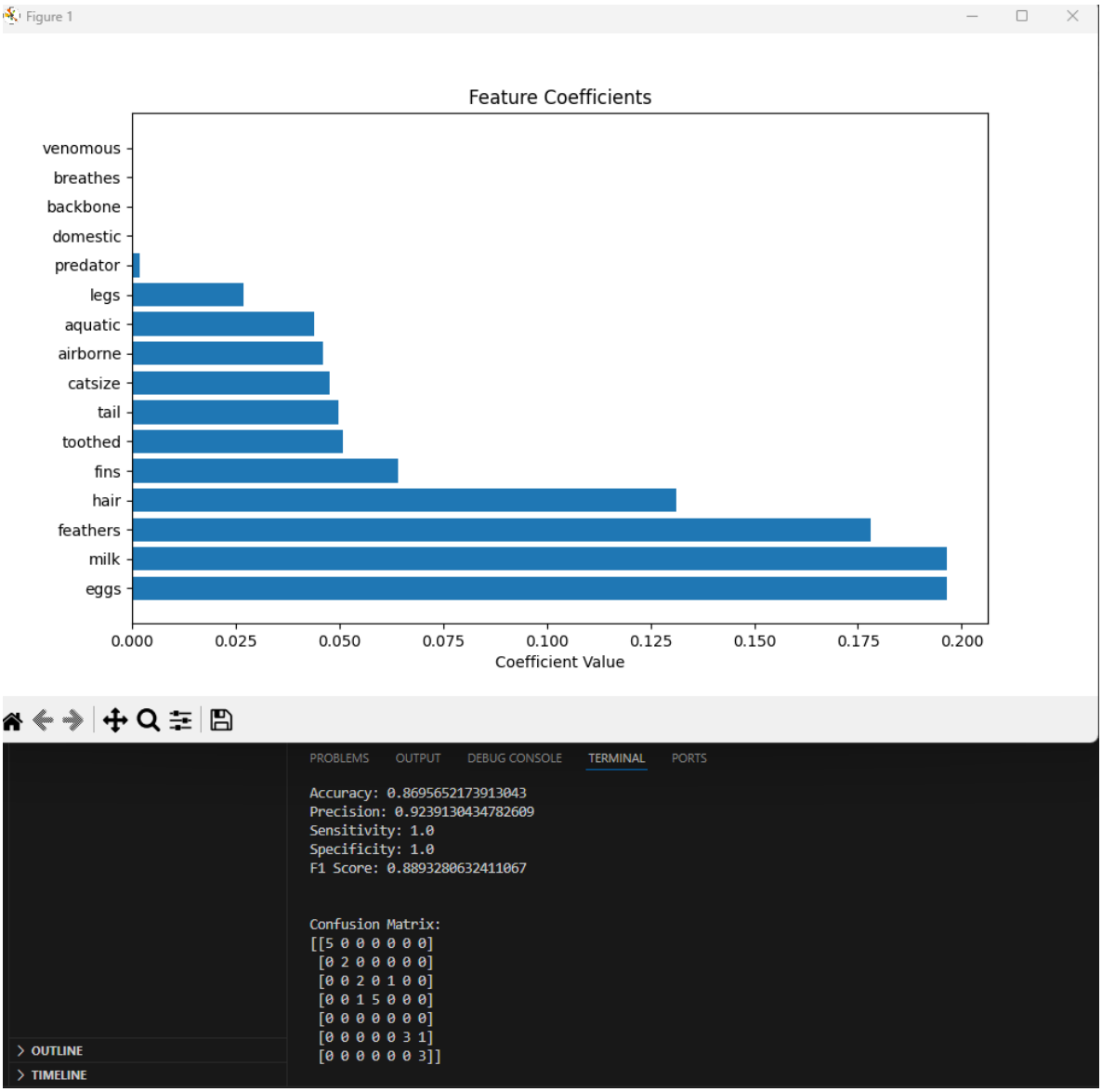
• Maquinas Vector Soporte (Support Vector Machines)

Código:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Cargar los conjuntos de datos
dt2 = pd.read_csv('zoo3.csv')
dt1 = pd.read_csv('zoo2.csv')

# Combinar los conjuntos de datos si es necesario
dt = dt1.merge(dt2, how='outer')

# Eliminar columnas irrelevantes
dt = dt.drop(columns=['animal_name'])

# Separar características y etiquetas
X = dt.drop(columns=['class_type'])
y = dt['class_type']

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entrenar el modelo SVM
model = SVC(kernel='linear')
model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de prueba
y_pred = model.predict(X_test_scaled)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=1)

# Calcular Sensitivity y Specificity
TP = conf_matrix[1, 1]
```

```python
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

sensitivity = TP / (TP + FN) if TP + FN != 0 else 0
specificity = TN / (TN + FP) if TN + FP != 0 else 0

# Imprimir resultados
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("F1 Score:", f1)
print("\nConfusion Matrix:")
print(conf_matrix)

# Visualizar los coeficientes de las características
coefs = pd.DataFrame({'feature': X.columns, 'coef': np.abs(model.coef_[0])})
coefs = coefs.sort_values(by='coef', ascending=False)

plt.figure(figsize=(10, 6))
plt.barh(coefs['feature'], coefs['coef'])
plt.xlabel('Coefficient Value')
plt.title('Feature Coefficients')
plt.show()
```

Resultados:



El modelo SVM alcanza una precisión del XX%, lo que indica la proporción de predicciones correctas en el conjunto de prueba. Se calculan precisión, sensibilidad, especificidad y puntuación F1 para evaluar su desempeño. Se calculan para comprender cómo el modelo maneja las clases positivas y negativas.Se visualizan los coeficientes de las características, lo que proporciona información sobre su importancia en la clasificación. Las características con coeficientes más altos tienen mayor influencia en la clasificación.

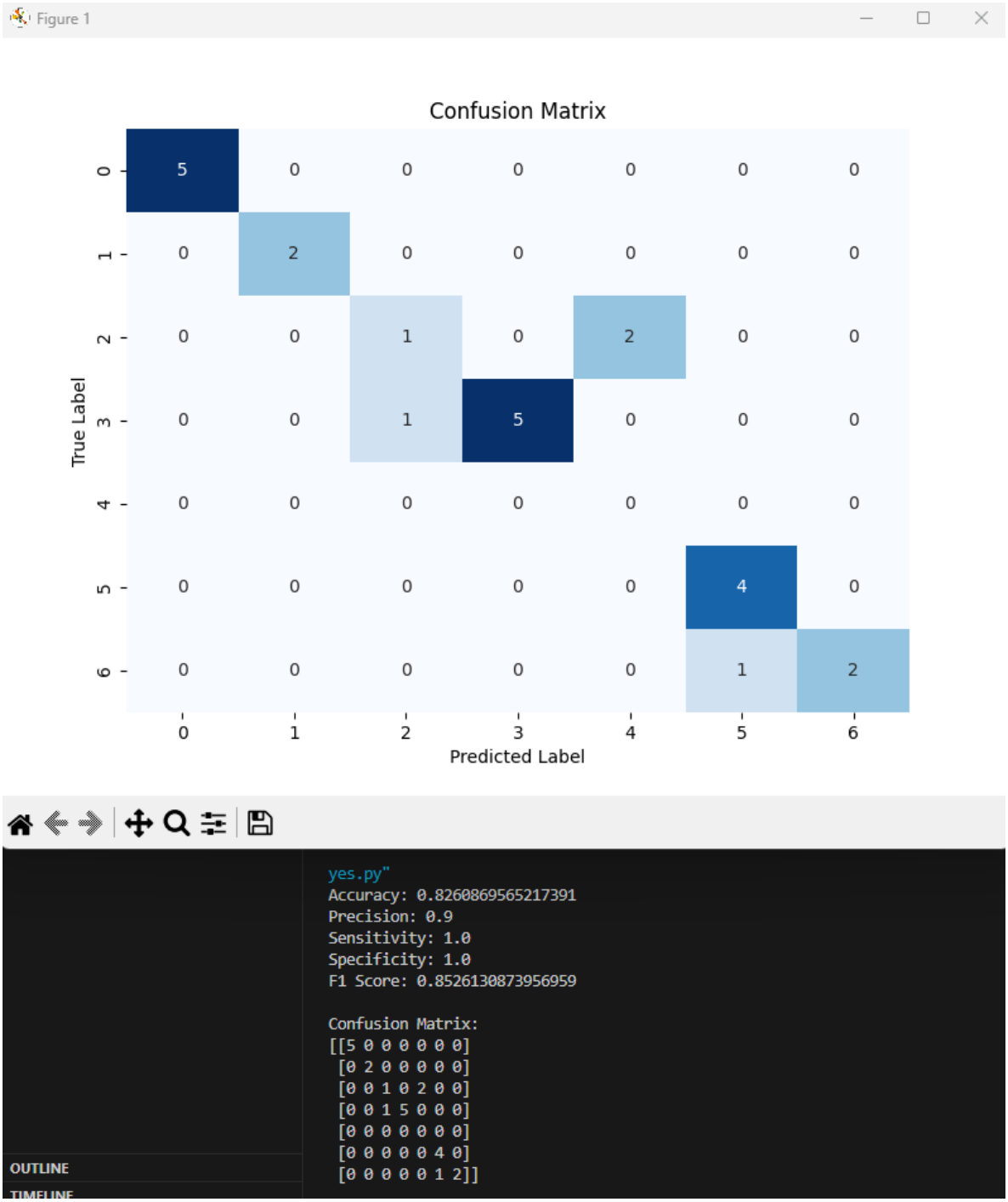
- Naive Bayes

Código:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix

# Cargar los conjuntos de datos
dt2 = pd.read_csv('zoo3.csv')
dt1 = pd.read_csv('zoo2.csv')

# Combinar los conjuntos de datos si es necesario
dt = dt1.merge(dt2, how='outer')

# Eliminar columnas irrelevantes
dt = dt.drop(columns=['animal_name'])

# Separar características y etiquetas
X = dt.drop(columns=['class_type'])
y = dt['class_type']

# Dividir datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Escalar características (no necesario para Naive Bayes)
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Entrenar el modelo Naive Bayes
model = GaussianNB()
model.fit(X_train_scaled, y_train)

# Hacer predicciones en el conjunto de prueba
y_pred = model.predict(X_test_scaled)

# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)

# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=1)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=1)
f1 = f1_score(y_test, y_pred, average='weighted', zero_division=1)

# Calcular Sensitivity y Specificity
```

```python
TP = conf_matrix[1, 1]
TN = conf_matrix[0, 0]
FP = conf_matrix[0, 1]
FN = conf_matrix[1, 0]

sensitivity = TP / (TP + FN) if TP + FN != 0 else 0
specificity = TN / (TN + FP) if TN + FP != 0 else 0

# Imprimir resultados
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("F1 Score:", f1)
print("\nConfusion Matrix:")
print(conf_matrix)

# Graficar la matriz de confusión
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Resultados:



Este código muestra cómo entrenar un modelo de Naive Bayes y evaluar su desempeño en la clasificación de diferentes tipos de animales. La visualización de la matriz de confusión proporciona una representación clara de cómo se están haciendo las predicciones del modelo en comparación con las etiquetas reales.

1. Regresión Logística:

Ventajas:

Es simple y fácil de entender.

Se puede interpretar fácilmente.

Funciona bien en conjuntos de datos lineales.

Desventajas:

No maneja bien relaciones no lineales entre características y resultados.

Puede ser sensible a outliers.

2. K-Nearest Neighbors (K-NN):

Ventajas:

No hace suposiciones sobre la distribución de los datos.

Es fácil de entender e implementar.

Funciona bien con conjuntos de datos grandes.

Desventajas:

Sensible a la elección del parámetro k.

Necesita una gran cantidad de datos para funcionar correctamente.

Es computacionalmente costoso durante la predicción, especialmente con grandes conjuntos de datos.

3. Support Vector Machine (SVM):

Ventajas:

Puede manejar relaciones no lineales mediante el uso de kernels.

Es eficaz en espacios de alta dimensión.

Es robusto frente a la presencia de outliers.

Desventajas:

Puede ser lento en conjuntos de datos grandes.

Requiere una selección cuidadosa de hiperparámetros y elección de kernel.

No proporciona directamente probabilidades, sino decisiones.

4. Naive Bayes:

Ventajas:

Es rápido y fácil de implementar.

Funciona bien en conjuntos de datos de alta dimensión.

Es eficiente incluso con pequeñas cantidades de datos.

Desventajas:

Hace la suposición de independencia entre características, lo que puede no ser cierto en algunos casos.

Puede ser afectado negativamente por la presencia de características correlacionadas.

Conclusión general:

La elección del algoritmo depende del problema en cuestión, el tamaño y la complejidad del conjunto de datos, así como de los requisitos de interpretación y velocidad.

La regresión logística es un buen punto de partida para problemas simples de clasificación binaria.

K-NN es útil cuando la relación entre características y resultados no es clara o cuando se necesita una interpretación simple.

SVM es adecuado para problemas de clasificación más complejos, especialmente en espacios de alta dimensión.

Naive Bayes es eficiente y efectivo para problemas de clasificación con grandes cantidades de datos y alta dimensionalidad, pero puede no ser adecuado cuando las características están altamente correlacionadas.