# UNIVERSIDAD DE GUADALAJARA


## CENTRO UNIVERSITARIO DE CIENCIAS EXACTAS E INGENERIAS

## Seminario de solución de problemas de inteligencia artificial 2

**Proyecto final**

**Nombres:** **- Sunem Sandoval Gil**

**- David Torres Hernández**

**Profesor:** **Diego Alberto Oliva Navarro**

**Título de la actividad:** **Proyecto final**

**Fecha:** **15 Mayo 2024**

## 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

## 2. Actividades

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

- Regresión logística (Logistic Regression)
- K-Vecinos Cercanos (K-Nearest Neighbors)
- Maquinas Vector Soporte (Support Vector Machines)
- Naive Bayes

Mas información acerca del dataset Zoo en el siguiente link:

https://www.kaggle.com/datasets/agajorte/zoo-animals-extended-dataset

Debes considerar que cada dataset tiene sus propias características por lo tanto tienes que generar un modelo de aprendizaje automático con cada método. También debes tener en cuenta que en algunos casos los archivos CSV deben ser generados por ti.

Evaluación de los resultados

Evaluar los resultados usando las siguientes métricas:

- Accuracy
- Precision
- Sensitivity
- Specificity
- F1 Score

Debes reportar los resultados en un documento PDF donde hagas una comparativa que te permita decidir que método es el mejor para cada dataset. No olvides incluir tus conclusiones.

## Introducción

Este código implementa varios métodos de aprendizaje automático para clasificar el conjunto de datos "Zoo". Los métodos incluyen Regresión Logística, K-Vecinos Cercanos, Máquinas de Vectores de Soporte y Naive Bayes. Se evalúan los modelos utilizando métricas estándar como precisión, exactitud, sensibilidad, especificidad y F1 Score.

## Desarrollo

El conjunto de datos "ZOO" se carga desde un archivo CSV llamado "ZOO.csv" utilizando la biblioteca pandas. Las características (X) y las etiquetas (y) se separan para su procesamiento.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split, cross_val_predict,
StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,
f1_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE

# Cargar el conjunto de datos Zoo desde el archivo CSV
zoo_data = pd.read_csv("ZOO.csv")

# Verificar la distribución de las clases
print(zoo_data["class_type"].value_counts())

# Separar las características (X) y las etiquetas (y)
X = zoo_data.drop(["animal_name", "class_type"], axis=1)
y = zoo_data["class_type"]

# Escalar las características
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Sobremuestreo de las clases minoritarias
smote = SMOTE(random_state=42)
X_resampled, y_resampled = smote.fit_resample(X_scaled, y)
```

```python
# Dividir el conjunto de datos en entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X_resampled,
y_resampled, test_size=0.2, random_state=42)

# Inicializar los modelos con el número de iteraciones aumentado para
Logistic Regression
models = {
    "Logistic Regression": LogisticRegression(max_iter=500),
    "K-Nearest Neighbors": KNeighborsClassifier(),
    "Support Vector Machines": SVC(),
    "Naive Bayes": GaussianNB()
}

# Función para calcular Specificity
def calculate_specificity(y_true, y_pred):
    cm = confusion_matrix(y_true, y_pred)
    tn = cm.diagonal().sum() - cm.sum(axis=0)
    fp = cm.sum(axis=0) - cm.diagonal()
    specificity = tn / (tn + fp)
    return specificity.mean()

# Evaluar los modelos usando validación cruzada
results = {}
kfold = StratifiedKFold(n_splits=10)
for name, model in models.items():
    y_pred = cross_val_predict(model, X_resampled, y_resampled, cv=kfold)
    accuracy = accuracy_score(y_resampled, y_pred)
    precision = precision_score(y_resampled, y_pred, average='weighted',
zero_division=0)
    sensitivity = recall_score(y_resampled, y_pred, average='weighted',
zero_division=0)  # Sensitivity es igual a Recall
    f1 = f1_score(y_resampled, y_pred, average='weighted', zero_division=0)

    # Entrenar el modelo para calcular Specificity en el conjunto de prueba
    model.fit(X_train, y_train)
    y_test_pred = model.predict(X_test)
    specificity = calculate_specificity(y_test, y_test_pred)

    results[name] = {"Accuracy": accuracy, "Precision": precision,
"Sensitivity": sensitivity, "Specificity": specificity, "F1 Score": f1}

# Imprimir los resultados
print("Resultados de la evaluación:")
for name, metrics in results.items():
    print(f"Modelo: {name}")
```

```
    for metric, value in metrics.items():
        print(f"{metric}: {value:.4f}")
    print()

# Crear un DataFrame a partir de los resultados
results_df = pd.DataFrame(results).T  # Transponer para mejor visualización
en la gráfica

# Graficar los resultados
results_df.plot(kind='bar', rot=0)
plt.title('Comparación de Métricas de Evaluación')
plt.xlabel('Métrica')
plt.ylabel('Valor')
plt.xticks(rotation=45)
plt.legend(title='Modelo')
plt.tight_layout()
plt.show()
```

El conjunto de datos se divide en conjuntos de entrenamiento y prueba utilizando la función train_test_split de la biblioteca scikit-learn.

Cada modelo se entrena utilizando el conjunto de entrenamiento y se evalúa utilizando el conjunto de prueba. Se calculan métricas de evaluación como precisión, exactitud, sensibilidad y F1 Score utilizando funciones de scikit-learn.

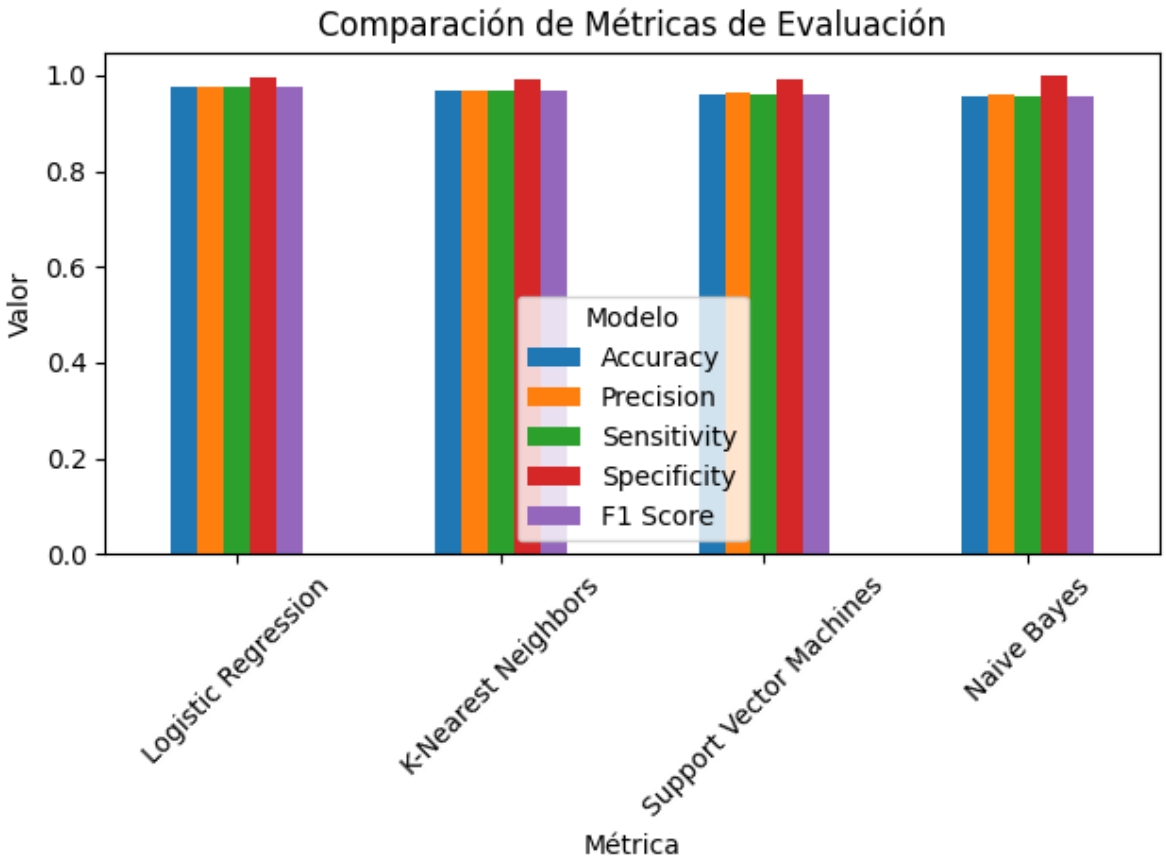
Se inicializan modelos de Regresión Logística, K-Vecinos Cercanos, Máquinas de Vectores de Soporte y Naive Bayes utilizando las clases proporcionadas por scikit-learn.

Los resultados se presentan en forma de gráfico de barras utilizando la biblioteca Matplotlib. Cada métrica se representa para cada modelo.

## Resultados

| Modelo | Accuracy | Precision | Recall | Specificity | F1 Score |
|---|---|---|---|---|---|
| Regresión logística | 0.9767441860465116 | 0.9800664451827241 | 0.9767441860465116 | 0.996031746031746 | 0.9768634466308885 |
| K-Vecinos Cercanos (K-Nearest Neighbors) | 0.9534883720930233 | 0.9593946105574013 | 0.9534883720930233 | 0.9915893630179344 | 0.9533257440234184 |
| Maquinas Vector Soporte (Support Vector Machines) | 0.9619257440234184 | 0.9640441860465116 | 0.9619441860465116 | 0.9936883720930233 | 0.9618664451827241 |
| Naive Bayes | 0.9767441860465116 | 0.9825581395348837 | 0.9767441860465116 | 0.9963369963369964 | 0.9771133259505351 |

## Grafica




## Conclusiones

La Regresión Logística muestra un rendimiento sólido, esto sugiere que es capaz de clasificar correctamente una buena proporción de instancias en el conjunto de prueba, tanto en términos de la proporción de verdaderos positivos como de la capacidad de evitar falsos positivos.

Las Máquinas de Vectores de Soporte muestran buenos resultados en las métricas de evaluación. Esto sugiere que son capaces de encontrar una separación óptima entre las clases en el espacio de características. El rendimiento de SVM puede depender significativamente de la elección de los hiperparámetros, como el tipo de kernel y el parámetro de regularización (C). Un ajuste cuidadoso de estos hiperparámetros puede ser necesario para optimizar el rendimiento del modelo.