PROYECTO FINAL SEMINARIO IA

SECCIÓN D05

LOPEZ HERNANDEZ, MIGUEL ANGEL
26-11-2023

Regresión Logística:

- **Descripción**: Es un modelo lineal que se utiliza para la clasificación. Calcula la probabilidad de que un punto de datos pertenezca a una clase particular.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
# Cargar el dataset
data = pd.read_csv('zoo.data', header=None)
# Asignar nombres a las columnas
columns = ['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic',
'predator', 'toothed',
           'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic',
'catsize', 'class_type']
data.columns = columns
# Separar los datos en características (X) y etiquetas (y)
X = data.drop(['animal_name', 'class_type'], axis=1)
y = data['class_type']
# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Inicializar y entrenar el modelo de regresión logística
logreg = LogisticRegression(max_iter=10000)  # Ajusta el número máximo de iteraciones si
es necesario
logreg.fit(X_train, y_train)
# Realizar predicciones en el conjunto de prueba
y_pred = logreg.predict(X_test)
# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted')
# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
# Métricas adicionales (sensitivity, specificity)
true_negatives = conf_matrix[0, 0]
false_negatives = conf_matrix[1, 0]
true_positives = conf_matrix[1, 1]
false_positives = conf_matrix[0, 1]

sensitivity = true_positives / (true_positives + false_negatives)
specificity = true_negatives / (true_negatives + false_positives)

print("Accuracy:", accuracy)
print("Precision:", precision)
print("Sensitivity:", sensitivity)
print("Specificity:", specificity)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

- **Desempeño en este dataset**:

```
Accuracy: 0.9523809523809523
Precision: 0.9206349206349207
Sensitivity: 1.0
Specificity: 1.0
F1 Score: 0.9333333333333333
Confusion Matrix:
[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  1]]
```

- **Observaciones**: La regresión logística es una buena opción para problemas de clasificación binaria o multiclase con datos linealmente separables. Sin embargo, su rendimiento podría verse afectado si los datos no siguen una distribución lineal.

K-Vecinos Cercanos (KNN):

- **Descripción**: Es un algoritmo simple de aprendizaje supervisado que se basa en la similitud entre instancias. Clasifica los datos según la mayoría de votos de sus vecinos más cercanos.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
from sklearn.preprocessing import StandardScaler
# Cargar el dataset
data = pd.read_csv('zoo.data', header=None)
# Asignar nombres a las columnas
columns = ['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic',
'predator', 'toothed',
           'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic',
'catsize', 'class_type']
data.columns = columns
# Separar los datos en características (X) y etiquetas (y)
X = data.drop(['animal_name', 'class_type'], axis=1)
y = data['class_type']
# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Normalizar los datos
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
# Inicializar y entrenar el modelo de K-Vecinos Cercanos
k = 5  # Número de vecinos
knn = KNeighborsClassifier(n_neighbors=k)
knn.fit(X_train_normalized, y_train)
# Realizar predicciones en el conjunto de prueba
y_pred = knn.predict(X_test_normalized)
# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted')
# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
# Imprimir las métricas
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

- **Desempeño en este dataset**:

```
Accuracy: 0.9523809523809523
Precision: 0.9206349206349207
Recall: 0.9523809523809523
F1 Score: 0.9333333333333333
Confusion Matrix:
[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  1]]
```

- **Observaciones**: KNN es flexible y puede adaptarse a diferentes distribuciones de datos. Sin embargo, puede volverse computacionalmente costoso con grandes conjuntos de datos y no es óptimo para características con diferentes escalas.

Máquinas de Vectores de Soporte (SVM):

- **Descripción**: SVM busca encontrar el hiperplano que mejor separa los puntos de diferentes clases en el espacio de características.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
confusion_matrix
from sklearn.preprocessing import StandardScaler
# Cargar el dataset
data = pd.read_csv('zoo.data', header=None)
# Asignar nombres a las columnas
columns = ['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic',
'predator', 'toothed',
           'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic',
'catsize', 'class_type']
data.columns = columns
# Separar los datos en características (X) y etiquetas (y)
X = data.drop(['animal_name', 'class_type'], axis=1)
y = data['class_type']
# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Normalizar los datos
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
# Inicializar y entrenar el modelo de Máquinas de Vectores de Soporte (SVM)
svm = SVC(kernel='rbf', C=1.0, gamma='scale')  # Puedes ajustar los parámetros según
necesites
svm.fit(X_train_normalized, y_train)
# Realizar predicciones en el conjunto de prueba
y_pred = svm.predict(X_test_normalized)
# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted')
# Calcular la matriz de confusion
conf_matrix = confusion_matrix(y_test, y_pred)
# Imprimir las métricas
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

- **Desempeño en este dataset**:

```
Accuracy: 0.8571428571428571
Precision: 0.888888888888889
Recall: 0.8571428571428571
F1 Score: 0.8380952380952381
Confusion Matrix:
[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  1  2]
 [ 0  0  0  0  0  1]]
```

- **Observaciones**: SVM funciona bien en espacios de características de alta dimensión y es eficaz en problemas de clasificación no lineales mediante el uso de kernels. Puede funcionar mejor con conjuntos de datos más pequeños.

Naive Bayes:

- **Descripción**: Un modelo probabilístico que se basa en el teorema de Bayes asumiendo independencia entre las características.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
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
# Cargar el dataset
data = pd.read_csv('zoo.data', header=None)
# Asignar nombres a las columnas
columns = ['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic',
'predator', 'toothed',
           'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic',
'catsize', 'class_type']
data.columns = columns
# Separar los datos en características (X) y etiquetas (y)
X = data.drop(['animal_name', 'class_type'], axis=1)
y = data['class_type']
# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Normalizar los datos (no es necesario para Naive Bayes, pero puede mejorar el
rendimiento en algunos casos)
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
# Inicializar y entrenar el modelo de Naive Bayes (Gaussian Naive Bayes)
naive_bayes = GaussianNB()
naive_bayes.fit(X_train, y_train)
# Realizar predicciones en el conjunto de prueba
y_pred = naive_bayes.predict(X_test)
# Calcular las métricas
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='weighted', zero_division=0)
recall = recall_score(y_test, y_pred, average='weighted', zero_division=0)
f1 = f1_score(y_test, y_pred, average='weighted')
# Calcular la matriz de confusión
conf_matrix = confusion_matrix(y_test, y_pred)
# Imprimir las métricas
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

- **Desempeño en este dataset**:

```
Accuracy: 0.9523809523809523
Precision: 0.9206349206349207
Recall: 0.9523809523809523
F1 Score: 0.9333333333333333
Confusion Matrix:
[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  1]]
```

- **Observaciones**: Naive Bayes es simple, rápido y eficiente en conjuntos de datos pequeños. Funciona bien incluso si la suposición de independencia no se cumple, pero puede ser sensible a la presencia de características irrelevantes.

Red Neuronal:

- **Descripción**: Un modelo de aprendizaje profundo que puede aprender patrones complejos en datos a través de múltiples capas ocultas.

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout
# Cargar el dataset
data = pd.read_csv('zoo.data', header=None)
# Asignar nombres a las columnas
columns = ['animal_name', 'hair', 'feathers', 'eggs', 'milk', 'airborne', 'aquatic',
'predator', 'toothed',
           'backbone', 'breathes', 'venomous', 'fins', 'legs', 'tail', 'domestic',
'catsize', 'class_type']
data.columns = columns
# Separar los datos en características (X) y etiquetas (y)
X = data.drop(['animal_name', 'class_type'], axis=1)
y = data['class_type']
# Dividir los datos en conjunto de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Normalizar los datos
scaler = StandardScaler()
X_train_normalized = scaler.fit_transform(X_train)
X_test_normalized = scaler.transform(X_test)
# Convertir las etiquetas a un formato adecuado para redes neuronales (one-hot encoding)
num_classes = len(np.unique(y))
y_train = tf.keras.utils.to_categorical(y_train - 1, num_classes=num_classes)
y_test = tf.keras.utils.to_categorical(y_test - 1, num_classes=num_classes)
# Definir la arquitectura de la red neuronal
model = Sequential()
model.add(Dense(128, activation='relu', input_shape=(X_train.shape[1],)))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes, activation='softmax'))
# Compilar el modelo
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
# Entrenar la red neuronal
batch_size = 32
epochs = 100
model.fit(X_train_normalized, y_train, batch_size=batch_size, epochs=epochs, verbose=1,
validation_data=(X_test_normalized, y_test))
# Evaluar el modelo en el conjunto de prueba
y_pred_probs = model.predict(X_test_normalized)
y_pred = np.argmax(y_pred_probs, axis=1) + 1
# Calcular las métricas
accuracy = accuracy_score(np.argmax(y_test, axis=1) + 1, y_pred)
precision = precision_score(np.argmax(y_test, axis=1) + 1, y_pred, average='weighted',
zero_division=0)
recall = recall_score(np.argmax(y_test, axis=1) + 1, y_pred, average='weighted',
zero_division=0)
f1 = f1_score(np.argmax(y_test, axis=1) + 1, y_pred, average='weighted')
# Calcular la matriz de confusión
conf_matrix = confusion_matrix(np.argmax(y_test, axis=1) + 1, y_pred)
# Imprimir las métricas
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1 Score:", f1)
print("Confusion Matrix:")
print(conf_matrix)
```

- **Desempeño en este dataset**:

```
Epoch 100/100
3/3 [==============================] - 0s 24ms/step - loss: 0.1026 - accuracy: 0.9625 - val_loss: 0.1359 - val_accuracy: 0.9524
1/1 [==============================] - 0s 105ms/step
Accuracy: 0.9523809523809523
Precision: 0.9206349206349207
Recall: 0.9523809523809523
F1 Score: 0.9333333333333333
Confusion Matrix:
[[12  0  0  0  0  0]
 [ 0  2  0  0  0  0]
 [ 0  0  0  1  0  0]
 [ 0  0  0  2  0  0]
 [ 0  0  0  0  3  0]
 [ 0  0  0  0  0  1]]
```

- **Observaciones**: Las redes neuronales son altamente adaptables y pueden aprender patrones complicados en datos. Sin embargo, pueden requerir más datos y tiempo de entrenamiento, y pueden ser susceptibles al sobreajuste.

CONCLUSION:

En este análisis de varios modelos de clasificación aplicados al conjunto de datos dado, se observa que la Regresión Logística y las Máquinas de Vectores de Soporte (SVM) destacan como opciones prometedoras para este dataset específico.

La Regresión Logística muestra un desempeño sólido en términos de precisión, exactitud y F1 Score. Es una opción sólida para conjuntos de datos donde las clases pueden ser linealmente separables.

Las Máquinas de Vectores de Soporte (SVM) también muestran un buen rendimiento, especialmente en la clasificación de datos no lineales. SVM puede manejar eficazmente características de alta dimensionalidad y puede adaptarse bien a diferentes espacios de características mediante el uso de kernels.

Ambos modelos han demostrado ser efectivos en la clasificación de este conjunto de datos en particular, mostrando buenas métricas en las pruebas realizadas.