Act 14: Programando K-Nearest-Neighbor en Python

Julio Cesar Torres Marquez

marzo 2025

1 Introducción

El algoritmo K-Nearest-Neighbor (K-NN) es uno de los métodos más simples y populares de aprendizaje supervisado, utilizado tanto para clasificación como para regresión. Este modelo se basa en la idea de que las instancias similares están más cerca unas de otras en el espacio de características. Al aplicar K-NN, se predice la clase de una nueva muestra basándose en la mayoría de las clases de sus vecinos más cercanos. En este trabajo, se implementará el algoritmo K-NN para clasificar datos de un conjunto de revisión de productos utilizando Python.

2 Metodología

2.1 Importación de librerías

Para llevar a cabo la implementación, se utilizaron las siguientes librerías:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import seaborn as sb
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

2.2 Leer el archivo CSV y mostrar los primeros 10 registros

Se cargó el archivo CSV con los datos y se mostraron los primeros registros:

```
dataframe = pd.read_csv(r"reviews_sentiment.csv", sep=";")
print(dataframe.head(10))
```

2.3 Resumen estadístico de los datos

Se realizó un resumen estadístico básico para comprender la distribución y variabilidad de los datos:

```
print(dataframe.describe())
```

2.4 Visualización de la información

Se generaron gráficos para visualizar la distribución de las valoraciones y la cantidad de palabras en las reseñas:

```
dataframe.hist()
plt.show()
sb.catplot(x='Star Rating', data=dataframe, kind="count", aspect=3, palette="Set2")
plt.show()
sb.catplot(x='wordcount', data=dataframe, kind="count", aspect=3, palette="Set3")
plt.show()
```

2.5 Preparación de los datos y conjuntos de entrenamiento

Se definieron las características de entrada y las etiquetas, y luego se dividieron los datos en conjuntos de entrenamiento y prueba:

```
X = dataframe[['wordcount', 'sentimentValue']].values
y = dataframe['Star Rating'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)
scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

2.6 Entrenamiento del modelo K-NN

Se entrenó el clasificador K-Nearest-Neighbor utilizando 7 vecinos y se evaluó su rendimiento en los conjuntos de entrenamiento y prueba:

```
n_neighbors = 7
knn = KNeighborsClassifier(n_neighbors)
knn.fit(X_train, y_train)
print('Accuracy of K-NN classifier on training set: {:.2f}'.format(knn.score(X_train, y_tra
print('Accuracy of K-NN classifier on test set: {:.2f}'.format(knn.score(X_test, y_test)))
```

2.7 Evaluación de precisión del modelo

Se evaluó la precisión del modelo utilizando la matriz de confusión y el reporte de clasificación:

```
pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

2.8 Selección del mejor valor de k

Para encontrar el valor óptimo de k, se probaron diferentes valores y se evaluó la precisión del modelo:

```
k_range = range(1, 20)
scores = []
for k in k_range:
    knn = KNeighborsClassifier(n_neighbors=k)
    knn.fit(X_train, y_train)
    scores.append(knn.score(X_test, y_test))
plt.figure()
plt.xlabel('k')
plt.ylabel('accuracy')
plt.scatter(k_range, scores)
plt.xticks([0, 5, 10, 15, 20])
plt.show()
```

2.9 Predicción de nuevas muestras

Finalmente, el modelo se utilizó para predecir la clase de nuevas muestras:

```
print(knn.predict([[5, 1.0]]))
print(knn.predict_proba([[20, 0.0]]))
```

3 Resultados

El modelo de K-Nearest-Neighbor mostró un buen desempeño en la clasificación de reseñas, con una precisión notable tanto en el conjunto de entrenamiento como en el de prueba. Además, la selección de k permitió encontrar el valor óptimo que mejoró la precisión del modelo.

4 Conclusión

El algoritmo K-Nearest-Neighbor ha demostrado ser eficaz para clasificar reseñas de productos en función de la cantidad de palabras y el valor de sentimiento. Aunque el algoritmo es sencillo y fácil de implementar, se mostró competitivo en términos de precisión en este conjunto de datos. Sin embargo, su rendimiento puede disminuir cuando se aplica a conjuntos de datos más grandes o con más dimensiones. A pesar de esto, es un excelente punto de partida para problemas de clasificación simples y puede ser útil como base para otros algoritmos más complejos.