Inteligencia Artificial
Act 14: Programando K-Nearest-Neighbor en Python

Estudiante: José Luis Calderón Galarza - 2132939
Docente: Luis Ángel Gutiérrez Rodríguez

1 Introducción

El algoritmo K-Nearest Neighbors (K-NN) es un método de aprendizaje supervisado utilizado para la clasificación y regresión. En clasificación, asigna una clase a una nueva observación basándose en la mayoría de los vecinos más cercanos en el espacio de características. Es un algoritmo no paramétrico y simple de implementar, ampliamente utilizado en sistemas de recomendación, reconocimiento de patrones y minería de datos.

2 Metodología

En esta actividad, se implementó el algoritmo K-NN para clasificar reseñas de productos en función del número de palabras y un valor de sentimiento asociado. Se siguieron los siguientes pasos:

1. **Importación de librerías y carga de datos:** Se utilizaron bibliotecas como pandas, numpy, matplotlib, seaborn y sklearn.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
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. **Exploración y visualización de datos:** Se generaron histogramas y gráficos de barras para entender la distribución de los datos.

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

3. **Preprocesamiento de datos:** Se normalizaron las variables y se dividieron 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)
```

4. **Entrenamiento del modelo:** Se utilizó K-NN con k=7 para entrenar el modelo.

```
n_neighbors = 7
knn = KNeighborsClassifier(n_neighbors)
knn.fit(X_train, y_train)
```

5. **Evaluación del modelo:** Se calcularon 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))
```

6. **Visualización de fronteras de decisión:** Se graficó la separación de clases en el espacio de características.

```
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold, edgecolor='k', s=20)
plt.show()
```

7. **Análisis de precisión con diferentes valores de k:** Se analizó la precisión variando k entre 1 y 20.

```
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.scatter(k_range, scores)
plt.xlabel('k')
plt.ylabel('accuracy')
plt.show()
```

3 Resultados

3.1 Histograma de características de entrada

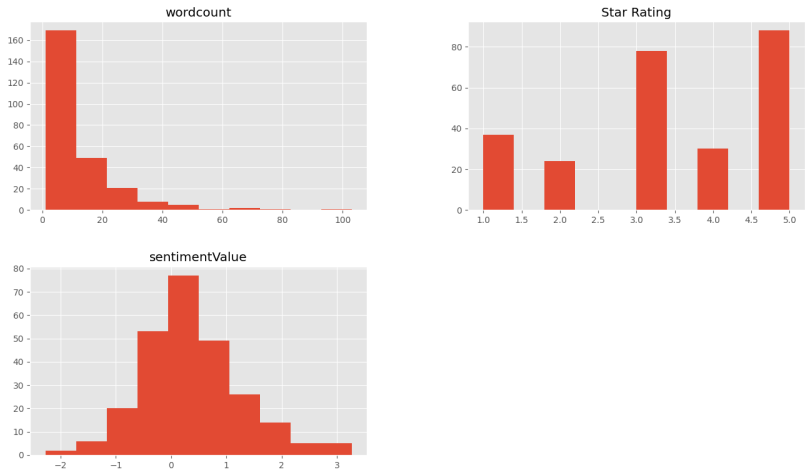

Figure 1: Histograma de las características de entrada.

3.2 Distribución de calificaciones

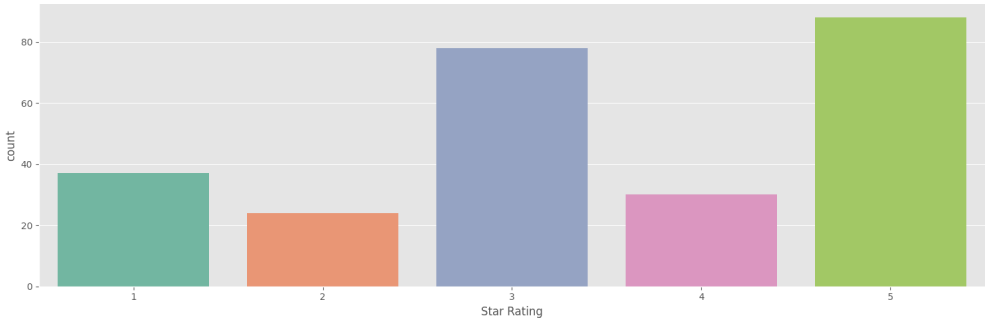

Figure 2: Gráfico de barras de la distribución de calificaciones.

3.3 Frontera de decisión del modelo K-NN

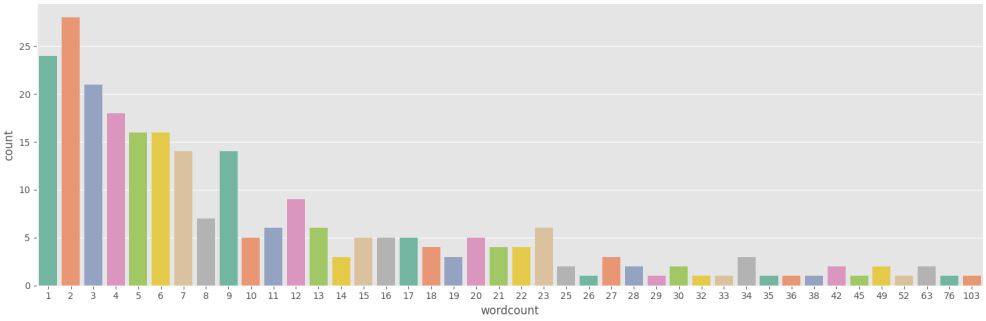

Figure 3: Frontera de decisión del clasificador K-NN.

3.4 Precisión del modelo para distintos valores de k

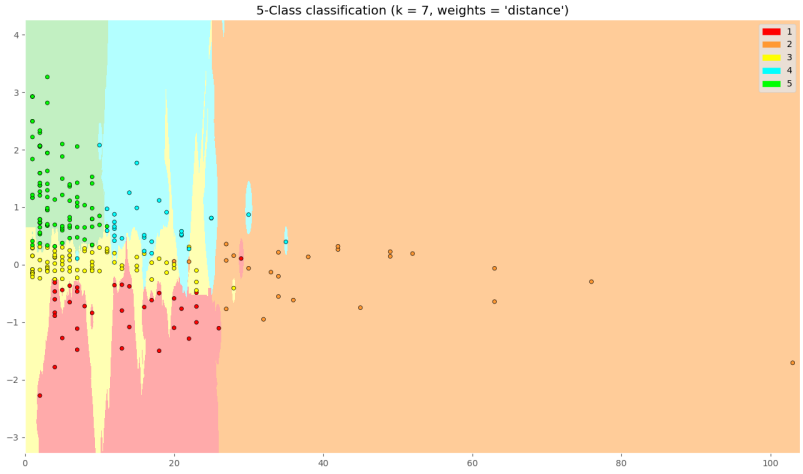

Figure 4: Precisión del modelo en función del número de vecinos k.

4 Conclusión

Se implementó el modelo K-Nearest Neighbors para clasificar reseñas de productos en función del número de palabras y su valor de sentimiento. Se evaluó su precisión utilizando distintas métricas y se analizaron las fronteras de decisión para diferentes valores de k. Se concluye que el modelo ofrece un buen desempeño en la clasificación y que la elección del valor de k es crucial para obtener resultados óptimos.