UANL

UNIVERSIDAD AUTÓNOMA DE NUEVO LEÓN

FCFM

FACULTAD DE CIENCIAS FÍSICO MATEMÁTICAS

Universidad Autónoma de Nuevo León

Licenciatura en Ciencias Computacionales

A14: Programando K-Nearest-Neighbor en Python

Materia: Inteligencia Artificial

Maestro: Luis Ángel Gutiérrez Rodríguez

Rebeca Jaramillo Camarillo

Matrícula: 2132988

Grupo: 031

Fecha: 30 de marzo de 2025

1. Introducción

El algoritmo de k vecinos más cercanos, también conocido como KNN o k-NN, es un clasificador de aprendizaje supervisado no paramétrico, que utiliza la proximidad para hacer clasificaciones o predicciones sobre la agrupación de un punto de datos individual.
Si bien se puede usar para problemas de regresión o clasificación, generalmente se usa como un algoritmo de clasificación, partiendo de la suposición de que se pueden encontrar puntos similares cerca uno del otro.

Para los problemas de clasificación, se asigna una etiqueta de clase sobre la base de un voto mayoritario, es decir, se utiliza la etiqueta que se representa con más frecuencia alrededor de un punto de datos determinado. Si bien esto técnicamente se considera "voto por mayoría", el término "voto por mayoría"se usa más comúnmente en la literatura. La distinción entre estas terminologías es que "voto mayoritario"técnicamente requiere una mayoría superior al 50 %, lo que funciona principalmente cuando solo hay dos categorías. Cuando tiene varias clases, por ejemplo, cuatro categorías, no necesita necesariamente el 50 % de los votos para llegar a una conclusión sobre una clase; puede asignar una etiqueta de clase con un voto superior al 25 %.

2. Metodología

A continuación, se muestran los pasos para resolver el ejercicio:

Importar librerías necesarias

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
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
from sklearn.metrics import confusion_matrix
```

2.1. Leer el archivo csv y cargarlo como un dataset de Pandas

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

Visualización del archivo

```
dataframe.hist()
plt.show()

print(dataframe.groupby('Star Rating').size())

sb.countplot(x='Star Rating', data=dataframe)
plt.show()
sb.countplot(x='wordcount', data=dataframe)
plt.show()
```

2.2. Preparar el dataset

```
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)
```

2.3. Crear el Modelo

```
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_train)))
print('Accuracy of K-NN classifier on test set: {:.2f}'
     .format(knn.score(X_test, y_test)))

## Precision del modelo
pred = knn.predict(X_test)
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))
```

2.4. Gráfica de la Clasificación Obtenida

```
h = .02  # step size in the mesh

# Create color maps
cmap_light = ListedColormap(['#FFAAAA', '#ffcc99', '#ffffb3','#b3ffff','#
    c2f0c2'])
cmap_bold = ListedColormap(['#FF0000', '#ff9933','#FFFF00','#00ffff','#00
    FF00'])

# we create an instance of Neighbours Classifier and fit the data.
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)

# Plot the decision boundary. For that, we will assign a color to each
# point in the mesh [x_min, x_max]x[y_min, y_max].
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                     np.arange(y_min, y_max, h))
Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])

# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

# Plot also the training points
plt.scatter(X[:, 0], X[:, 1], c=y, cmap=cmap_bold,
                edgecolor='k', s=20)
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())

patch0 = mpatches.Patch(color='#FF0000', label='1')
patch1 = mpatches.Patch(color='#ff9933', label='2')
patch2 = mpatches.Patch(color='#FFFF00', label='3')
patch3 = mpatches.Patch(color='#00ffff', label='4')
patch4 = mpatches.Patch(color='#00FF00', label='5')
plt.legend(handles=[patch0, patch1, patch2, patch3,patch4])


plt.title("5-Class classification (k = %i, weights = '%s')"
              % (n_neighbors, 'distance'))

plt.show()
```

2.5. Elegir el mejor valor de k

```
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()
```

2.6. Clasificar ó Predecir nuevas muestras

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

3. Resultados

Visualización del archivo

Se observa que la distribución de "estrellas" no está balanceada, la gráfica de Valores de Sentimientos parece una campana movida levemente hacia la derecha del cero y la cantidad de palabras se centra sobre todo de 0 a 10.

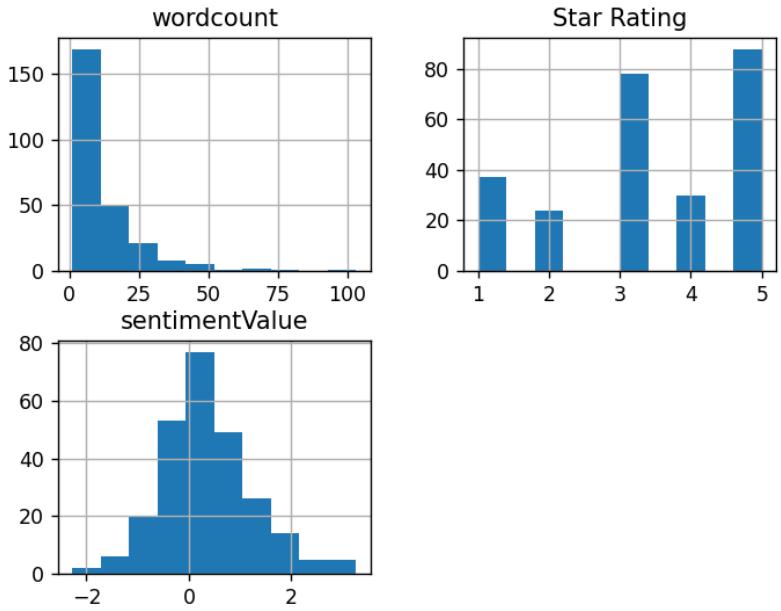

Figura 1: Gráficas de datos de entrada

Después se observa cuantas Valoraciones de Estrellas realmente se tienen, y se concluye que hay una mayoría de valoración de 3 y 5 estrellas.

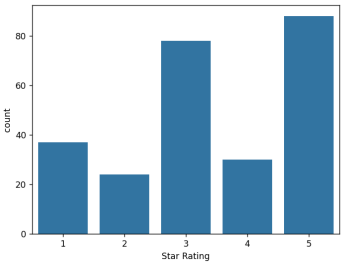

Figura 2: Gráficas de datos de entrada

También se grafica la mayoría de palabras y se observa que la mayoría de reviews tienen entre 1 y 10 palabras.

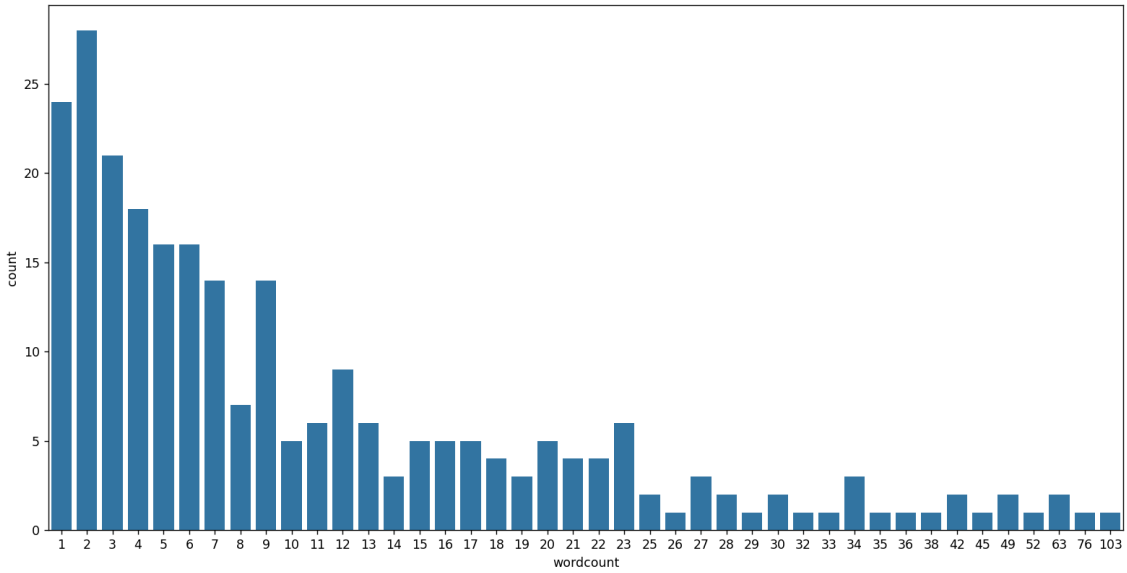

Figura 3: Gráficas de datos de entrada

Gráfica de la Clasificación Obtenida

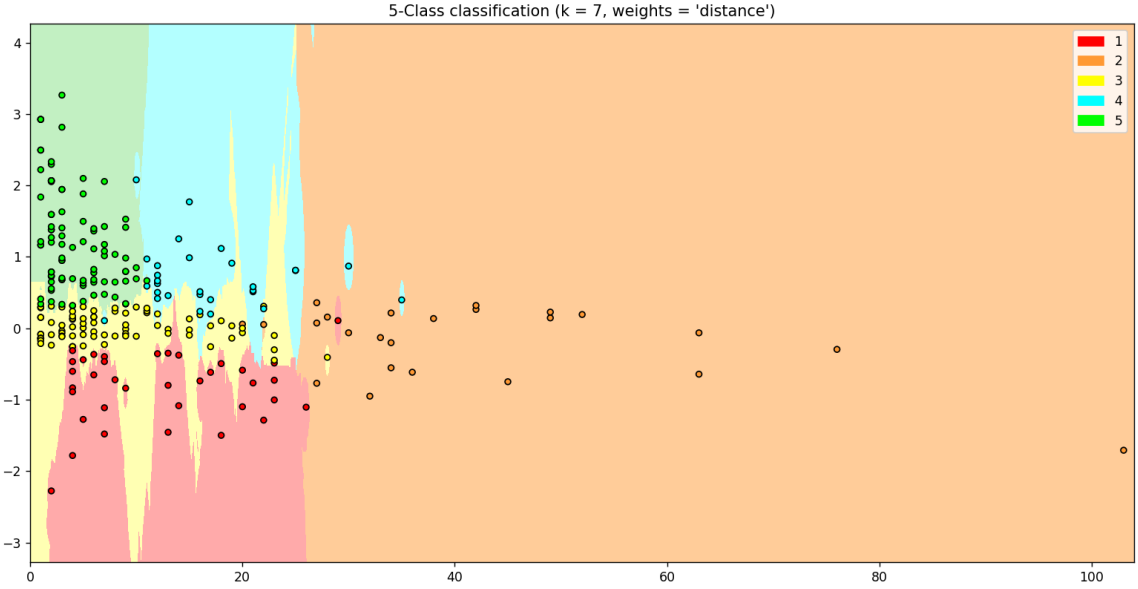

Figura 4: Gráficas de la Clasificación Obtenida

Algunos datos que se podrían "leer" de la gráfica son los siguientes:

- Los usuarios que ponen 1 estrella tienen sentimiento negativo y hasta 25 palabras.
- Los usuarios que ponen 2 estrellas dan muchas explicaciones (hasta 100 palabras) y su sentimiento puede variar entrene gativo y algo positivo.
- Los usuarios que ponen 3 estrellas son bastante neutrales en sentimientos, puesto que están en torno al cero y hasta unas 25 palabras.
- Los usuarios que dan 5 estrellas son bastante positivos (de 0, 5 en adelante, aproximadamente) y ponen pocas palabras (hasta 10).

El mejor valor de k

En la siguiente gráfica se muestra de donde se obtiene el valor de "k" utilizado en el código. Como resultado, se puede ver que el mejor valor de "k" es entre 7 y 14.

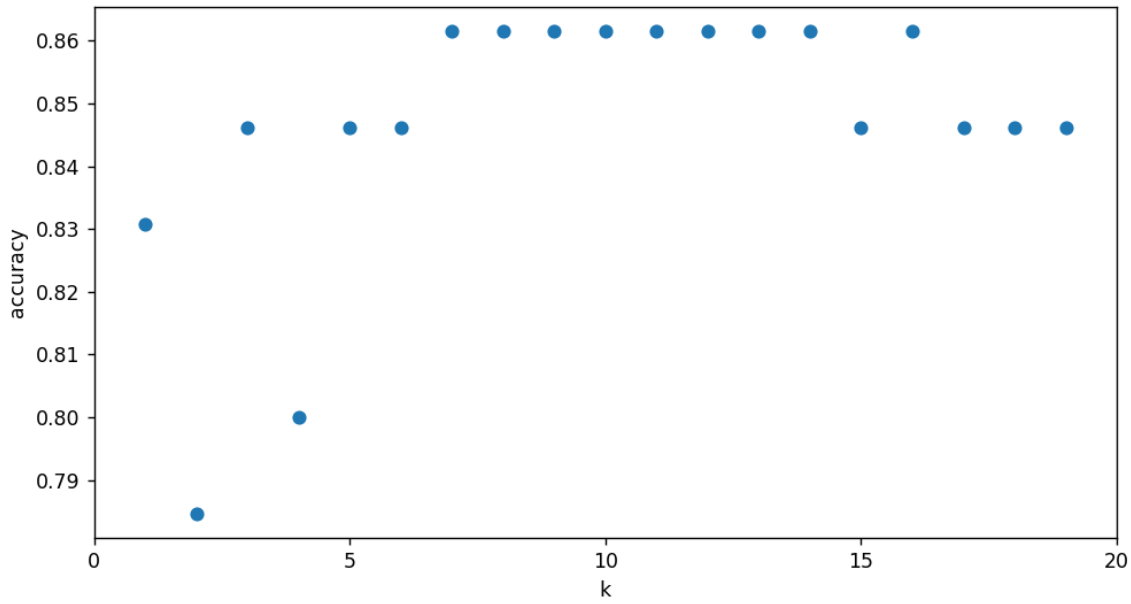

Figura 5: Gráficas de datos de entrada

Clasificar ó Predecir nuevas muestras

La siguiente imagen muestra la clasificación de una review de 5 palabras y sentimiento 1, es decir, 5 estrellas. Otra opción seria ver la probabilidad de obtener 1, 2, 3, 4 o 5 estrellas de una review de 20 palabras y sentimiento 0, que seria de 3 estrellas con una probabilidad del 97 %.

```
[5]
[[0.00381998 0.02520212 0.97097789 0.         0.        ]]
```

Figura 6: Gráficas de datos de entrada

4. Conclusión

Este ejercicio me permitió aplicar el algoritmo KNN (K-Nearest Neighbors) para clasificar reseñas según su calificación de estrellas, utilizando como características el conteo de palabras (wordcount) y el valor de sentimiento (sentimentValue). Desde mi punto de vista, el método KNN es requiere un bien análisis de datos, ya que es importante una buena elección del parámetro "k" y es necesario el uso de una mayor cantidad de datos y características si se quiere lograr un omdelo con una precisión.