Inteligencia Artificial
Act14: K-Nearest-Neighbor (KNN)

Arturo Garza Rodríguez

March 2025

1. Introducción

¿Qué es K-Nearest-Neighbor

El algoritmo K-Nearest-Neighbor (KNN) es un clasificador de aprendizaje no supervisado no paramétrico que utiliza la proximidad entre puntos de datos para realizar predicciones o clasificaciones. Este algoritmo se basa en la idea de que los puntos de datos similares tienden a agruparse, y clasifica un punto nuevo en función de los puntos más cercanos a él dentro del conjunto de entrenamiento. KNN es ampliamente utilizado debido a su simplicidad y facilidad de implementación, y puede ser aplicado tanto a problemas de clasificación como de regresión. Su popularidad radica en su capacidad para proporcionar soluciones efectivas sin necesidad de hacer supuestos sobre la distribución de los datos.

2. Metodología

2.1. Requerimientos

2.1.1. Jupyter notebook y dataset

En este trabajo estarémos utilizando un entorno de una *Jupyter Notebook* para segmentar el código e ir tomando nota de los resultados con cada procedimiento.
En este caso usaremos el dataset 'reviews_sentiment.csv' que se nos proporciona en el apartado pertinente del libro.

2.2. Desarrollo de código

2.2.1. Imports

```python
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

%matplotlib inline
plt.rcParams['figure.figsize'] = (16, 9)
plt.style.use('ggplot')

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
```

Python

Figura 1: Se realizan los imports de las librerías necesarias para configurar nuestro modelo KNN.

2.2.2. Definición de dataset

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

Python

	Review Title	Review Text	wordcount	titleSentiment	textSentiment	Star Rating	sentimentValue
0	Sin conexión	Hola desde hace algo más de un mes me pone sin...	23	negative	negative	1	-0.486389
1	faltan cosas	Han mejorado la apariencia pero no	20	negative	negative	1	-0.586187
2	Es muy buena lo recomiendo	Andres e puto amoooo	4	NaN	negative	1	-0.602240
3	Version antigua	Me gustana mas la version anterior esta es mas...	17	NaN	negative	1	-0.616271
4	Esta bien	Sin ser la biblia.... Esta bien	6	negative	negative	1	-0.651784
5	Buena	Nada del otro mundo pero han mejorado mucho	8	positive	negative	1	-0.720443
6	De gran ayuda	Lo malo q necesita de ....,pero la app es muy buena	23	positive	negative	1	-0.726825
7	Muy buena	Estaba más acostumbrado al otro diseño, pero e...	16	positive	negative	1	-0.736769
8	Ta to guapa.	Va de escándalo	21	positive	negative	1	-0.765284
9	Se han corregido	Han corregido muchos fallos pero el diseño es ...	13	negative	negative	1	-0.797961

Figura 2: Se lee el dataset usando la librería pandas y se imprimen los primeros 10 registros, para ver el formato del dataset, a través de *data.head(10)*

```python
dataframe.describe()
```

Python

	wordcount	Star Rating	sentimentValue
count	257.000000	257.000000	257.000000
mean	11.501946	3.420233	0.383849
std	13.159812	1.409531	0.897987
min	1.000000	1.000000	-2.276469
25%	3.000000	3.000000	-0.108144
50%	7.000000	3.000000	0.264091
75%	16.000000	5.000000	0.808384
max	103.000000	5.000000	3.264579

Figura 3: *data.describe()* es útil para obtener una visión rápida de la distribución y características de los datos en el DataFrame.

2.2.3. Análisis gráfico

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

Python

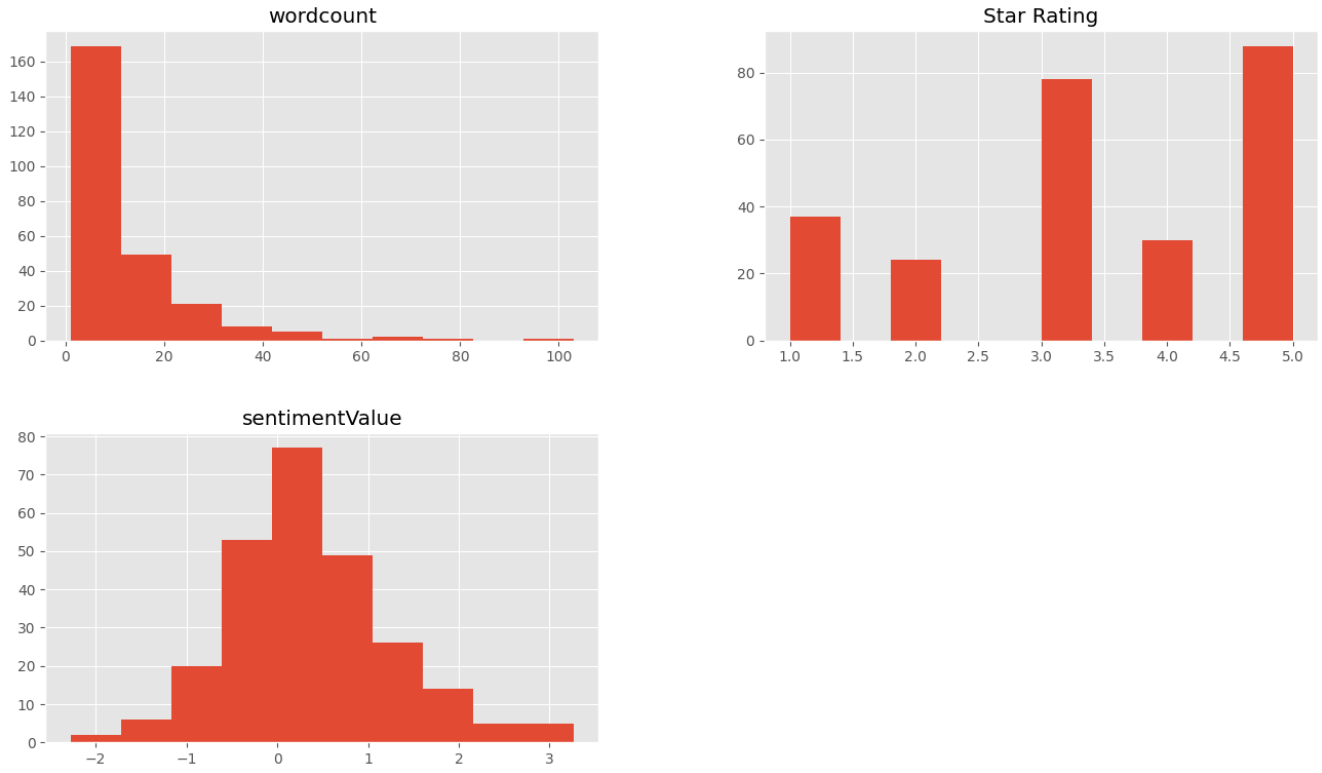

Figura 4: Vemos que la distribución de "estrellas" no está balanceada, esto no es bueno. Convendría tener las mismas cantidades en las salidas, para no tener resultados "tendenciosos".

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

Python

```
Star Rating
1    37
2    24
3    78
4    30
5    88
dtype: int64
```

+ Code + Markdown

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

Python

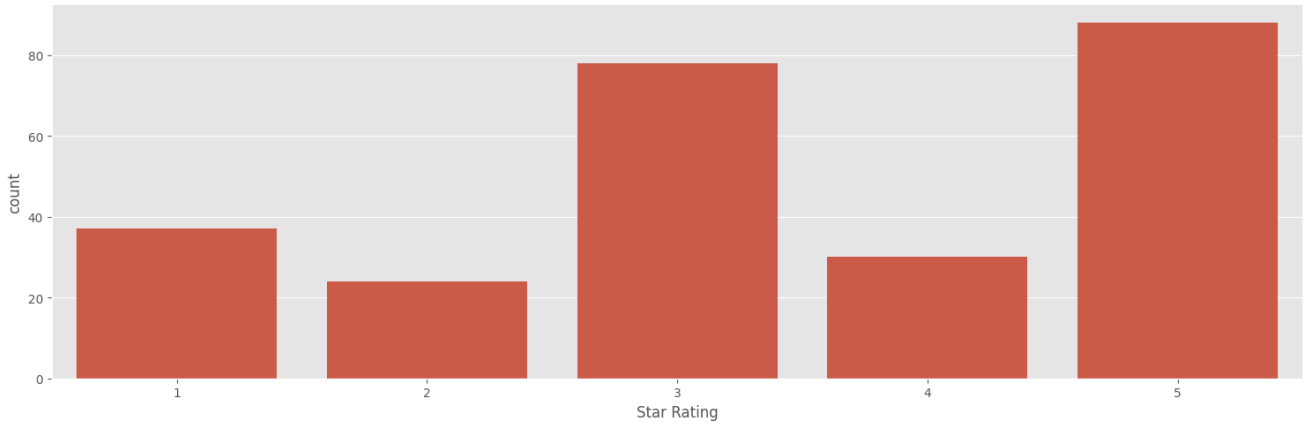

Figura 5: Con eso confirmamos que hay sobre todo de 3 y 5 estrellas.

```
sb.catplot(x='wordcount',data=dataframe,kind="count", aspect=3)
```

Python

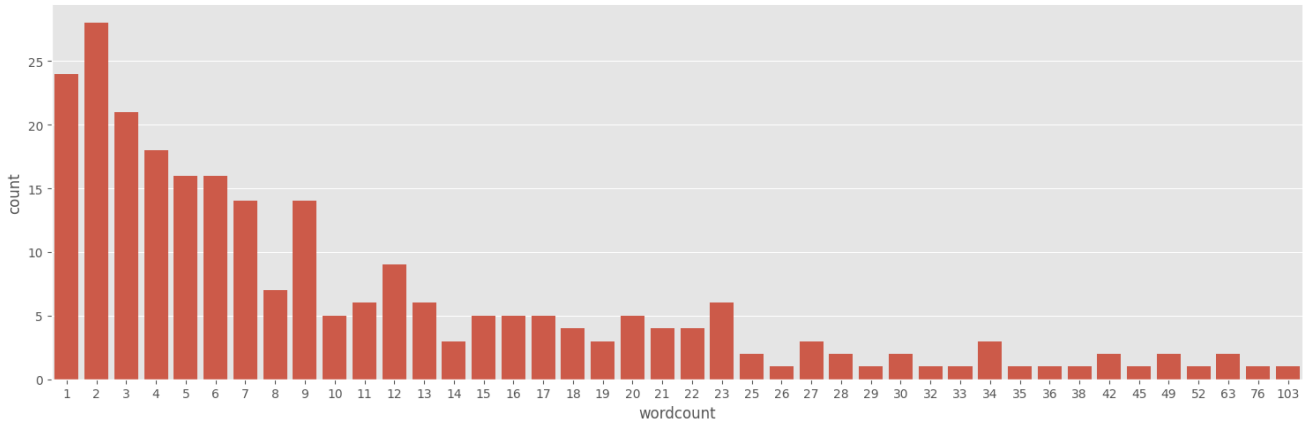

Figura 6: Graficamos mejor la cantidad de palabras y confirmamos que la mayoría están entre 1 y 10 palabras.

2.2.4. KNN Model

```python
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)
```

Python

```python
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)))
```

Python

```
Accuracy of K-NN classifier on training set: 0.90
Accuracy of K-NN classifier on test set: 0.86
```

Figura 7: Vemos que la precisión que nos da es de 90 % en el set de entrenamiento y del 86 % para el de test.

3. Resultados

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

Python

```
[[ 9  0  1  0  0]
 [ 0  1  0  0  0]
 [ 0  1 17  0  1]
 [ 0  0  2  8  0]
 [ 0  0  4  0 21]]
              precision    recall  f1-score   support

           1       1.00      0.90      0.95        10
           2       0.50      1.00      0.67         1
           3       0.71      0.89      0.79        19
           4       1.00      0.80      0.89        10
           5       0.95      0.84      0.89        25

    accuracy                           0.86        65
   macro avg       0.83      0.89      0.84        65
weighted avg       0.89      0.86      0.87        65
```

Figura 8: Como se ve la puntuación F1 es del 87 %, bastante buena

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

cmap_light = ListedColormap(['#FFAAAA', '#ffcc99', '#ffffb3','#b3ffff','#c2f0c2'])
cmap_bold = ListedColormap(['#FF0000', '#ff9933','#FFFF00','#00ffff','#00FF00'])
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)

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()])
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

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'))
```

Figura 9: Realizaremos la gráfica con la clasificación obtenida, la que nos ayuda a ver fácilmente en donde caerán las predicciones.

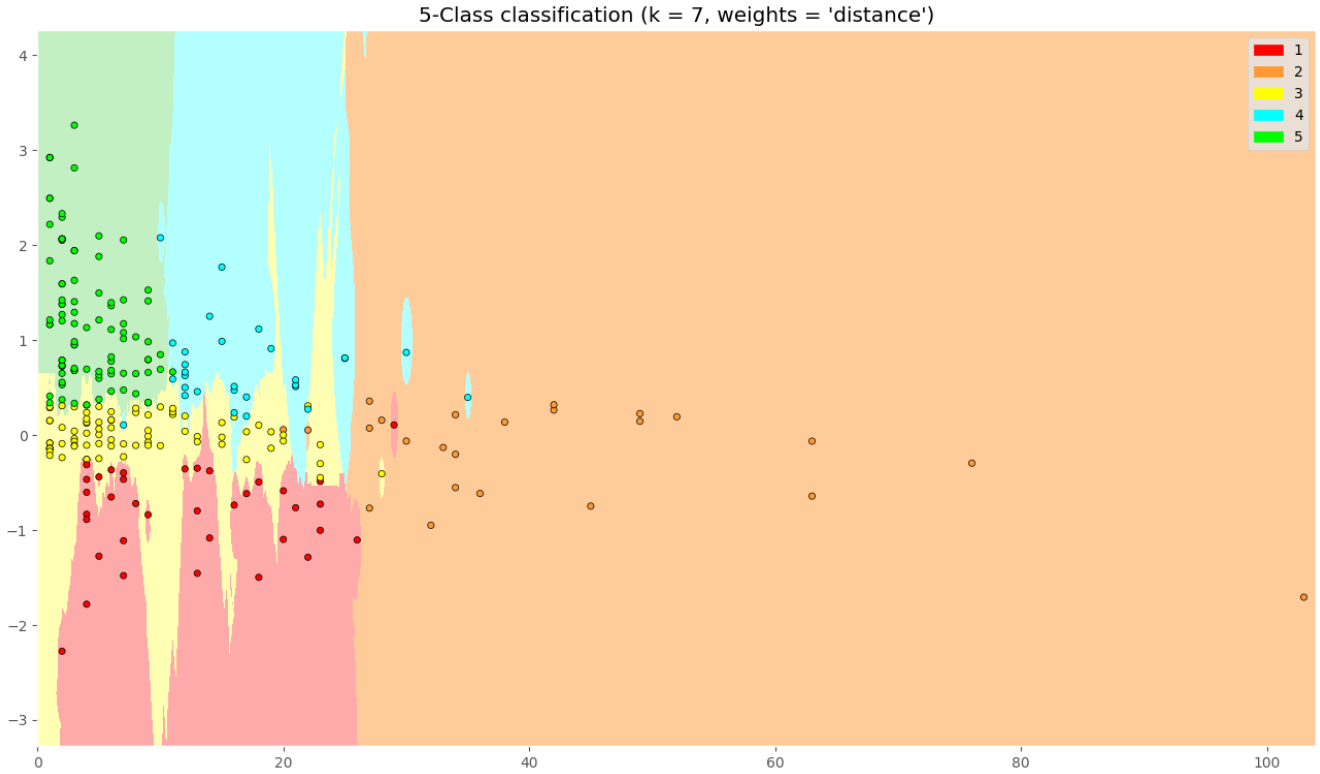

Figura 10: Vemos las 5 zonas en las que se relacionan cantidad de palabras con el valor de sentimiento de la Review que deja el usuario..

4. Conclusión

En este trabajo, se implementó el algoritmo K-Nearest Neighbors (KNN) para clasificación, obteniendo una precisión del 90 % en el conjunto de entrenamiento y del 86 % en el de prueba. El modelo mostró ser eficiente y fácil de implementar, con un puntaje F1 de 87 %. Sin embargo, la distribución desequilibrada de las clases podría haber influido en los resultados. En general, KNN demostró ser una herramienta efectiva para problemas de clasificación en Machine Learning.

5. Referencias

- Material de clase
- https://datascientest.com/en/knn-what-is-the-knn-algorithm
- https://www.ibm.com/mx-es/topics/knn