K-Nearest-Neighbor

Valeria Ybarra López

29, Marzo 2025

1 Introducción

El algoritmo K-Nearest-Neighbor (K-NN) es un modelo de tipo supervisado basado en instancias, utilizado en Machine Learning. Se puede aplicar para clasificar nuevas muestras (valores discretos) o para realizar predicciones (regresión, valores continuos). Su simplicidad lo convierte en una buena opción para principiantes. El algoritmo K-NN consiste en clasificar valores al identificar los puntos de datos más cercanos o similares aprendidos durante la fase de entrenamiento, lo que permite inferir nuevos puntos basándose en estas similitudes. Tiene como ventaja, su facilidad de comprensión y aplicación. K-NN es más eficiente en datasets pequeños y con pocas características (columnas).

2 Metodología

2.1 Carpeta K-Nearest-Neighbor

Creamos una carpeta llamada K-Nearest-Neighbor en donde descargaremos el archivo csv proporcionado por el libro para poder realizar la actividad, también crearemos en esa misma carpeta un código .py para la codificación de la actividad.

2.2 Archivo csv

El archivo nos proporciona con 257 registros de opiniones de usuarios sobre una app(Reviews).

2.3 Código

Comenzamos importando las librerías:

```
1 import pandas as pd
2 import numpy as np
3 import matplotlib.pyplot as plt
4 from matplotlib.colors import ListedColormap
5 import matplotlib.patches as mpatches
6 import seaborn as sb
```

```
7
8  plt.rcParams['figure.figsize'] = (16, 9)
9  plt.style.use('ggplot')
10
11 from sklearn.model_selection import train_test_split
12 from sklearn.preprocessing import MinMaxScaler
13 from sklearn.neighbors import KNeighborsClassifier
14 from sklearn.metrics import classification_report
15 from sklearn.metrics import confusion_matrix
```

Leemos archivo de entrada

Leemos el csv con pandas, usando separador de punto y coma, pues en las reviews hay textos que usan coma. Con .head(10) vemos los 10 primeros registros. También usaremos .describe() para ver las estadísticas de los datos:

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

Visualizaciones

Usamos .hist() para poder ver los datos en gráficas:

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

Dividimos los datos en grupos según los valores en la columna "Star Rating", y usamos el método .size() para ver el número de elementos que hay en cada grupo, también graficamos estos resultados:

```
1 print(dataframe.groupby('Star␣Rating').size())
2 sb.catplot(x='Star␣Rating',data=dataframe,kind="count", aspect=3)
3 plt.show()
```

Graficamos también la cantidad de palabras ('wordcount') para aseguranos que los elementos esten entre 1 y 10 palabras:

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

Preparamos las entradas

Como variables de entrada tenemos a "X" y "y", creamos los sets de entrenamiento y test:

```
1  X = dataframe[['wordcount','sentimentValue']].values
2 y = dataframe['Star␣Rating'].values
3
4 X_train, X_test, y_train, y_test = train_test_split(X, y,
      random_state=0)
5 scaler = MinMaxScaler()
6 X_train = scaler.fit_transform(X_train)
7 X_test = scaler.transform(X_test)
```

Modelo K-NN con Scikit Learn

Al crear nuestro modelo K-NN, definimos el valor de k en 7 y creamos el clasificador, también usando knn.score, imprimimos la precisión del set de entrenamiento y del test:

```
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)))
```

Resultados obtenidos del entrenamiento

Imprimimos la precisión del entrenamiento de los datos:

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

Gráfica de la Clasificación Obtenida

Graficaremos la clasificación obtenida para poder ver donde caerán las predicciones:

```
h = .02

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'))

plt.show()
```

Mejor valor de K

Realizamos un análisis para evaluar el rendimiento del modelo. Para cada valor de k, el modelo se entrena con los datos: X_train, y_train, y se mide su precisión con los datos de prueba: X_test, y_test. Guardamos el resultado en la lista "score", y por último graficamos cómo cambia la precisión según el 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])
```

Predecir nuevas muestras

Ya teniendo nuestro modelo entrenado, comenzemos a usarlo. Vamos a predecir las estrellas de un usuario de dos maneras, la primera manera (usando el método .predict()):

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

Para que las probabilidades nos den 1,2,3,4 o 5 estrellas, usamos .predict_proba():

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

3 Resultados

Datos de entrada

	Review Title	...	sentimentValue
0	Sin conexión	...	-0.486389
1	faltan cosas	...	-0.586187
2	Es muy buena lo recomiendo	...	-0.602240
3	Version antigua	...	-0.616271
4	Esta bien	...	-0.651784
5	Buena	...	-0.720443
6	De gran ayuda	...	-0.726825
7	Muy buena	...	-0.736769
8	Ta to guapa.	...	-0.765284
9	Se han corregido	...	-0.797961

[10 rows x 7 columns]

Figure 1: Resultado de dataframe.head(10), nos muestra los primeros 10 registros.

	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

Figure 2: Vemos las estadísticas de los datos, son 257 registros,las estrellas ("Star Rating") van del 1 al 5 la cantidad de palabras ("wordcount") van de 1 hasta 103, y las valoraciones de sentimiento ("sentimentValue") están entre -2.27 y 3.26.

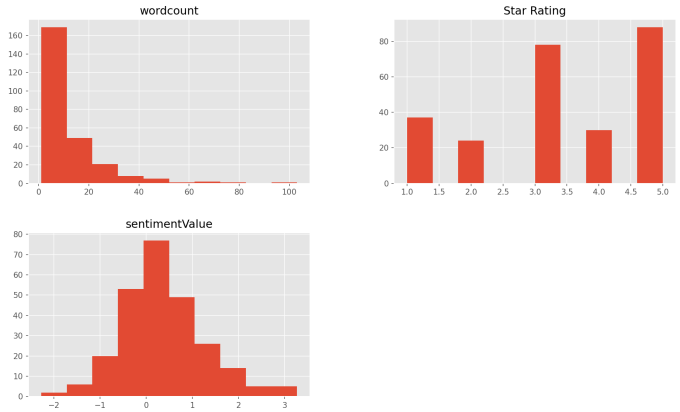

Figure 3: La distribución de "Star Rating" no esta balanceada.

Star Rating	
1	37
2	24
3	78
4	30
5	88

dtype: int64

Figure 4: Podemos ver que hay mayor cantidad de elementos en el grupo de 3 y 5 estrellas.

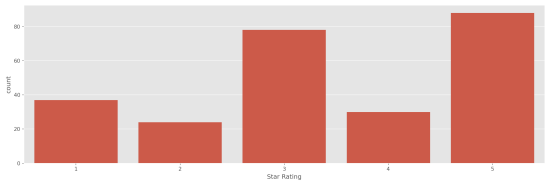

Figure 5: Gráfica con los elementos de cada grupo de la columna "Star Rating"

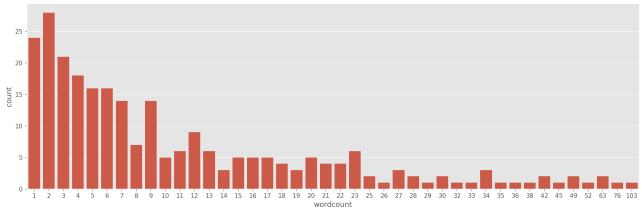

Figure 6: Gráfica de la cantidad de palabras

Modelo K-NN

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

Figure 7: El set de entrenamiento tiene una presición de 90% y el test un 86%

Resultado Precisión de entrenamiento

```
[[ 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
```

Figure 8: La puntuación F1 es del 87% lo cual es bueno.

Clasificación Obtenida

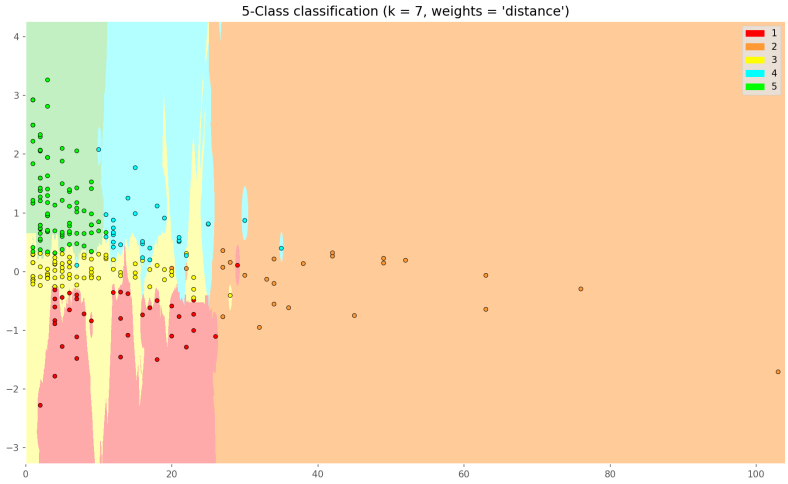

Figure 9: Vemos las 5 zonas en las que se relacionan cantidad de palabras con el valor de sentimiento de la Review que deja el usuario.

- 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 entre negativo 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).

Mejor valor de K

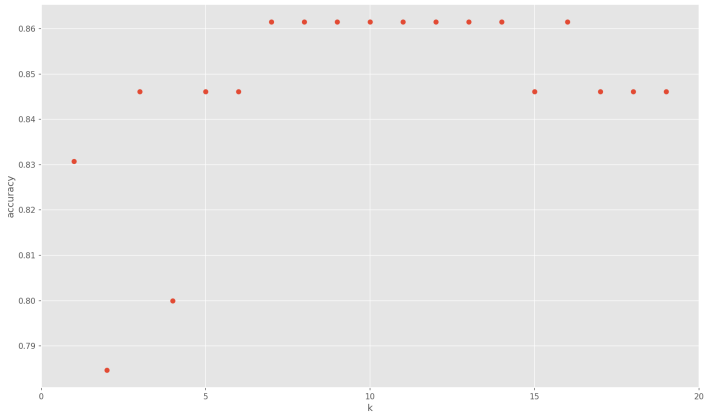

Figure 10: La gráfica muestra que al tener k=7 a k=14 se logra mayor precisión.

Predecir nuevas muestras

```
[5]
```

Figure 11: El resultado de la primera predicción, nos da un valor de [5] lo cual nos indica que para 5 palabras y sentimiento 1, nos valorarán la app con 5 estrellas.

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

Figure 12: El resultado de la segunda predicción nos dice que para las coordenadas de 20,0.0 hay un 97% probabilidad de que nos den 3 estrellas.

4 Conclusión

Esta actividad nos ayudó a desarrollar un modelo en Python, empleando el algoritmo K-Nearest Neighbor. Este método se basa en analizar los "k vecinos más cercanos" para clasificar nuevos puntos. En este tipo de algoritmo es necesario tener con un número adecuado de muestras para garantizar un entrenamiento eficiente del modelo, en este ejercio solo usamos dos columnas para entrenar el modelo pero de todas maneras las predicciones tenian un buen resultado de precisión. Realizar nuevas predicciones me permitió entnder mejor como funciona el algoritmo K-NN.

5 Referencias

Bagnato, J. (2020). Aprende Machine Learning en Español.