Actividad #14: Programando K-Nearest-Neighbor en Python

Nombre: Dayla Marely Carrizales Ortega

Matricula: 1952471

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. Como pros tiene sobre todo que es sencillo de aprender e implementar. Tiene como contras que utiliza todo el dataset para entrenar “cada punto” y por eso requiere de uso de mucha memoria y recursos de procesamiento (CPU). Por estas razones kNN tiende a funcionar mejor en datasets pequeños y sin una cantidad enorme de features (las columnas).

2. Metodología

Lo principal en esta actividad fue la carga del archivo CSV, en mi caso lo descargue directamente de las bases de datos de Kaggle. Este archivo o base de datos contiene dos características importantes: el número de palabras en la reseña wordcount y un valor llamado sentimentValue. También se incluyó la clasificación de la reseña, representada por la columna Star Rating, además de algunas estadísticas y la distribución de las calificaciones en estrellas.

Hicimos la extracción de datos, los mencionados anteriormente (wordcount, sentimentValue) y las etiquetas Star Rating para el entrenamiento y prueba del modelo. Los datos fueron escalados utilizando MinMaxScaler para asegurarse de que las características tuvieran un rango comparable.

En el caso del entrenamiento del modelo, fue un modelo K-NN con un número de vecinos k=7. A continuación, se evaluó el modelo utilizando la precisión en los conjuntos de entrenamiento y prueba, y se presentó la matriz de confusión y el informe de clasificación.

Los resultados fueron complementados con la matriz de confusión y las métricas de clasificación.

Para visualizar los datos, se utilizó la frontera de decisión del modelo K-NN en el espacio de características, lo que permitió observar cómo el modelo clasifica las diferentes reseñas según su número de palabras y valor de sentimiento.

Exploramos el rendimiento del modelo con diferentes valores de K. Esto se hizo evaluando la precisión del modelo en el conjunto de prueba para diferentes valores de K, y luego se visualizó

Por último, se graficaron los resultados para mostrar cómo varía la precisión con el valor de K.

2.1 Código K-Nearest-Neighbor

```
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

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

dataframe = pd.read_csv(r"reviews_sentiment.csv",sep=';')
dataframe.head(10)
dataframe.describe()
dataframe.hist()
plt.show()

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

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

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)

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)))

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

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','#00FF00'])

# 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()

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])

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

3. Resultados

Los resultados obtenidos son:

- La matriz de confusión mostró que el modelo fue capaz de predecir correctamente muchas de las clasificaciones de las reseñas, pero también cometió algunos errores, especialmente en las clases de calificación intermedia.
- El informe de clasificación proporcionó una visión detallada de las métricas de rendimiento para cada clase, mostrando que el modelo fue más preciso en la clasificación de reseñas con calificaciones extremas (1 y 5 estrellas), pero menos preciso en calificaciones intermedias.
- La visualización de la frontera de decisión permitió observar cómo el modelo separa las clases de calificación en función del número de palabras y el valor de sentimiento, proporcionando una representación clara de cómo el modelo realiza las clasificaciones.

4. Conclusión

El algoritmo K-Nearest Neighbors (K-NN) demostró ser efectivo, aunque el modelo mostró una buena precisión, especialmente para las calificaciones, la precisión general podría mejorarse aún más ajustando parámetros como el valor de K y explorando otras técnicas de preprocesamiento o características adicionales.