k-Nearest Neighbor en Python

1 Introducción

k-Nearest Neighbors (k-NN) es un algoritmo de aprendizaje supervisado no paramétrico basado en instancias que clasifican nuevas observaciones según la mayoría de votos de sus k vecinos más cercanos en el espacio de características. Este modelo es más usado en problemas de clasificación, cuando se tiene una variable dependiente categórica. k-NN hace predicciones basándose en el consenso local de una pluralidad de sus estimadores k. Al igual que con el modelo Random Forest, k-NN también se puede usar para la predicción de valores númericos promediando los valores.

2 Metodología

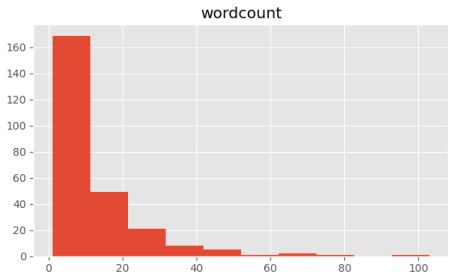
Para realizar el ejercicio de regresión logística, tuve que seguir los siguientes pasos: Visualización de los datos de entrada, Creación y Ajuste del modelo y Evaluación del Modelo.

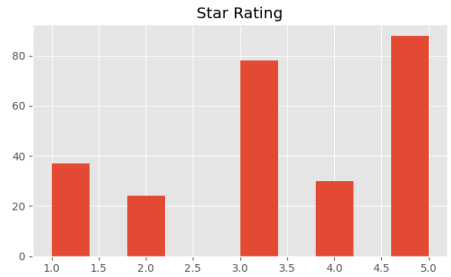
```
#Importamos las librerias
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap
import matplotlib.patches as mpatches
import seaborn as sb

%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

#Creamos un dataframe
dataframe = pd.read_csv(r"reviews_sentiment.csv",sep=';')
```

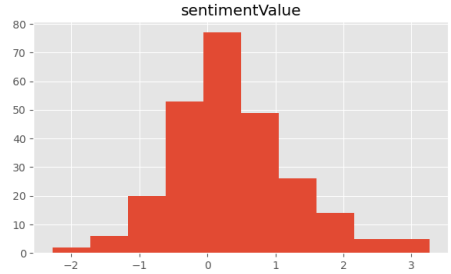
2.1 Visualización de los datos de entrada

En esta sección vamos a visualizar las distribuciones de las columnas del *dataset*.

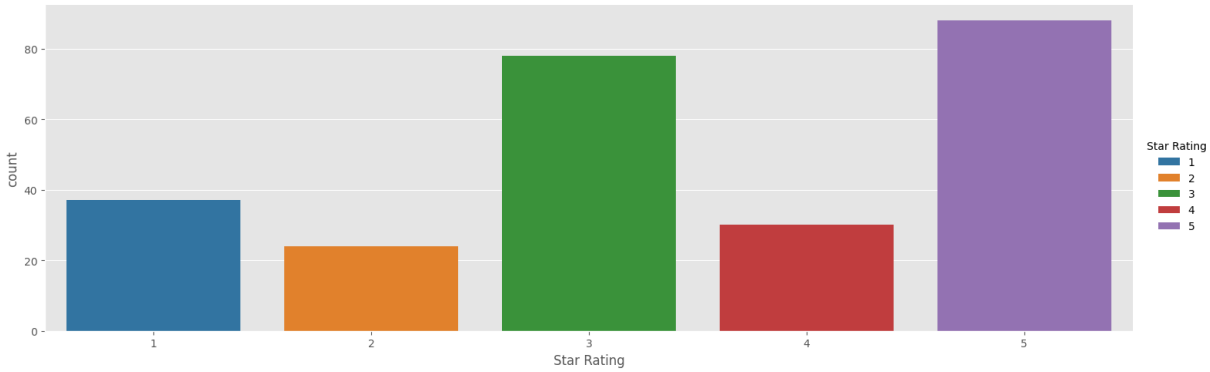
```
dataframe.hist()
plt.show()
```



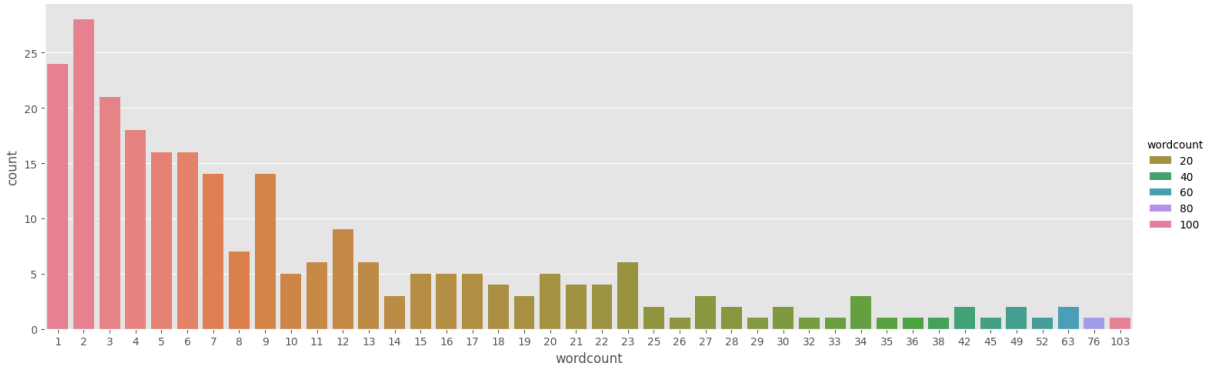





```
sb.catplot(x='Star Rating',data=dataframe,kind="count",
    aspect=3, hue='Star Rating', palette='tab10')
```




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



Vamos a utilizar las columnas del sentimiento del usuario y de la cantidad de palabras en la reseña para predecir el número de estrellas.

2.2 Creación y Ajuste del modelo

En esta sección vamos a separar los datos en conjuntos de entrenamiento y prueba para ajustar el modelo de regresión k-Nearest Neighbor.

```
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)))
```

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

Debido a que el *dataset* tiene relativamente pocos datos y porque los valores en el número de estrellas no están muy desequilibrados, es mejor utilizar una cantidad de k más pequeña. En este caso podemos ver que el modelo es muy preciso, incluso con los datos de prueba que no ha visto previamente.

2.3 Evaluación del modelo

En esta sección vamos a realizar las predicciones del modelo e imprimir la matriz de confusión. Esta matriz nos va a mostrar la diferencia entre la clase predecida y la clase real de cada observación del conjunto de prueba.

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

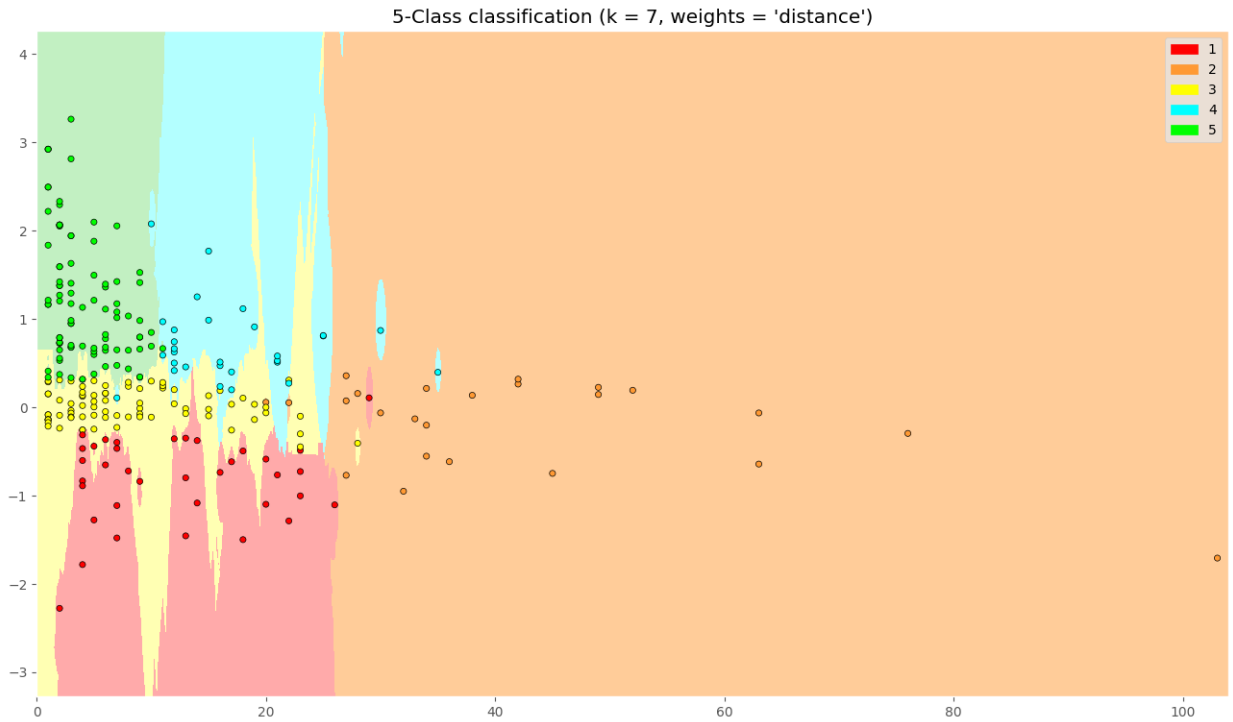
```
[[ 9  0  1  0  0]
[ 0  1  0  0  0]
[ 0  1 17  0  1]
[ 0  0  2  8  0]
[ 0  0  4  0 21]]
              precision    recall  f1-score   support

           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
```

Podemos ver que el modelo casi siempre pudo predecir reseñas malas (1 estrella) y reseñas buenas (4 a 5 estrellas) pero tuvo más problemas con las reseñas a mediación (2 a 3 estrellas). Sin embargo, el modelo pudo clasificar correctamente más de la mitad de estos casos.

3 Resultados

En esta sección vamos a graficar los resultados vistos en la sección anterior. En la gráfica de colores siguiente los dos ejes representan las dos columnas de X y los espacios de colores representa cada una de las clases de nuestra variable dependiente (la cantidad de estrellas en la reseña).

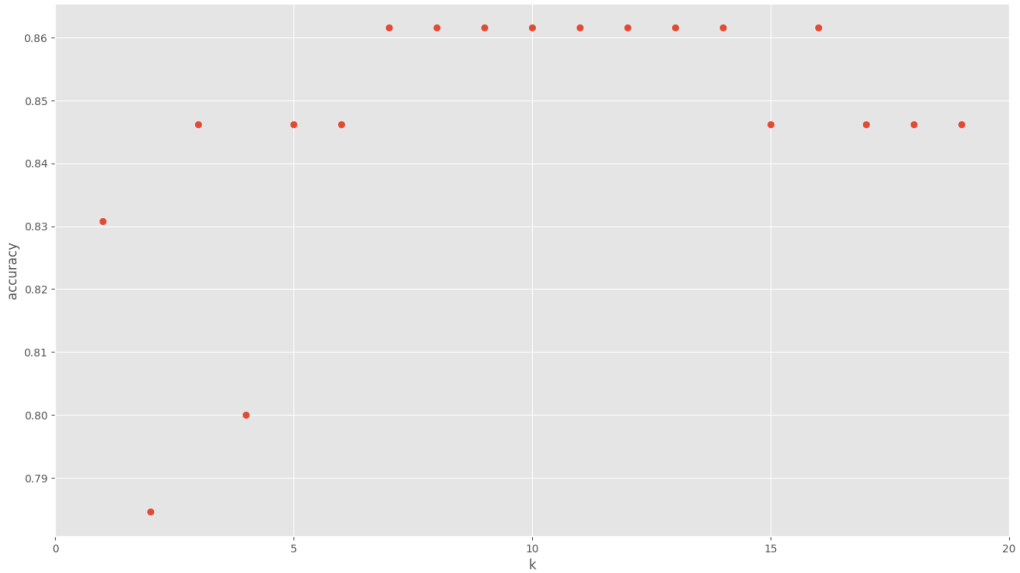
```
#Tamaño de pasos en la malla
h = .02

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

#Creamos una instancia de k-Nearest Neighbors y ajustamos los datos.
clf = KNeighborsClassifier(n_neighbors, weights='distance')
clf.fit(X, y)

#Graficamos el límite de decisiones al asignar un color a cada
#punto en la malla
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()])

#Graficamos el resultado en una gráfica de colores
Z = Z.reshape(xx.shape)
plt.figure()
plt.pcolormesh(xx, yy, Z, cmap=cmap_light)

#Graficamos los puntos de entrenamiento
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()
```



Como nos muestra la gráfica y como se veía en el histograma de la cantidad de palabras la mayoría de las reseñas no son muy largas, pero algo nuevo que nos muestra esta gráfica es que la mayoría de las reseñas que tienen muchas palabras son de 2 estrellas. Todas las otras clases de la variable están concentradas en la parte izquierda de la gráfica, es decir, las reseñas de 1 estrella o de 3 a 5 son más probables de no tener palabras o muy pocas.

Ahora vamos a checar cuantas k necesitamos en el modelo para que este sea más preciso.

```
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])
```



Como se puede ver en la gráfica anterior, el modelo da los mejores resultados cuando hay entre 7 y 14 estimadores k.

4 Conclusión

Los modelos de k-Nearest Neighbor (k-NN) son herramientas del aprendizaje supervisado que hace predicciones basandose en una pluralidad de resultados dados por sus estimadores k. Lo que lo hace diferente a otros modelos que se basan en clasificación por consenso es que k-NN permite asignar pesos a las distancias entre estimadores k, lo que permite crear una preferencia por valores que están cerca de sí mismos.

La cantidad de k en un modelo k-NN depende mucho de las características de los datos que se están usando. Un *dataset* más pequeño o con clases balanceadas necesita menos k que un *dataset* más grande o con clases desequilibradas. Muchas k en un modelo también puede causar $overfitting$ y empeorar la precisión del modelo con datos que no ha visto.

En el caso de este *dataset*, se descubrió que las clases de nuestra variable dependiente (el número de estrellas de una reseña) tienen una relación peculiar con el número de palabras de la reseña: las reseñas con más palabras son de 2 estrellas mientras que las reseñas cortas pueden ser de cualquier otro número. También se uso la columna de sentimiento del usuario, la cual es similar a lo que estamos intentando predecir y lo que le permitió al modelo clasificar correctamente las reseñas. También se vió que la cantidad óptima de estimadores k para este modelo es entre 7 y 14.