KNN Algorithm

The k-nearest neighbor algorithm, sometimes referred to as KNN or k-NN, is a nonparametric supervised learning classifier that relies on closeness to assign labels to or predict the grouping of a single data point.

Although it can be applied to regression or classification issues, it is typically employed as a classification technique because it is predicated on the idea that similar points can be identified nearby.

A majority vote is used to apply a class label to classification problems, meaning the label that is most frequently expressed around a particular data point is utilized. Although this counts as "majority voting," the term "majority vote" is more frequently used in literature. These terms differ in that a "majority vote" actually requires a majority of more than 50%, which only applies when there are only two options. When there are many classes, such as four categories, deciding on a class does not always require a majority of votes; a class label can be assigned with a vote of more than 25%.

Classification difficulties use a notion similar to regression problems, except in this situation, a classification prediction is made using the average of the k nearest neighbors. Classification is utilized for discrete data whereas regression is used for continuous values in this situation. However, defining the distance is necessary before a categorization can be determined. The most popular measurement is euclidean distance, and we shall go into more detail later.

Noting that the KNN algorithm just saves a training data set rather than going through a training phase, it should be noted that it belongs to a family of "lazy learning" models. This also implies that whenever a classification or prediction is made, all calculation takes place. It is also known as an instance-based or memory-based learning approach since it stores all of its training data entirely in memory.

General Pseudocode

```
Function kNN_Classify(dataset: Array of Example,
          unlabeled: Example,
          k: Integer) -> Label:
        Step 1: Compute distances between the unlabeled example and all examples in the dataset
  distances = []
  For each example in dataset:
    distance = ComputeDistance(unlabeled, example)
    distances.append((distance, example))

        Step 2: Sort the distances in ascending order
  Sort(distances)

        Step 3: Select the k smallest distances
  kNearestNeighbors = distances[0:k]

        Step 4: Get the labels of the k nearest neighbors
  kLabels = []
  For each (_, example) in kNearestNeighbors:
    kLabels.append(example.label)

         Step 5: Determine the most common label among the k neighbors
  predictedLabel = GetMostCommonLabel(kLabels)

  Return predictedLabel

Function ComputeDistance(a: Example, b: Example) -> Float:
  This can be any distance metric, e.g., Euclidean distance
```

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(y_i - x_i)^2}$$

```
  Return distance

Function GetMostCommonLabel(labels: Array of Label) -> Label:
  Count the frequency of each label
  labelCounts = CountFrequencies(labels)

  Sort labels by frequency
  sortedLabels = SortByFrequency(labelCounts)

  Return the label with the highest frequency
  Return sortedLabels[0]
```

Pseudocode

1. Import necessary libraries

- Data handling and manipulation: pandas, numpy
- Plotting: matplotlib, seaborn
- Machine Learning: sklearn

2. Load the dataset
    - Assume that the dataset is loaded into a variable named 'dataframe'
3. Display general statistics of the dataset
    - Print dataframe.describe()
4. Prepare the features and target variable
    - X = dataframe[['wordcount','sentimentValue']]
    - y = dataframe['Star Rating']
5. Split the dataset into training and testing sets
    - Use train_test_split method from sklearn and split X, y
    - Assign the result to X_train, X_test, y_train, y_test
6. Scale the features
    - Create a MinMaxScaler object
    - Fit and transform X_train using the scaler
    - Transform X_test using the same scaler
7. Choose the appropriate 'k' value for K-NN
    - Iterate k from 1 to 20
        - Create a KNeighborsClassifier with k neighbors
        - Fit the classifier on the training set
        - Compute the accuracy on the test set and store it
    - Plot the accuracies for all k values
8. Create a K-NN classifier with the chosen 'k' value, e.g., k=7
    - Fit the classifier on the training set
    - Print the accuracy on both training and test sets
9. Display confusion matrix and classification report
    - Predict the classes of the test set
    - Print the confusion matrix and classification report
10. Visualize the decision boundaries
    - Create a mesh grid of the feature space
    - Predict the class for each point in the mesh grid
    - Plot the decision boundaries with different colors for each class
    - Plot the training points on the same plot with corresponding colors
    - Add legends and title to the plot
    - Display the plot

```
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
```

```
# Here we see the dataset to be worked
dataframe = pd.read_csv(r"reviews_sentiment.csv",sep=';')
dataframe.head(10)
```

	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

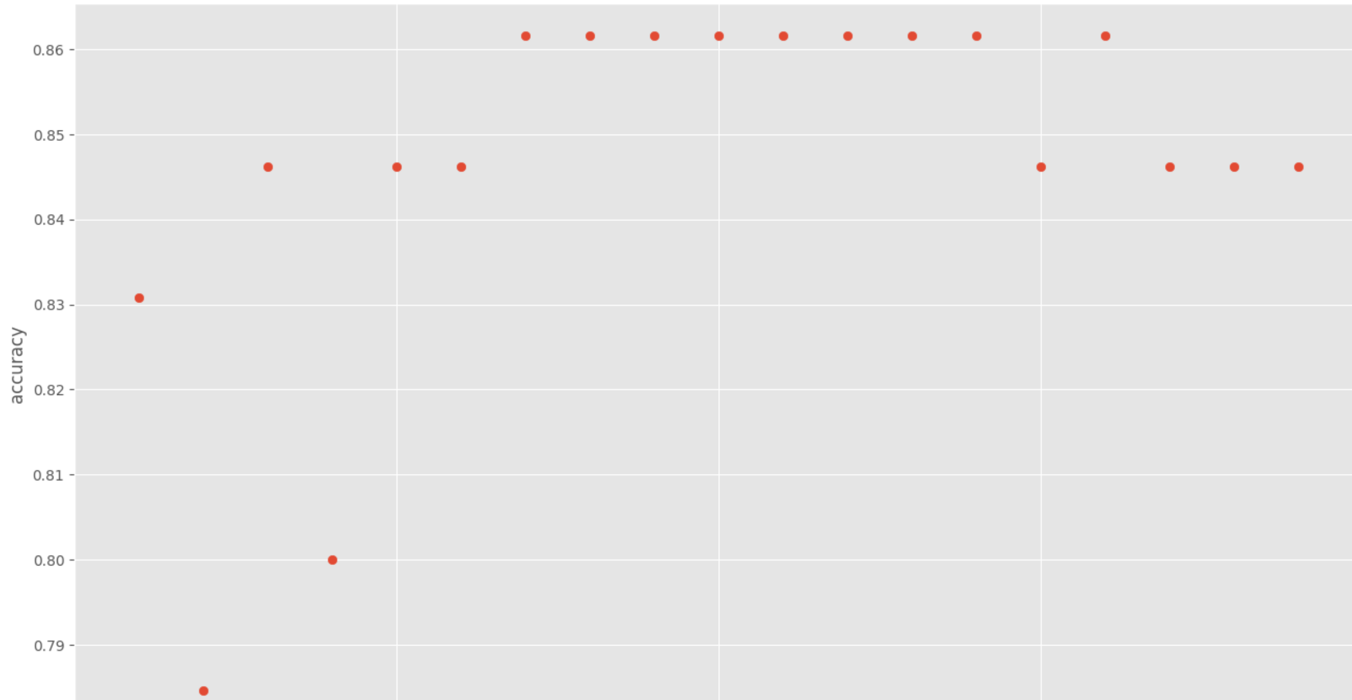
```
# We se general Statistics
dataframe.describe()
```

	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

```
# We define our X and Y, as test as well as train
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)
```

```
#This code help to us to vizualize wich K is better for the problem
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])
```

```
([<matplotlib.axis.XTick at 0x7ce0dc7eada0>,
  <matplotlib.axis.XTick at 0x7ce0dc7eae30>,
  <matplotlib.axis.XTick at 0x7ce0dc7eace0>,
  <matplotlib.axis.XTick at 0x7ce0dc7fbcd0>,
  <matplotlib.axis.XTick at 0x7ce0dc80cb20>],
 [Text(0, 0, '0'),
  Text(5, 0, '5'),
  Text(10, 0, '10'),
  Text(15, 0, '15'),
  Text(20, 0, '20')])
```




```
# We create the knn and we train and obtain results
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)))
```

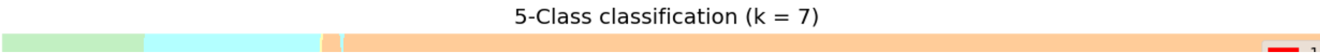
```
Accuracy of K-NN classifier on training set: 0.90
Accuracy of K-NN classifier on test set: 0.86
```

```
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
```

```
#Here we can check the results
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)"
              % (n_neighbors))

plt.show()
```



A k-NN (k-Nearest Neighbors) classifier from the scikit-learn library does not use a loss function or an optimizer in the way that, for example, a neural network would. k-NN classifies points according to the majority of votes of the k nearest neighbors, and there is no iterative optimization process involved as in other learning models.

If you want to add a loss function and an optimizer, you would need to switch to a different model that requires them, such as a neural network.