KNN Classifier Explanation with Pseudo Code & Calculations

In this documentation, KNN explained along with its usage. Pros and Cons described and finally, the calculations explained with the help of pseudo code

1. What is the KNN algorithm?
2. Example for explaining KNN
3. K-Nearest Neighbors and Bias–Variance Tradeoff
4. What are the advantages and disadvantages of KNN?
5. Areas where KNN is used

What is the KNN algorithm?

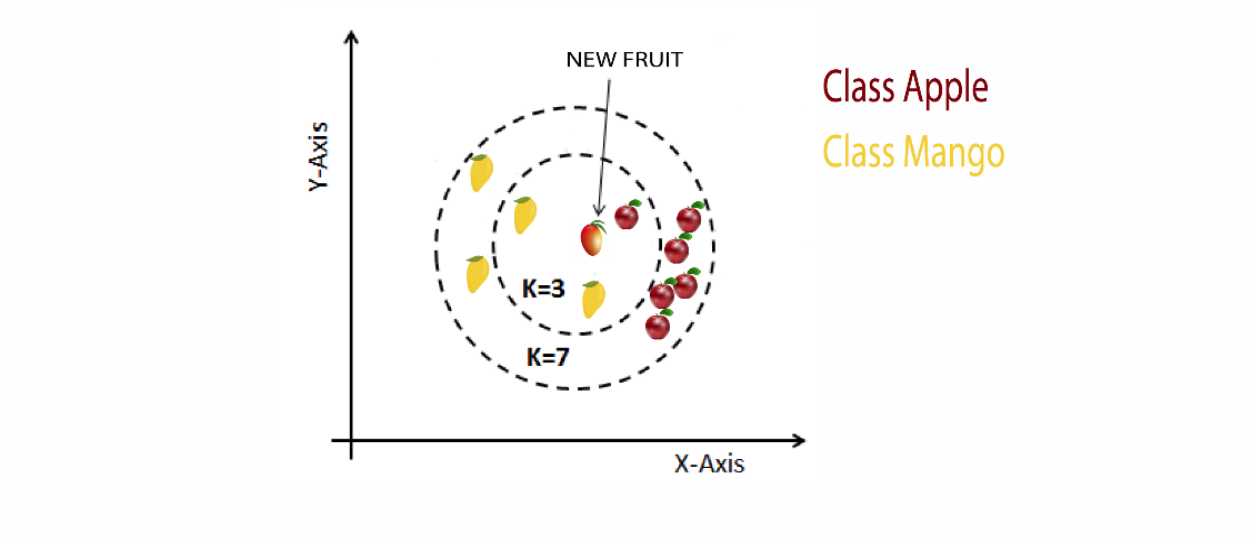
The k-nearest neighbors (KNN) is lazy learner, it is non-parametric and used in both classification and regression problems, but commonly used for classification problems. It classifies the input data into classes that are closest to them by calculating the distance of every dimension of input data from every dimension in given data with classes. Model does not train, no action/calculations are performed on training data, which makes KNN good for data mining.

Any of the distance formulas can be used to find the distance between neighbors, some of the commonly used distance algorithms are:

- Euclidean Distance
- Hamming Distance
- Manhattan Distance
- Minkowski Distance

Example for explaining KNN

There is a basket full of mangos and apples. Each fruit differs from each other based on their features, mangos have more height and their color is yellow, whereas apples height is short and their color is red. KNN algorithm will decide the class of new fruit with respect to its resemblance of features.



K-Nearest Neighbors and Bias–Variance Tradeoff

Model accuracy is perfect when K is equal to one on the same data, but it gets chances of error when new data occurs. Increasing the value of k will cause high bias but low variance. Increase in K means more neighbors. To reduce error rate, change the value of k multiple times to get better accuracy.

What are the advantages and disadvantages of KNN?

Advantages:

- Implementation is straightforward
- Training time is reduced because the data points are classified on the basis of distance from other classes
- It can deal data having numerous classes

Disadvantages:

- Not perform well in high dimensions, because it will have to compute distance for every dimension.
- It deviates from classifying classes if the features are not normalized
- Cost computation is high in large datasets as it computes distances for every centroid
- Accuracy is affected with noise
- Results digress as value of k increase
- The number of classes not to be same as multiples of k
- For the two classes, k should be an odd value

Areas where KNN is used

Knn is often used in these areas:

- From text mining, it classifies unstructured text into appropriate document
- Species identification
- Recommend movies depending on user's feed
- Used in recommending relevant movies to the audience
- Pattern recognition

KNN Explained with Pseudo Code along with Calculations

Algorithm is described with Sport Classification example, It is implemented by Nabeer Dar, reference code in python (knn.py) and dataset.csv files are uploaded here: ***https://github.com/nabeerdar/knn***
Code from the main body from above Github repository is explained with pseudo-code along with calculations and screenshots of every output is described below:

Step 1: Read training data

```
dataset = read_dataset('dataset.csv')
```

```
          Name  Age  Gender Class of Sport
0        Goku   32       0       Football
1  James Bond   40       0        Neither
2  Nabeer Dar   16       1        Cricket
3       Mulan   34       1        Cricket
4     Pikachu   55       0        Neither
5      Ben 10   40       0        Cricket
6        Luna   20       1        Neither
```

Step 2: Set new data points

1. Let first point be 45 representing age
2. Let second point be 0 (male) representing gender

Step 3: Input size of K

```
k = int(input("Enter value of K: "))
```

```
Enter value of K: 3
```

Step 4: Compute distance of new data points from respective dimensions, here euclidean distance formula is used:

$$d(p, q) = d(q, p) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2}$$

$$= \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

```
calculated_distance = calculate_distance(data_tuple, new_point_x, new_point_y, dataset)
```

```
Calculated Distances

distance = ((45 - 32) ** 2) + ((0 - 0) ** 2) ** 0.5 = 13.0
distance = ((45 - 40) ** 2) + ((0 - 0) ** 2) ** 0.5 = 5.0
distance = ((45 - 16) ** 2) + ((0 - 1) ** 2) ** 0.5 = 29.017236257093817
distance = ((45 - 34) ** 2) + ((0 - 1) ** 2) ** 0.5 = 11.045361017187261
distance = ((45 - 55) ** 2) + ((0 - 0) ** 2) ** 0.5 = 10.0
distance = ((45 - 40) ** 2) + ((0 - 0) ** 2) ** 0.5 = 5.0
distance = ((45 - 20) ** 2) + ((0 - 1) ** 2) ** 0.5 = 25.019992006393608
```

Step 5: Assigning calculated distances to given dataset

```
dataset_with_distances = updated_disances_df(calculated_distance, dataset)
```

```
dataset with calculated distances from new data points

          Name  Age  Gender Class of Sport  distances
0        Goku   32       0       Football  13.000000
1  James Bond   40       0        Neither   5.000000
2  Nabeer Dar   16       1        Cricket  29.017236
3       Mulan   34       1        Cricket  11.045361
4     Pikachu   55       0        Neither  10.000000
5      Ben 10   40       0        Cricket   5.000000
6        Luna   20       1        Neither  25.019992
```

Step 6: Select top Get index of k largest distances from dataset_with_distances

```
top_k_distances = select_top_k_distances(dataset_with_distances, k)
```

```
Top K Distances
 2    29.017236
6    25.019992
0    13.000000
```

Step 7: Calculate index count of each class of sport with respect to indexes returned from top_k_distances

```
class_wise_index_count = calculate_class_wise_index_count(top_k_distances, dataset_with_distances)
```

```
classes count:  Counter({'Cricket': 1, 'Neither': 1, 'Football': 1})
```

Step 8: Check the class with maximum indexes, if count of all the indexes within class is same, choose any of them

```
most_similar_class = check_most_similar_class(class_wise_index_count)
```

```
This new data belongs to class:  Cricket
```