KNN

August 28, 2018

Importing required Packages and loading the training and testing data

```
In [1]: from scipy.io import arff
        import pandas as pd
        import math
        import operator
        import matplotlib.pyplot as plt

        train_data = arff.loadarff('trainProdSelection.arff')
        training_set = pd.DataFrame(train_data[0])

        test_data = arff.loadarff('testProdSelection.arff')
        testing_set = pd.DataFrame(test_data[0])
```

Printing the training data

```
In [2]: training_set.head()
```

```
Out[2]:          Type        LifeStyle  Vacation  eCredit  salary  property  label
        0  b'student'  b'spend>saving'       6.0     40.0   13.62    3.2804  b'C1'
        1  b'student'  b'spend>saving'      11.0     21.0   15.32    2.0232  b'C1'
        2  b'student'  b'spend>saving'       7.0     64.0   16.55    3.1202  b'C1'
        3  b'student'  b'spend>saving'       3.0     47.0   15.71    3.4022  b'C1'
        4  b'student'  b'spend>saving'      15.0     10.0   16.96    2.2825  b'C1'
```

Printing the testing data

```
In [3]: testing_set.head()
```

```
Out[3]:            Type          LifeStyle  Vacation  eCredit   salary  property  label
        0    b'student'    b'spend<saving'      12.0     19.0  14.7900    3.7697  b'C1'
        1    b'student'   b'spend>>saving'      29.0     10.0  16.1900    2.4839  b'C1'
        2    b'student'   b'spend<<saving'      28.0     60.0  15.4600    1.1885  b'C1'
        3   b'engineer'    b'spend>saving'      15.0     41.0  21.2600    1.4379  b'C1'
        4  b'librarian'    b'spend<saving'       2.0      9.0  19.7207    0.6913  b'C1'
```

Checking the datatype for every column

```
In [4]: pd.DataFrame(train_data[0]).dtypes
```

```
Out[4]: Type           object
        LifeStyle      object
        Vacation      float64
        eCredit       float64
        salary        float64
        property      float64
        label          object
        dtype: object
```

Training set pre-processing

```
In [5]: training_set.Type = training_set.Type.str.decode("UTF-8")
        training_set.LifeStyle = training_set.LifeStyle.str.decode("UTF-8")
        training_set.label = training_set.label.str.decode("UTF-8")
```

```
In [6]: minValue = training_set.Vacation.min()
        maxValue = training_set.Vacation.max()
        training_set.Vacation = training_set.Vacation.apply(lambda x:(x-minValue)/(maxValue-minV

        minValue = training_set.eCredit.min()
        maxValue = training_set.eCredit.max()
        training_set.eCredit = training_set.eCredit.apply(lambda x:(x-minValue)/(maxValue-minVal

        minValue = training_set.salary.min()
        maxValue = training_set.salary.max()
        training_set.salary = training_set.salary.apply(lambda x:(x-minValue)/(maxValue-minValue

        minValue = training_set.property.min()
        maxValue = training_set.property.max()
        training_set.property = training_set.property.apply(lambda x:(x-minValue)/(maxValue-minV
```

Training set pre-processing done
Testing set pre-processing

```
In [7]: testing_set.Type=testing_set.Type.str.decode("UTF-8")
        testing_set.LifeStyle=testing_set.LifeStyle.str.decode("UTF-8")
        testing_set.label=testing_set.label.str.decode("UTF-8")
```

```
In [8]: minValue = testing_set.Vacation.min()
        maxValue = testing_set.Vacation.max()
        testing_set.Vacation = testing_set.Vacation.apply(lambda x:(x-minValue)/(maxValue-minVal

        minValue = testing_set.eCredit.min()
        maxValue = testing_set.eCredit.max()
        testing_set.eCredit = testing_set.eCredit.apply(lambda x:(x-minValue)/(maxValue-minValue

        minValue = testing_set.salary.min()
        maxValue = testing_set.salary.max()
        testing_set.salary = testing_set.salary.apply(lambda x:(x-minValue)/(maxValue-minValue))
        minValue = testing_set.property.min()
        maxValue = testing_set.property.max()
        testing_set.property = testing_set.property.apply(lambda x:(x-minValue)/(maxValue-minVal
```

Testing set pre-processing done
KNN function

```
In [9]: def knn(k):
            predictions=[]
            for x in range(len(testing_set)):
                neighbors = getNeighbors(training_set.values, testing_set.values[x], k)
                result = getResponse(neighbors)
                predictions.append(result)
            accuracy = getAccuracy(testing_set.values, predictions)
            return repr(accuracy)
```

```
In [10]: def euclideanDistance(instance1, instance2, length):
             distance = 0
             for i in range(2):
                 if (instance1[i]==instance2[i]):
                     distance += 1
             for x in range(2,length):
                 distance += pow((instance1[x] - instance2[x]), 2)
             return math.sqrt(distance)

         def getNeighbors(trainingSet, testInstance, k):
             distances = []
             length = len(testInstance)-1
             for x in range(len(trainingSet)):
                 dist = euclideanDistance(testInstance, trainingSet[x], length)
                 distances.append((trainingSet[x], dist))
             distances.sort(key=operator.itemgetter(1))
             neighbors = []
             for x in range(k):
                 neighbors.append(distances[x][0])
             return neighbors

         def getResponse(neighbors):
             classVotes = {}
             for x in range(len(neighbors)):
                 response = neighbors[x][-1]
                 if response in classVotes:
                     classVotes[response] += 1
                 else:
                     classVotes[response] = 1
             sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1), reverse=True)
             return sortedVotes[0][0]
```

```
def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0
```

Storing the predictions to a dictionary

```
In [11]: KNN={}
         for i in range(1, 100, 2):
             KNN[i]=float(knn(i))
         print(KNN)
```

```
{1: 19.047619047619047, 3: 23.809523809523807, 5: 19.047619047619047, 7: 14.285714285714285, 9:
```

Finding the more accurate value

```
In [12]: MAX=0
         for i in KNN:
             if(float(KNN[i])>=MAX):
                 MAX=float(KNN[i])
         print(MAX)
```

```
28.57142857142857
```

Adding dictionary keys and value to different lists

```
In [13]: k_values=[]
         Accuracy_list=[]
         for i in KNN:
             k_values.append(i)
             Accuracy_list.append(float(KNN[i]))
         print(k_values)
```

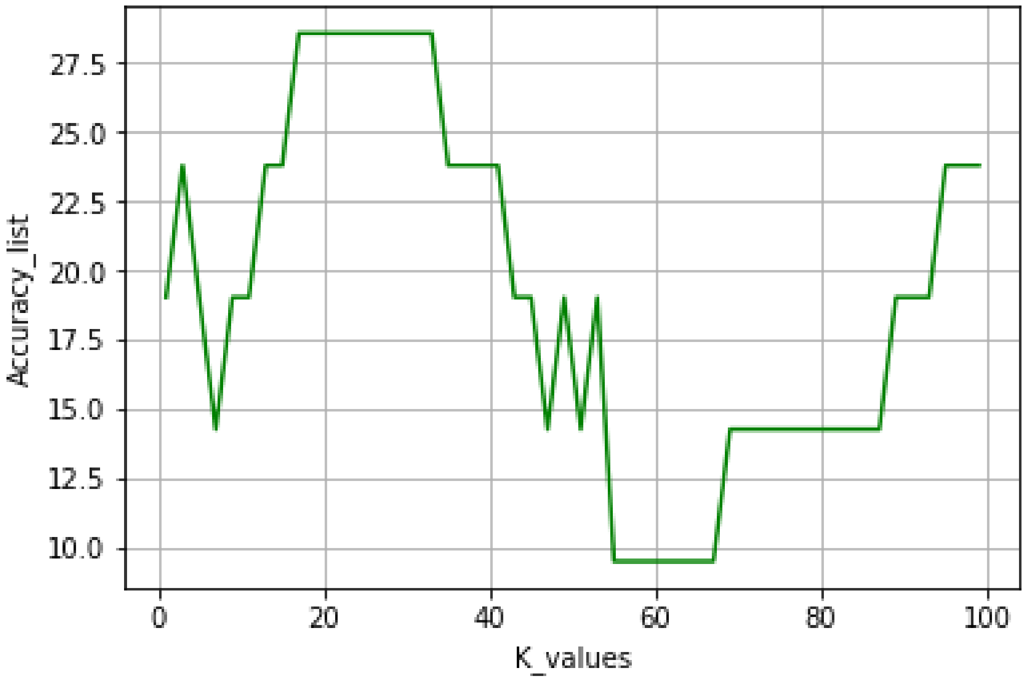
```
[1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35, 37, 39, 41, 43, 45, 47, 49,
```

```
In [14]: print(Accuracy_list)
```

```
[19.047619047619047, 23.809523809523807, 19.047619047619047, 14.285714285714285, 19.047619047619
```

Ploting a graph between k_values and respective Accuracy

```
In [15]: plt.plot(k_values,Accuracy_list,color='green')
         plt.xlabel("K_values")
         plt.ylabel("Accuracy_list")
         plt.grid(True)
         plt.show()
```


27.5
25.0
22.5
20.0
17.5
15.0
12.5
10.0
Accuracy_list
0
20
40
60
80
100
K_values