Project Report

Mahmoud Akl
03639631
mahmoud.akl@gmail.com

Letter Image Recognition Using KNN Algorithm

1 Dataset:

The dataset for this project was provided from your side and can be downloaded from here. The dataset consists of 20000 samples and it was stated, that the first 16000 samples should be considered as training data and the remaining 4000 samples should be considered as test data.

Each sample consists of 17 features:
1. Capital Letter
2. Horizontal Position of box
3. Vertical Position of box
4. Width of box
5. Height of box
6. Total # on pixels
7. Mean x of on pixels in box
8. Mean y of on pixels in box
9. Mean x variance
10. Mean y variance
11. Mean x y correlation
12. Mean of x * x * y
13. Mean of x * y * y
14. Mean edge count left to right
15. Correlation of mean edge count left to right with y
16. Mean edge count bottom to top
17. Correlation of mean edge count bottom to top with x

Sometimes, before starting to work with the dataset, feature scaling may be required. In this case it was not, since it was already done, i.e. the features in the dataset are scaled to fit into a range of integer values from 0 to 15.

2 Algorithm:

The algorithm I implemented for this classification problem was the KNN algorithm. The K-Nearest Neighbors algorithm is fairly simple; the input consists of the k closest training examples in feature space and the output is a class membership. In other words, an object is classified by the majority vote of its neighbors.
In order to know which training examples are closest to the input sample, a distance metric is required. A commonly used distance metric is the Euclidean Distance:

$$\sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \qquad \textit{,where n is the number of features}$$

In KNN, the choice of k affects the accuracy of the classification results and the best choice of k depends on the dataset. In my implementation I tested the classification for various choices of k. I chose k to be the odd numbers in the range $[1, 21]$. The choice of odd numbers was to avoid tied votes.

The implementation was done with Python 3, and the implementation code will be presented at the end of this document.

3 Results

As mentioned before, the classification was tested for different choices of k; $k = [1, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21]$. Figure 1 shows the results for each selected value of k. The histogram shows the number of correctly classified samples for each choice of k. The total number of samples was 4000. Figure 2 represents the accuracy in percentage of the classification for each selected k

$$\textbf{accuracy} = \frac{\textbf{correctly classified samples}}{\textbf{total number of samples}}$$

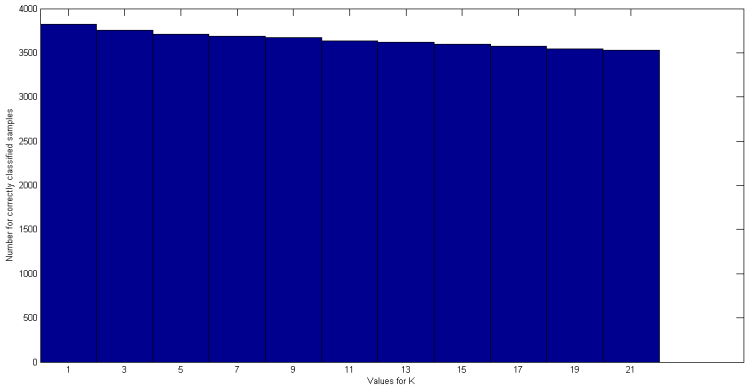

Figure 1: Number of correctly classified samples for different values of k.

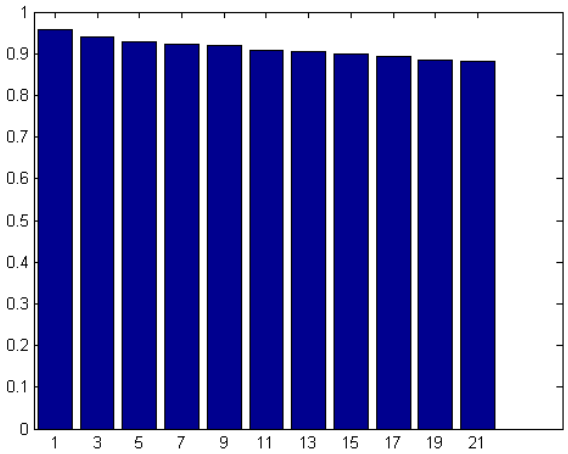

Figure 2: Accuracy in percentage for each selected value for k.

It is observed that the higher the value of k, the less accurate the classification is, i.e the less correctly classified samples exist. The highest accuracy recorded was for $k = 1$, which provided a classification accuracy of 95.65% is known as the Nearest Neighbor Classifier, where the output depends only on the closest example to it. Normally, Nearest Neighbor Classifier is prone to false classifications because of outliers, but this was not the case here. In figure 3, the histogram shows the classification error for each value of k in percentage.

Intuitively, the values of k that provided higher accuracy will provide lower error.

$$\textbf{error} = 1 - \textbf{accuracy}$$

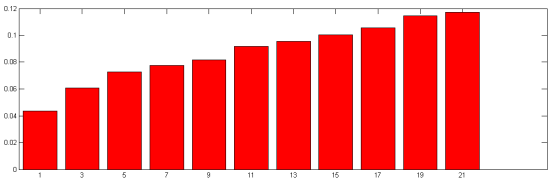

Figure 3: Classification Error in percentage for each value of k.

4 Future Work:

Further work that could be done to improve this project would be testing the KNN classifier using different distance measurements. Other distance measures for continuous variables include the Manhattan distance and the Minkowski distance. Another factor to consider in the future would be feature weighting.

Another point that I only noticed later, when it was too late to change is my choice of odd numbers for k. This was always the case whenever I studied or dealt with the KNN algorithm. Now that I think about it, this should only be the case in binary classification, when there are only two possible classes available. In the case of multi class classification (this case), tied votes can be produced for an odd or an even number of k.

5 Code

```
import numpy as np
import time
import math
from collections import Counter

start = time.time()
f = open("letter-recognition.data","r")
linelist = f.readlines()
data = []
for line in linelist:
    data.append(line.split(','))
training_data = data[0:16000]
test_data = data[16000:20000]
small_data = test_data[0:20]
prediction_data = [item[1:17] for item in test_data]
correct_classes = [item[0] for item in test_data]
small_pred_data = [item[1:17] for item in small_data]
small_classes = [item[0] for item in small_data]
correct = []
wrong = []
for k in range(1,22,2):
    letters = [None]*k
    distance = []
    predicted_classes = []
    correct_classified = 0
    wrong_classified = 0
    step = 0
    for pred in prediction_data:
        pred = list(map(int, pred))
        j = 0
        distance = []
        for train in training_data:
            train = list(map(int,train[1:17]))
            diff = [(a-b)**2 for a,b in zip(pred,train)]
            distance.append(math.sqrt(sum(diff)))
        distance_sorted = sorted(distance)
        for count in range(k):
            dist = distance_sorted[count]
            index = distance.index(dist)
            letters[count] = training_data[index][0]
        l = Counter(letters)
        mc = l.most_common()
        predicted_classes.append(mc[0][0])
        step = step+1
    for i,j in zip(predicted_classes,correct_classes):
        if i==j:
            correct_classified += 1
        else:
            wrong_classified   += 1
    correct.append(correct_classified)
    wrong.append(wrong_classified)
    print (k)
end = time.time()
print (end-start)
```