Ahmad Al-waili 33497588 aalw001@gold.ac.uk

Machine Learning, 2018-19
Coursework Part I
Report

1. Write-up of implementation

- Nested Cross Validation:
  - The basic implementation of the Nested Cross Validation goes with the lecture slides. So, the NCV splits the data set into k number of bins, let's take the example of 5-bins, and then uses one of the bins as the Test Fold, the other 4 folds as Training Fold, and then one of those which is 1 index away from the Test fold as Validation Fold. Shown in figure 1 below.
    Pseudo code:

```
def NCV(X, Y, kfold, n, dist):
      test, train, val = []
      for i in length(kfold):
            bins[i] = test
            for j in length(kfold):
                  if i == j:
                        bins[j] = train
            if (i+1) > length(kfold):
                  bins[0] = val
            else:
                  bins[i+1] = val
            //go through all parameters of kfold and dists

            for k in length(n):
                  for l in length(dists):
                        a = knnFunc(X,Y,n[k],dists[l]
                        //take best accuracy
                        test.append(a)
                  best = max(test)
```

Figure 1: pseudo code of ncv

- kNN Function:
  - The kNN function was a little harder to implement as I had to really understand how the nearest neighbour actually gets the nearest neighbour from a matrices perspective, as from a graph or visual point of view its really easy to understand, but relaying that information into the code was a challenge. The concept of kNN has to be split into different functions such as *getNeighbours* and also *assignLabel,* shown in the labs. These two functions allow us to gather the neighbours that we need of a test set that is passed through the parameters and assignLabels allows us to get the most frequent in that list, which is the most frequent neighbour.

```
def kNN(X, Y, test, n, distance):
      final = []
      for i in length(test):
            test[i] = x
            neighbours = getNeighbours(x, X, n, size(X),
distance)
            y_values = Y[neighbours]
            final.append(assignLabel(y_values))
      return final

def getNeighbours(x, test, nns, size, dist):
      dist = []
      for i in length(size):
            if dist == 'manhattan':
                  f = manhattanDistance(x, test[i])
                  dist.append(f)
            else:
                  f = euclideanDistance(x, test[i])
                  dist.append(f)

         dist = sort(dist)

         return dist[0 to nns]
```

Figure 2: knn

- Distance functions:
  - So, both distance were functions were calculated and used in the value of kNN as shown above in figure 2, the parameter of *distance* allows it to go through the if statement and return the value of sorted. However in figure 1 you can see with the NCV the best distance value is taken based upon the accuracy that is returned, meaning, that if Euclidian returned a larger number than Manhattan than Euclidean would be used in the final folds of the NCV. The two distance functions are shown below:

```
def euclideanDistance(in1,in2):
    return norm(in1-in2)
```

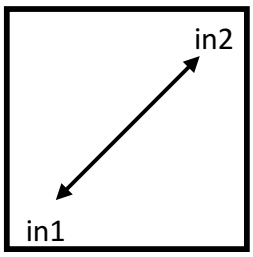

Figure 3: euclidean distance visualised

```
def manhattanDistance(in1, in2):
    return absolute(in1 - in2).sum(-1)
```

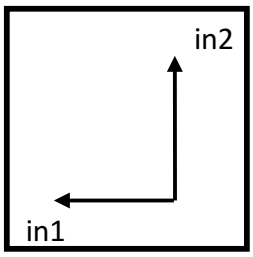

Figure 4: Manhattan Distance visualised

Ahmad Al-waili 33497588 aalw001@gold.ac.uk

2. Results

Clean data	Accuracy	k-neighbours	Distance	Confusion Matrix
Fold-1	96	3	euclidean	[11 0 0] [0 10 1] [0 0 8]
Fold-2	93	4	euclidean	[12 0 0] [0 11 0] [0 2 5]
Fold-3	93	1	euclidean	[5 0 0] [0 13 1] [0 1 10]
Fold-4	100	4	euclidean	[11 0 0] [0 7 0] [0 0 12]
Fold-5	96	1	euclidean	[11 0 0] [0 6 1] [0 0 12]
Total of Clean	96			

Noisy data	Accuracy	k-neighbours	Distance	Confusion Matrix
Fold-1	73	1	euclidean	[11 0 0] [0 7 4] [0 4 4]
Fold-2	83	9	euclidean	[12 0 0] [0 9 2] [0 3 4]
Fold-3	80	2	euclidean	[5 0 0] [0 11 3] [0 3 8]
Fold-4	100	5	euclidean	[11 0 0] [0 7 0] [0 0 12]
Fold-5	86	8	euclidean	[11 0 0] [0 7 0] [0 4 8]
Total of noisy	84			

Ahmad Al-waili 33497588 aalw001@gold.ac.uk

Total Confusion matrix:

Clean			Noisy		
50	0	0	50	0	0
0	47	3	0	41	9
0	3	47	0	14	36

1. Do the best parameters change per fold?
    a. Yes, the NVC takes all the parameters into consideration adds it into another array for testing and tests based upon accuracy, and then the best N and best Distance is taken.
2. Can you say that one parameter choice is better regardless of the data used?
    a. I cannot say that, as you can see per fold the value of N changes depending on the bin itself, which is the training data. So, in conclusion the parameters have to be taking into consideration for the accuracy.

       However, we could argue the case that for noisy data, you will need a higher value of K, as the accuracies which are highest shown in the results table show K as 9, 5 and 8, which are all higher than all the values of K in the clean data.
3. Does the best parameter choice change depending on whether we use clean or noisy data? (Answer for both distance function and number of neighbours.)
    a. Yes, it does. As mentioned above, you can see that the noisy data needs a higher value of K, and also has a higher chance of the value of K being odd, to avoid any tie breaking within the choosing of a neighbour. However, we could see that the value of the best distance function used is *always* Euclidean, as you can see in figure 3 and figure 4, the distance function of Euclidean is way more accurate and shows the true distance from the origin or the two points that are tested against.

3. Questions

a) **Exploratory Data Analysis.** What do you observe by plotting the figure for data without noise? What do you observe when you add Gaussian noise and plot again?

a. Without noise the data is together and seems linear with the plot, as shown in figure 1. Where as you can see in figure 2, you can see that the data is more all over the place and doesn't seem to show a linear growth. Furthermore, as you can see in the noisy data it is **much** harder to add a line of best fit for the data of the 3 different colours, and even if we do add the line of best fit, there is a possibility it could look similar to the clean data, giving the wrong impression of the data, if we did not have the full graph shown like below.

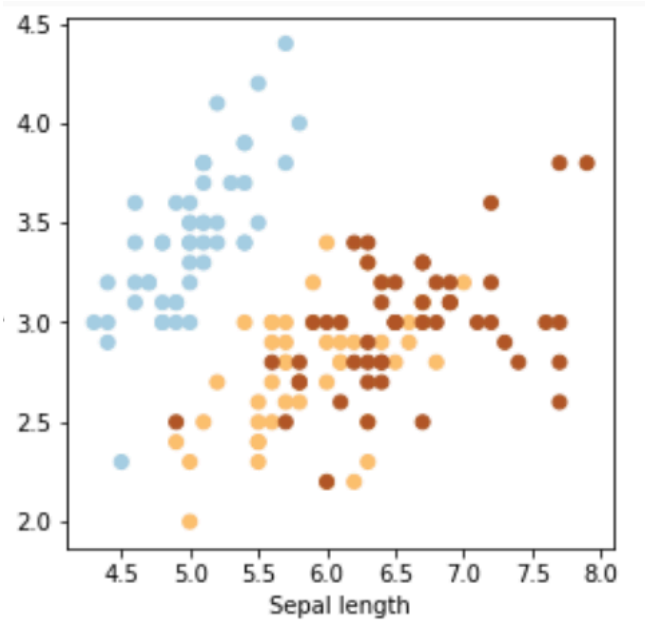

Figure 6: clean data

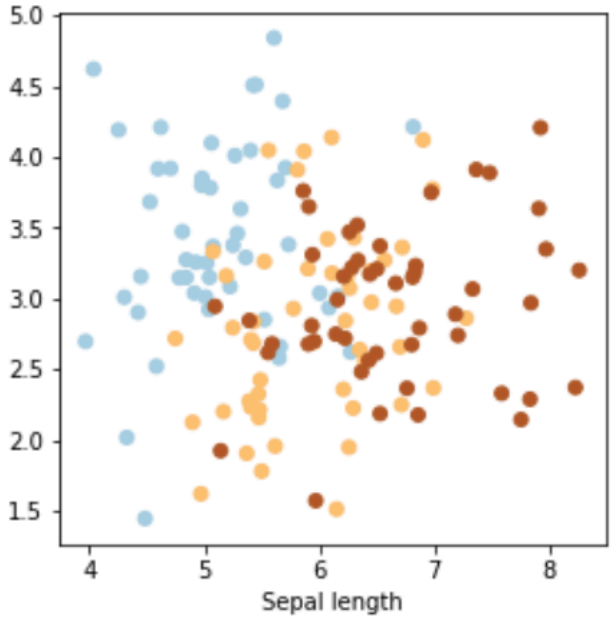

Figure 5: noisy data

b) **Tie breaking.** Assume that you have selected the number of neighbours to be an even number, e.g., 2. For one of the neighbours, the suggested class is 1, and for the other neighbour the suggested class is 2. How would you break the tie? Write example pseudocode that does this.

a. To break a tie within the classification in kNN, I would:

i. Increase the number of k to an even or an odd number
ii. choose the most frequent occurring neighbour in the dataset
iii. do the same with class 2
iv. randomly select between the neighbours that have occurred in the most frequent datasets.

```
def tieBreak(X, test, n):
        final = []
        if n is even:
              n ++
        for i in length(X):
              g = getNeighbours(X, test[i])
              final.append(assignLabel(g))
        r = random(0, length(final))
        return final[r]
```

c) **Improving performance on noisy data.** The performance of k-NN on the noisy data should be worse than on the clean data.

Suggest at least one way of improving the performance on the noisy data.

a. Of course one way to improve the value of kNN is not choose an even number when using K, this will therefore allow no ties in neighbours.

Another way to improve performance in noisy data, is to use the Euclidian distance rather than the Manhattan distance, as of course the Euclidian algorithm is more

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

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

accurate and shows the true distance using Pythagoras, in multi-dimensional sets of data, where;