CSC2515: Bonus 1

Due on Monday, February 5, 2018

Matthew Wong, 998803932

February 5, 2018

Problem 0

Introduction and Readme

Attached, you will find the *knn.py* and associated Latex files for the write-up. Code was written in Python 3.5 with the relevant numpy, ski-kit learn and matplotlib libraries. You can execute the code by simply running python knn.py in a command line - the images will be saved to the current local directory.

Problem 1

kNN Investigation

The purpose of this report is to outline instances where the performance of the kNN algorithm *increases* with increasing k. To highlight such an instance, three datasets were created using a normal random distribution. Out of the 3 instances, 2 were centered at the same mean, μ while the third was centered much farther away. Of the two centered at the same μ, the difference in standard deviations were large. Figure 1 is an illustration of the distribution of data.

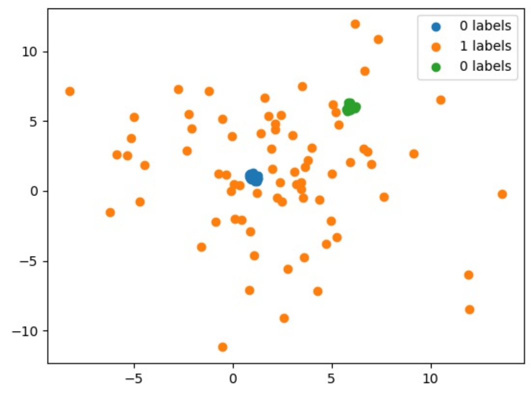

Figure 1: Randomly generated datasets, based on normal distributions

For the 0 labels, a small number of 0's is located near the 1 labels - the intuition here is that at low k values, performance would drop. At higher k values, with a majority voting scheme, the "far-away" 0 labels would have more weight in the decision process and since they are so close to each other (e.g. small μ) they would be considered over the 1-labelled points. This is indeed the case as shown in Figure 2 as there were two increases - one at $k \approx 50$ and the other at $k \approx 70$. Code for this experiment is located on the following page.

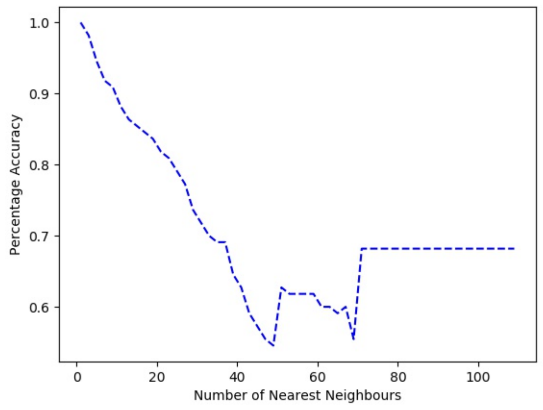

Figure 2: Percentage Accuracy of dataset vs. # of nearest neighbours

```python
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
import numpy as np
import matplotlib.pyplot as plt
#Set up seed and datasets
np.random.seed(1)
x1 = np.random.normal(loc = 1.0, scale = 0.15,size=(25,2))
x2 = np.random.normal(loc = 1.0, scale = 5.0,size=(75,2))
x3 = np.random.normal(loc = 6.0, scale = 0.15,size = (10,2))
#Plot and save the data
plt.scatter(x1[:,0],x1[:,1],label= "0 labels")
plt.scatter(x2[:,0],x2[:,1],label = "1 labels")
plt.scatter(x3[:,0],x3[:,1],label = "0 labels")
plt.legend()
plt.savefig('Data_plot.jpg')
plt.close()
#Stacking the matrices for X
x = np.vstack((x1,x2))
x = np.vstack((x,x3))
#Generate corresponding labels for clusters
y1 = np.zeros((25))
y2 = np.ones((75))
y3 = np.zeros((10))
#Stacking the labels
y = np.hstack((y1,y2))
y = np.hstack((y,y3))
#Stacking the matrices together
y = y[:,np.newaxis]
complete = np.hstack((x,y))
#Shuffle the matrix
np.random.shuffle(complete)
#Get the labels
labels = complete[:,-1]
#Get the data
complete = complete[:,:-1]
#KNN implementation
k_values, accuracy = [],[]
i = 1
while i < complete.shape[0]:
   knn = KNeighborsClassifier(n_neighbors=i)
   knn.fit(complete, labels)
   y_pred = knn.predict(complete)
   k_values.append(i)
   accuracy.append((accuracy_score(labels,y_pred)))
   i+=2
#Plotting the data
x_axis = k_values
y_axis = accuracy
plt.xlabel("Number of Nearest Neighbours")
plt.ylabel("Percentage Accuracy")
plt.plot(x_axis, y_axis, 'b--')
plt.savefig('Performance_Chart.jpg')
```

To test if this was just a product of noise, random seeds were selected:

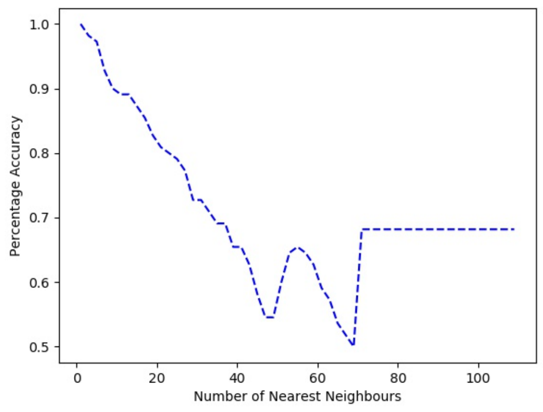

Figure 3: Percentage Accuracy of dataset vs. # of nearest neighbours, seed = 5

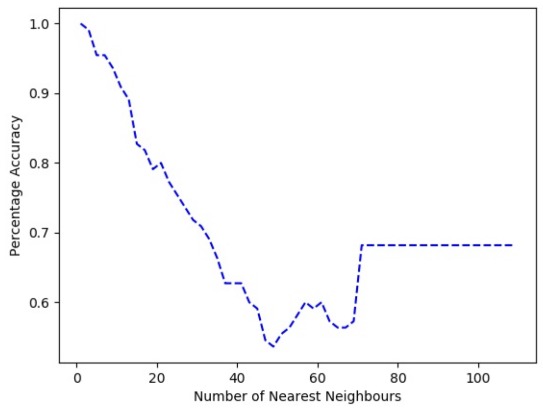

Figure 4: Percentage Accuracy of dataset vs. # of nearest neighbours, seed = 10

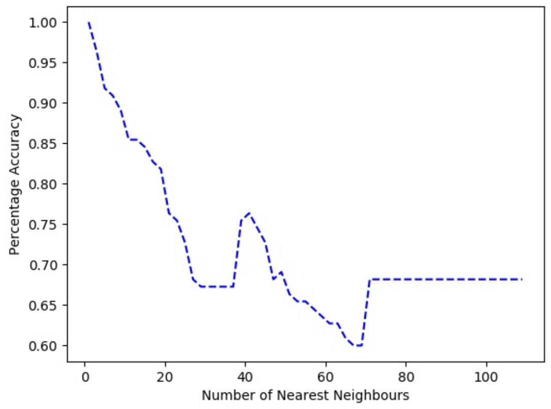

Figure 5: Percentage Accuracy of dataset vs. # of nearest neighbours, seed = 50

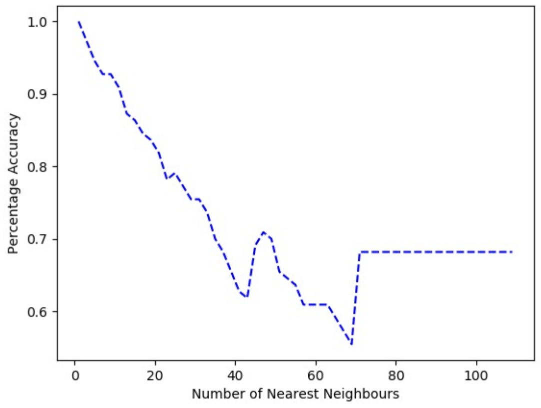

Figure 6: Percentage Accuracy of dataset vs. # of nearest neighbours, seed = 100