CC484: Pattern Recognition

Assignment 1
Face Recognition

Raghad Abo El Eneen	6360
Heba Hassan Elwazzan	6521
Nourhan Waleed	6609

Links:

Steps 1-6:
https://colab.research.google.com/drive/1ZbNnV_EyLkPzuWcMgMH3cDkYOyzUbD4j?usp=sharing
Step 7:
https://colab.research.google.com/drive/1HUbLYO0u3_Tk3NmknUVyM7AWVk4UyKYQ?usp=sharing
Bonus:
https://colab.research.google.com/drive/1FeCBx8zu2T-xPNDvMZ9CGTjEsv12D78g?usp=sharing

1. Download the Dataset and Understand the Format (10 Points)

```
from google.colab import files
uploaded = files.upload()
!unzip archive.zip -d images/
```

This imported the zip archive and unzipped it into Google Colab's directory

2. Generate the Data Matrix and the Label vector (10 Points)

We looped over every image in the subdirectories and flattened each image until we had a 400x10304 matrix (10 images per each of the 40 classes, each having 10304 dimensions).

Labels for each of the 40 subjects were also generated.

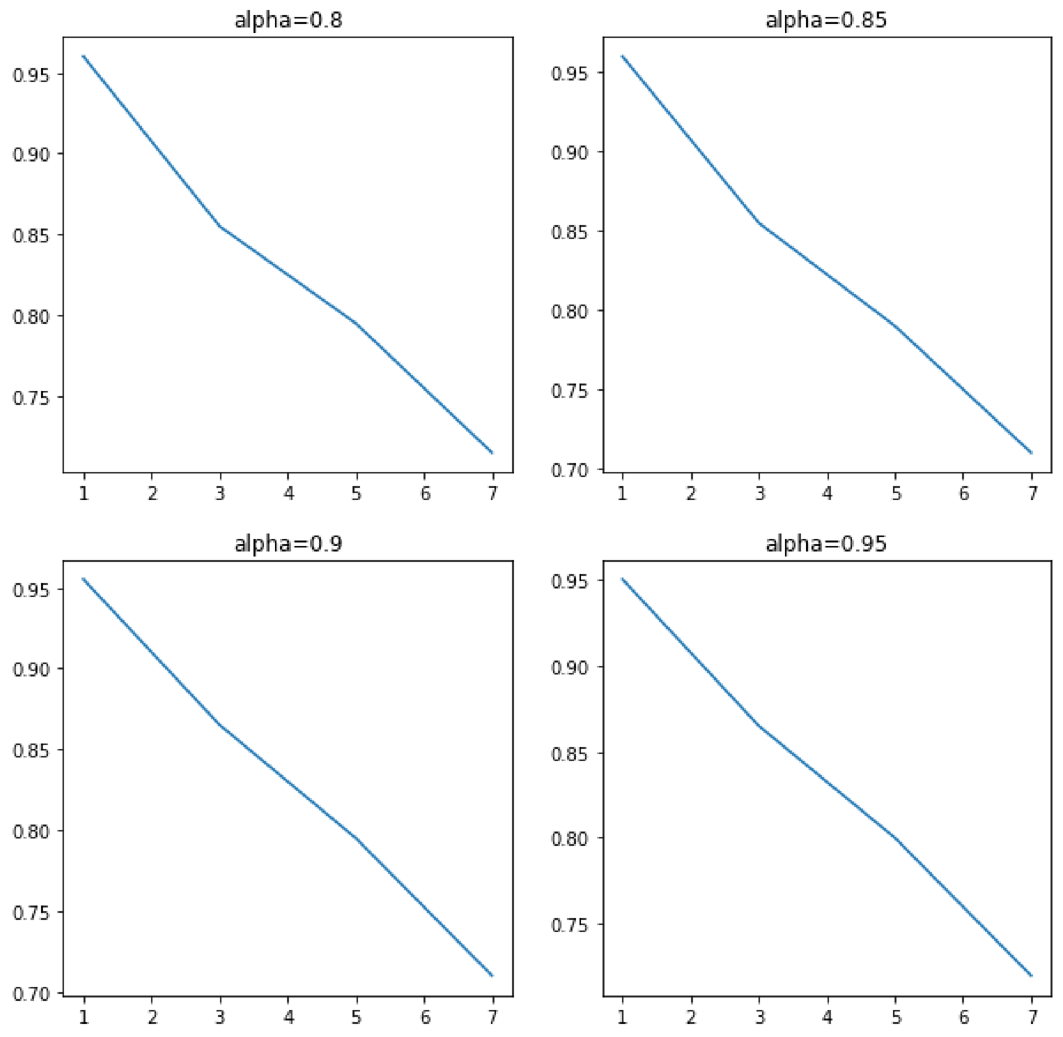
3. Split the Dataset into Training and Test sets (10 Points)

For a 50/50 split:

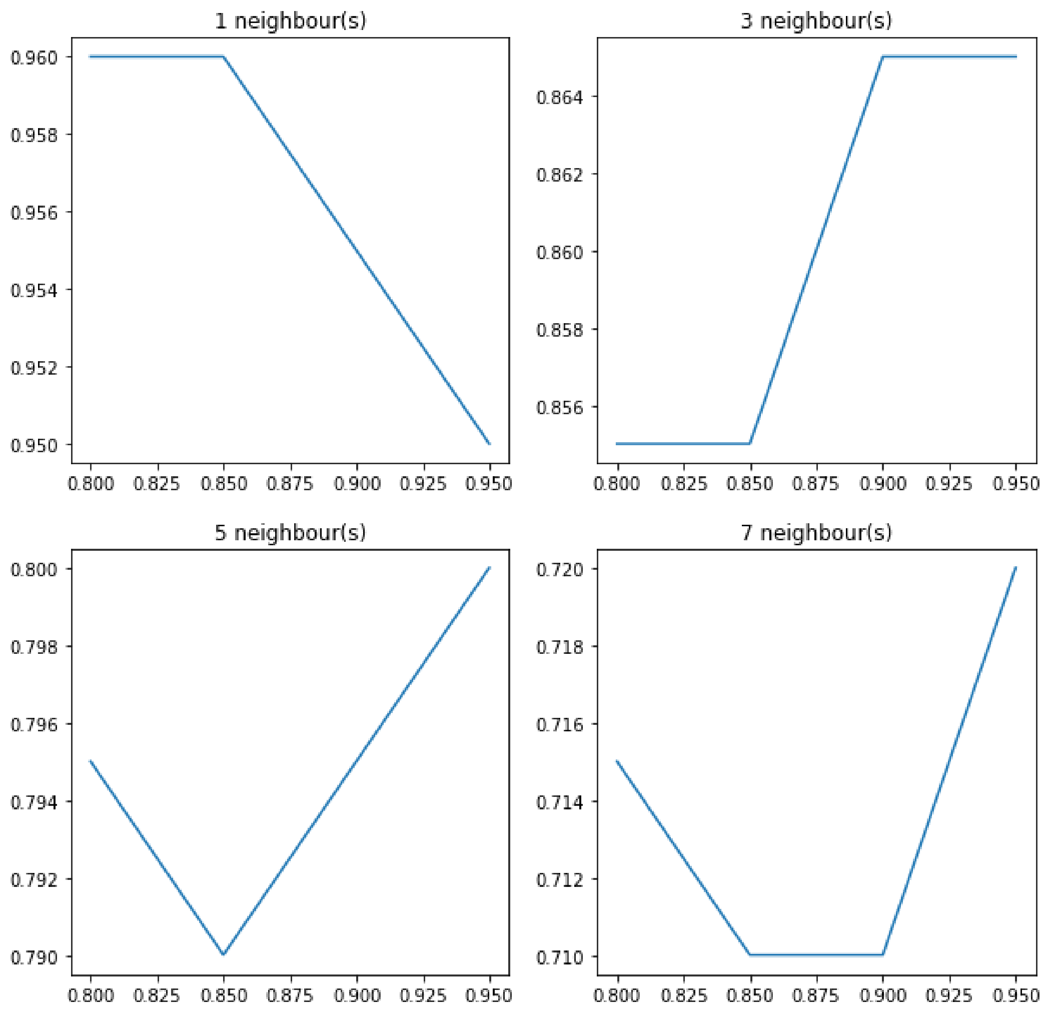
```
for i in range(0,400):
    if(i%2 ==  0 ):
        test_data.append(image_vectors[i])
        test_labels.append(labels[i])

    else:
        train_data.append(image_vectors[i])
        train_labels.append(labels[i])
```

4. Classification using PCA (30 points)

The plots below show k values (x-axis) plotted against the accuracies from PCA and KNN for each value of alpha:

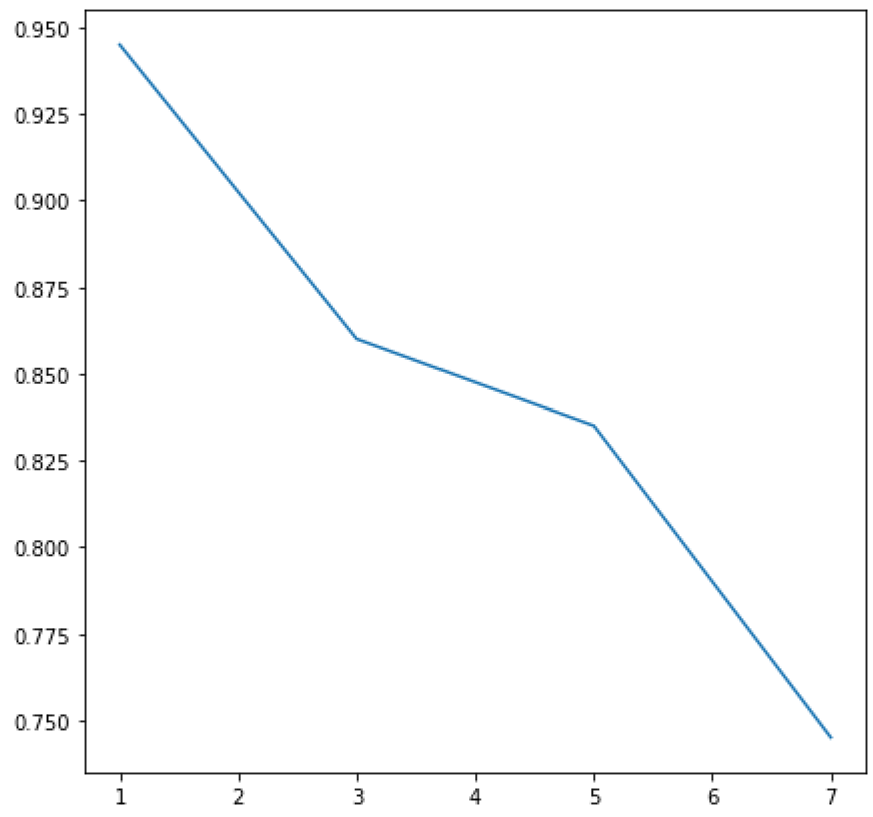


The plots below show the relationship between different alphas (x-axis) against the accuracies for each value of k:



5. Classification Using LDA (30 Points)

Using LDA, accuracies vs number of neighbours is as follows:



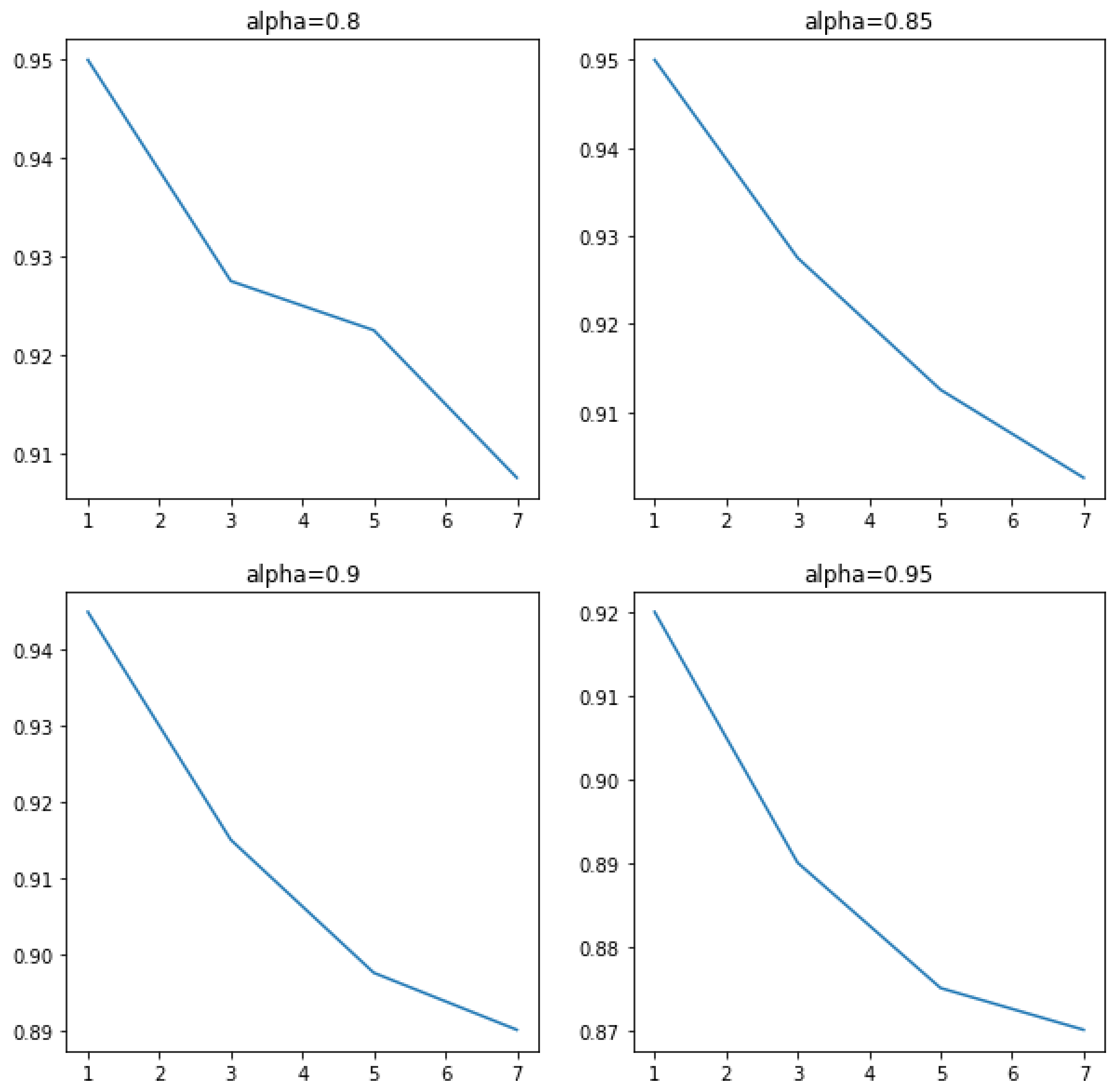
LDA with one neighbour reported an accuracy of 0.945, the highest for LDA. For the PCA, the highest accuracies were reported at also one neighbour at 0.96 for alpha values 0.8 and 0.85 (34 and 50 dimensions respectively).

6. Classifier Tuning (20 Points)

K=1 reported the best accuracies for both PCA and LDA. According to the lecture, we should not use the test data to tune the K hyperparameter, but we were not aware of how else we should go about it.

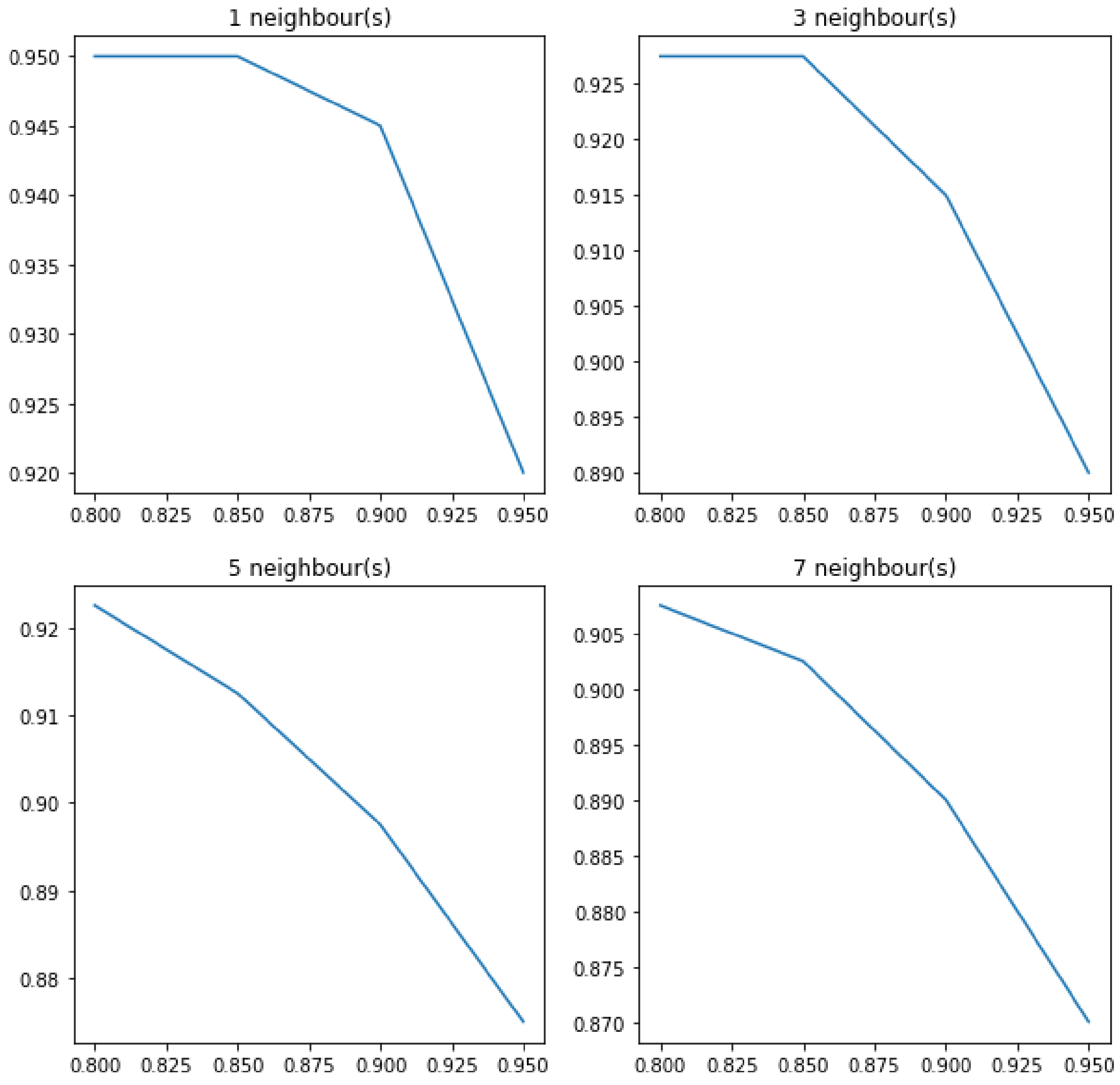
7. Compare vs Non-Face Images (15 Points)

Accuracy Vs KNN



Accuracy decreases as number of neighbours increases

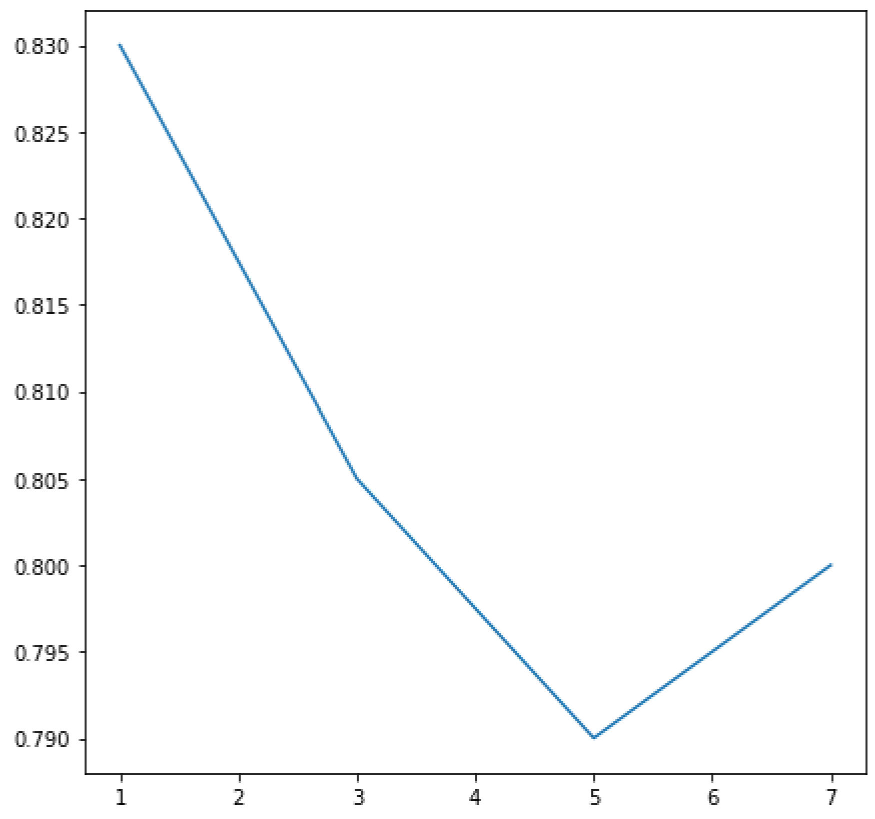
Accuracy Vs Alpha



Accuracy decreases as alpha increases

When we used 1 eigenvector, accuracy was 50%, but when we used 39, accuracy was around 70 or 80%

LDA Graph:



Output:

```
1 nearest neighbours
Correct labels are:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Predicted labels are:
[0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0 0 0 0 0 1 0 0 0 0
 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 1 1 0 0 1 1 1 0
 1 0 0 1 1 0 0 1 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1
 1 0 1 0 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1 1 1 1
 1 1 1 0 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1
 1 1 0 0 1 1 0 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 1 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 1 1 1]
```

```
Accuracy:  0.83

3 nearest neighbours
Correct labels are:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Predicted labels are:
[0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 1 0 1
 1 0 0 0 1 1 0 1 1 1 0 1 1 1 1 1 0 1 1 1 0 1 0 0 1 1 0 1 1 0 0 1 1 0 0 1 1
 1 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 0 0 0 1 1
 1 1 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 1 0 1 0 0 1 1 1 1 1
 1 1 1 1 0 1 0 1 1 0 1 1 0 1 1 0 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 1 1 1 1 0 1
 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1]
Accuracy:  0.805

5 nearest neighbours
Correct labels are:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Predicted labels are:
[0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 0 1
 1 0 0 0 0 1 0 1 1 1 0 0 1 1 0 1 1 1 1 1 0 1 0 1 0 1 1 1 1 0 0 1 1 0 0 1 1
 1 0 1 0 0 0 1 0 0 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 1 1 0 0 1 0 1 0 0 1 0 1 1
 1 0 1 0 1 0 1 0 1 0 0 1 1 1 1 1 0 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 1 0 0 0 0 1 1]
Accuracy:  0.79
```

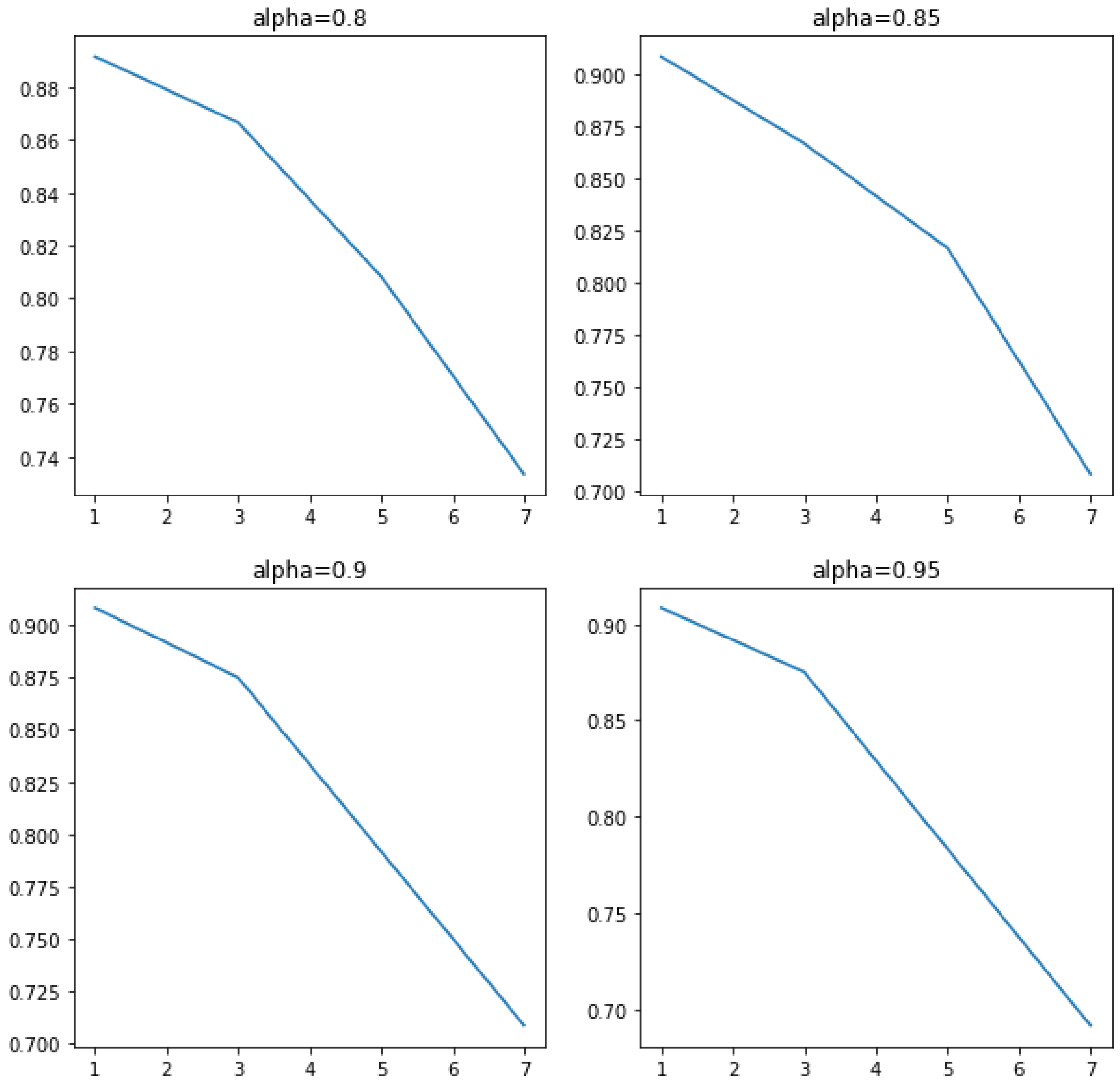
```
7 nearest neighbours
Correct labels are:
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1]
Predicted labels are:
[0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 1 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 1 1 1 1 0 1 0 0 1 1 0 1 0 1 1 0 1
 1 0 0 0 0 1 0 1 1 1 0 0 0 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 0 0 1 1 0 0 1 0
 1 0 1 0 0 1 1 0 1 1 0 0 1 1 1 0 0 1 0 1 1 0 1 1 0 0 0 0 1 1 1 0 0 1 0 1 1
 1 0 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 1 1 0 0 1 1 0 1 0 0 0 0 1 0 1 1 1 1
 1 1 1 1 1 1 0 1 1 0 1 1 1 0 1 0 0 1 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 0 0
 1 1 1 1 1 1 1 1 1 0 1 0 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 1 1]
Accuracy:  0.8
```

8. Bonus (5 Points)

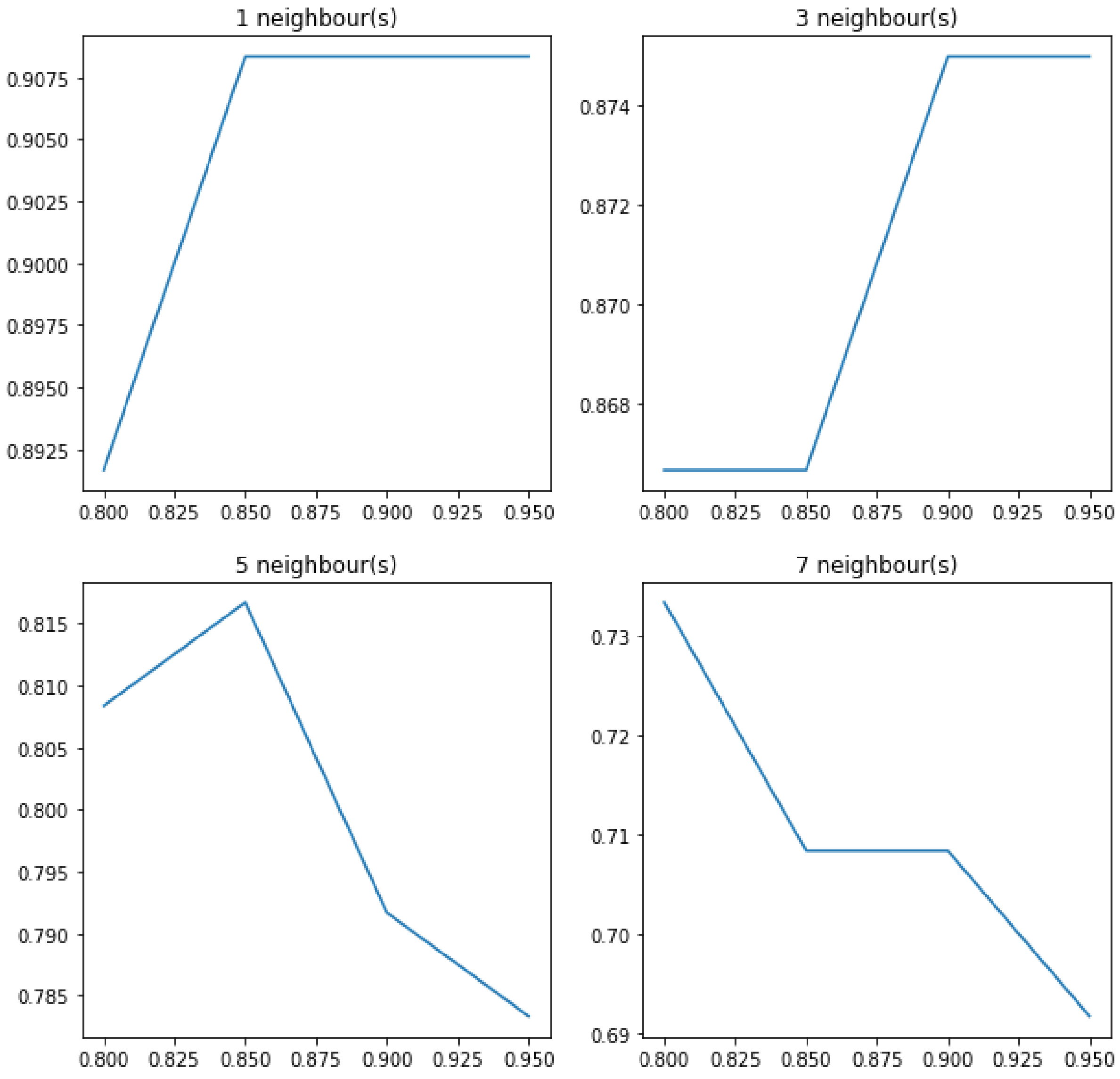
We split the data into 70:30 partitions using the following function:

```
def split_train_test(train_percentage, data, labels):
  test_percentage = 100-train_percentage
  number_of_data = data.shape[0]
  random.seed(2)

  train_split_size = (number_of_data * train_percentage) // 100
  test_split_size = number_of_data - train_split_size


  list1 = range(number_of_data)
  list11 = np.arange(number_of_data)

  train_indices = random.sample(list1, train_split_size)
  test_indices = np.delete(list11, train_indices)


  train_data = np.zeros( (train_split_size, data.shape[1]) )
  test_data = np.zeros( (test_split_size, data.shape[1] ) )
  train_labels = np.zeros( (train_split_size, ) )
  test_labels = np.zeros( (test_split_size, ) )

  for index1, index2 in enumerate(train_indices) :
    train_data[index1] = data[index2]
    train_labels[index1] = int(labels[index2])

  for index1, index2 in enumerate(test_indices):
    test_data[index1] = data[index2]
    test_labels[index1] = int(labels[index2])


  return train_data, test_data, train_labels, test_labels
```

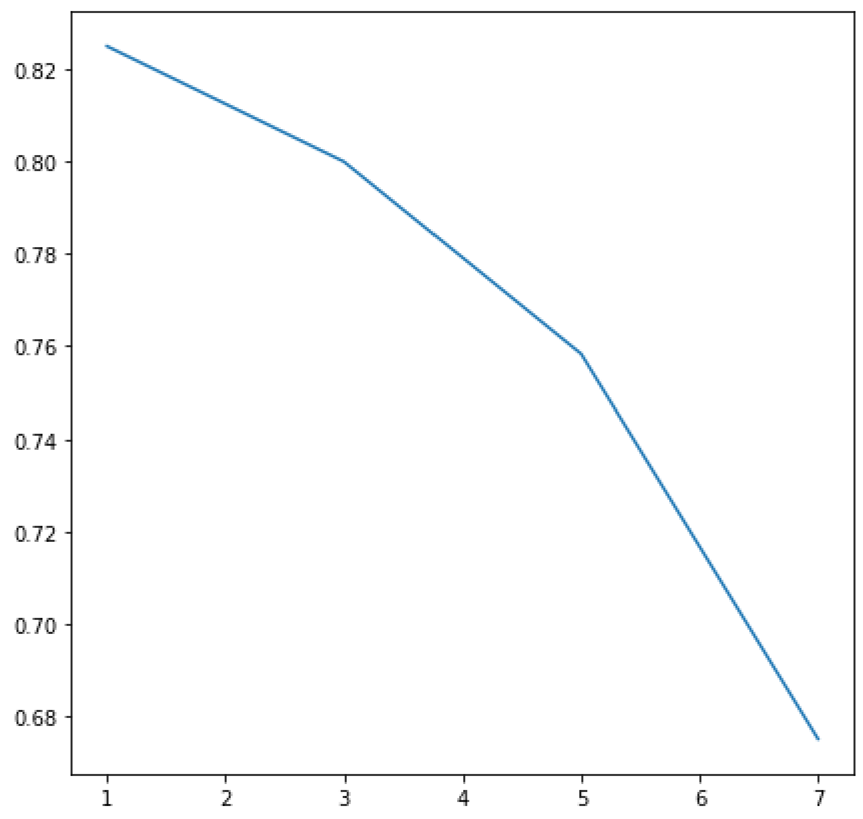
Comparing different K values for PCA



Comparing different alpha values for PCA:



LDA results:



On average, the 50/50 split gave slightly better accuracy.