National University
of computer and emerging sciences

Assignment 3

CS-461 Artificial Intelligence
BS(CS) – D
Batch 2018

Submitted By:

Hassan Shahzad 18i-0441

Submitted to:

Sir Saad Salman

Date of Submission:

12-06-21

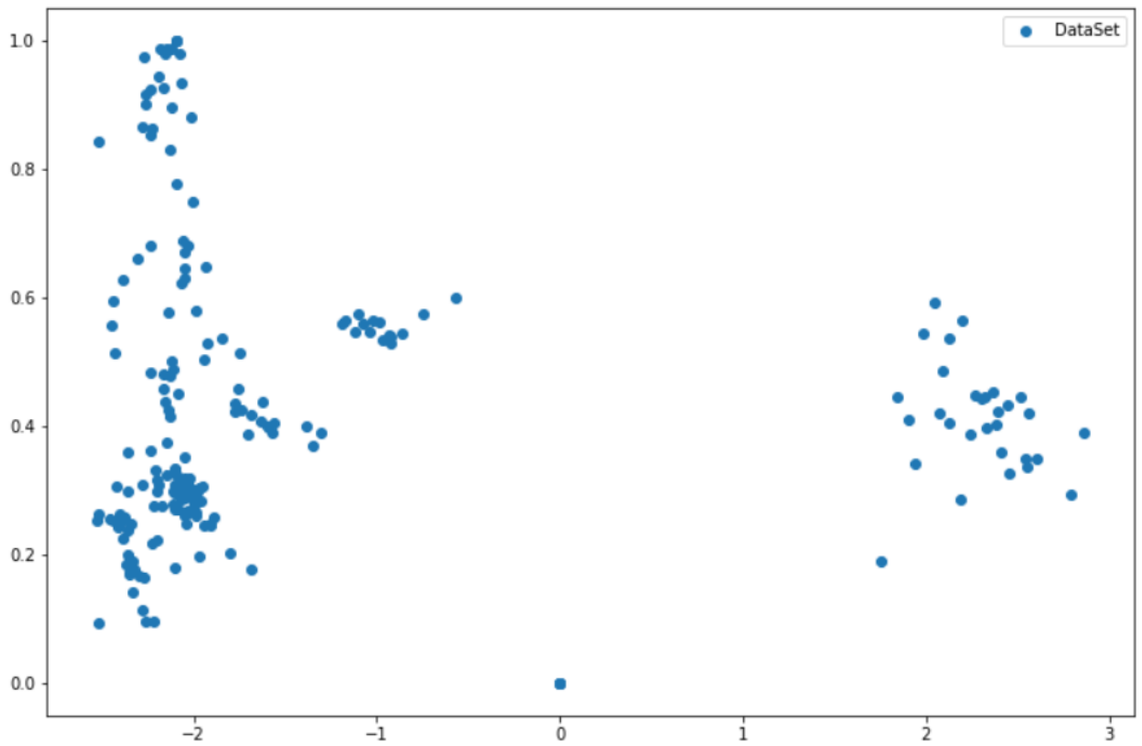
Table of Contents

Introduction:

In this assignment we had to choose a dataset of our choice and then implement K-Means algorithm on it and verify by visualizing it. Furthermore, we had to calculate the Davies Bouldin Index along with other evaluation parameters.
For this assignment I used Image Segmentation dataset. This dataset was downloaded from *"archive.ics.uci.edu"*. The information in the dataset was drawn randomly from a database of 7 outdoor images. The images were hand segmented to create a classification for every pixel. Each instance is a 3x3 region. The number of instances of Test and Training data are 2100 and 210 respectively.

Initial visualization of data is as follows:

```
Out[4]: <matplotlib.legend.Legend at 0x1c2c8506310>
```



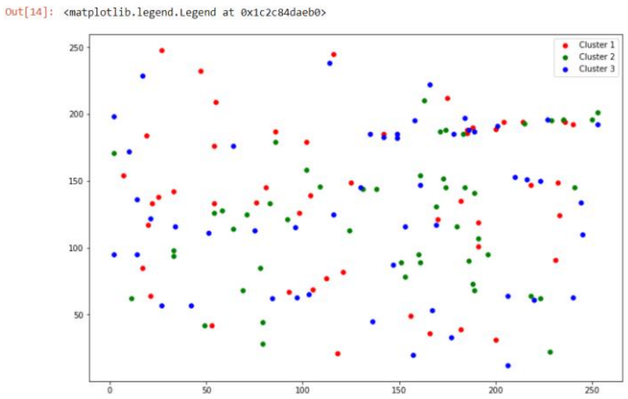
K-Means Clustering:

K-means algorithm is a centroid-based algorithm. It's an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (which we call clusters) where each datapoint belongs to only one group. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different as possible. It is one of the simplest and most popular unsupervised machine learning algorithms.

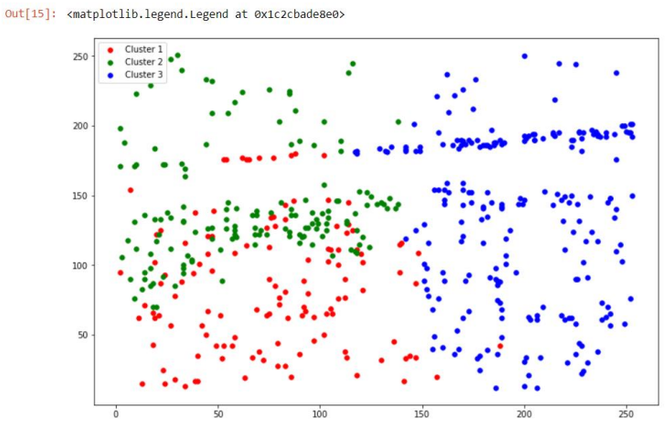
Implementation:

1. Firstly, we specify the number of clusters (K).
2. Then we choose centroids by selecting K random points from the data.
3. All the points are assigned to the closest cluster centroid by finding the Euclidean distance between the point and the centroid.
4. We recalculate the centroids of the newly formed clusters.
5. This process will keep repeating until the maximum number of iterations is reached or the centroids of the newly formed clusters retain their values (do not change).

The visualization of clusters before K-Means is as follows:



The visualization of clusters after K-Means is as follows:



Davies-Bouldin Index:

In classification, measuring the quality of the classification is very straightforward and simple. Quantifying clustering quality, however, is a little different. The performance metric being used here is the 'Davies Bouldin Index'. It is defined as a ratio between the cluster scatter and the clusters' separation; a lower value will mean that the clustering is better. The idea is that no cluster should be similar to another – so the best clustering algorithm would be minimizing the Davies-Bouldin index as clusters would be separate from each other.

The Davies-Bouldin Index is as follows:

```
In [22]:  1  compute_DB_index(X, labels1, centroids, 3)

Out[22]: 1.402993019611397
```

KNN (K-Nearest Neighbors):

KNN is a classification algorithm whose purpose is to use a database in which the datapoints are separated into several classes to predict the classification of a new sample point. It is based on feature similarity. A given data point is classified on the basis of how closely its features resemble the features of our training set. The algorithm outputs a discrete class in which it thinks an object belong, with this class being the one most common among its k nearest neighbors.

Implementation:

1. The dataset is loaded and sorted into training and test datasets and their respective labels/classes are assigned.
2. The Euclidean distance is calculated between each row of the test data and the training data.
3. These distances are sorted in ascending order and their indexes stored.
4. We only need to see k neighbors, so we take k number of indices and use it to retrieve the k nearest labels.
5. From these k nearest labels, the most common one is taken as predicted class.
6. This is applied to every row in the testing dataset and an array of corresponding labels/classes is generated and returned.

Evaluation Metrics:

We had to calculate several scores for each class that the model had predicted. Following are the scores that we calculated:

- **True Positive (TP):** It refers to the number of predictions where the classifier correctly predicts the positive class as positive
- **True Negative (TN):** It refers to the number of predictions where the classifier correctly predicts the negative class as negative
- **False Positive (FP):** It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive
- **False Negative (FN):** It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative
- **Accuracy:** It gives you the overall accuracy of the model, meaning the fraction of the total samples (for a particular class) that were correctly classified by the algorithm.

  Formula: (TP+TN)/(TP+TN+FP+FN)
- **Precision:** It tells you what fraction of predictions as a positive class were actually positive. A perfect precision would be one.

  Formula: TP/(TP+FP)
- **Recall:** It tells you what fraction of all positive samples of a class were correctly predicted as positive by the classifier.

  Formula: TP/(TP+FN)
- **F1-Score:** It combines precision and recall into a single measure. Mathematically it's the harmonic mean of precision and recall. An F1-score of 1 indicates a 100% accuracy for the algorithm.

  Formula: 2 * (precision * recall / precision + recall)

The micro F1-score is also taken by considering the total TP, total FP and total FN of the model (instead of considering each class individually.) The macro F1-score is taken by finding the unweighted mean of the F1-scores of all the classes.

In [33]:
```
run_KNN(np.array(train_x), np.array(test_x),train_y, np.array(test_y), 2)   # Testing KNN
```

```
For Optimal Value of K, K =  14
For Class =  BRICKFACE :  {'Accuracy': 0.9761904761904762, 'Precision': 1.0, 'Recall': 0.8571428571428571, 'F-1': 0.923076923076923}
For Class =  CEMENT :  {'Accuracy': 0.9714285714285714, 'Precision': 0.8666666666666667, 'Recall': 0.9285714285714286, 'F-1': 0.896551724137931}
For Class =  FOLIAGE :  {'Accuracy': 0.9714285714285714, 'Precision': 0.9, 'Recall': 0.9, 'F-1': 0.9}
For Class =  GRASS :  {'Accuracy': 0.9952380952380953, 'Precision': 1.0, 'Recall': 0.967741935483871, 'F-1': 0.9836065573770492}
For Class =  PATH :  {'Accuracy': 0.9952380952380953, 'Precision': 1.0, 'Recall': 0.967741935483871, 'F-1': 0.9836065573770492}
For Class =  SKY :  {'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'F-1': 1.0}
For Class =  WINDOW :  {'Accuracy': 0.9666666666666667, 'Precision': 0.8, 'Recall': 0.96, 'F-1': 0.8727272727272728}
No handles with labels found to put in legend.
For K =  16
For Class =  BRICKFACE :  {'Accuracy': 0.9714285714285714, 'Precision': 1.0, 'Recall': 0.8333333333333334, 'F-1': 0.9090909090909091}
For Class =  CEMENT :  {'Accuracy': 0.9666666666666667, 'Precision': 0.8333333333333334, 'Recall': 0.9259259259259259, 'F-1': 0.8771929824561403}
For Class =  FOLIAGE :  {'Accuracy': 0.9666666666666667, 'Precision': 0.9, 'Recall': 0.8709677419354839, 'F-1': 0.8852459016393444}
For Class =  GRASS :  {'Accuracy': 0.9952380952380953, 'Precision': 1.0, 'Recall': 0.967741935483871, 'F-1': 0.9836065573770492}
For Class =  PATH :  {'Accuracy': 0.9952380952380953, 'Precision': 1.0, 'Recall': 0.967741935483871, 'F-1': 0.9836065573770492}
For Class =  SKY :  {'Accuracy': 1.0, 'Precision': 1.0, 'Recall': 1.0, 'F-1': 1.0}
For Class =  WINDOW :  {'Accuracy': 0.9619047619047619, 'Precision': 0.7666666666666667, 'Recall': 0.9583333333333334, 'F-1': 0.8518518518518519}
No handles with labels found to put in legend.
```