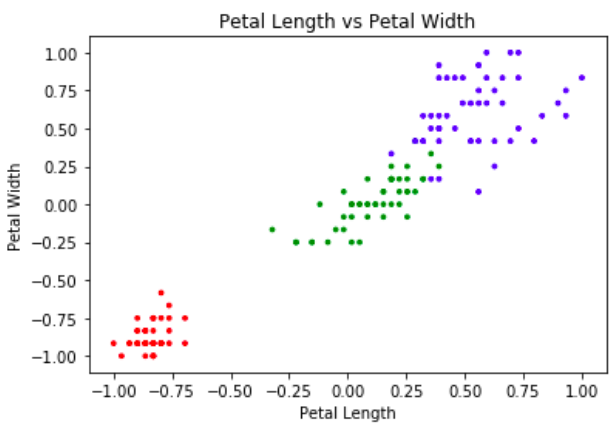
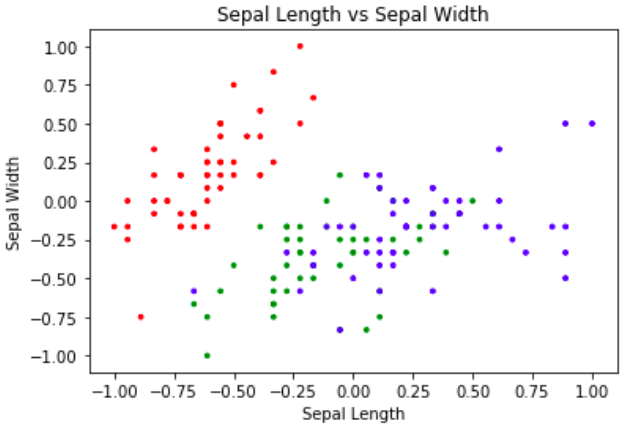
Lab Course Machine Learning
Exercise Sheet 11
Krithika Murugesan - 277537

Exercise 1: Implement K Means clustering algorithm

Dataset : **IRIS**

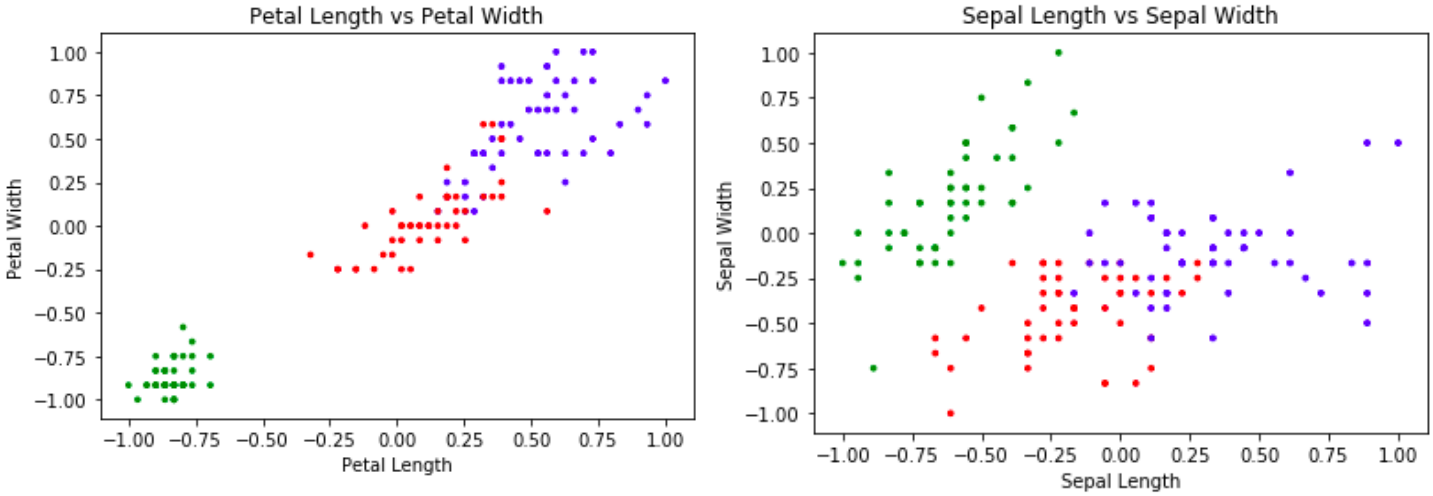
The data is in LibSvm format, each row represents one sample with first entry being the response variable followed by dependent variables. The data is read as a LibSvm file and converted to a dense matrix, since the data set is fairly small in size the matrix form can be used as it would not require huge computation capabilities. The class distribution is as follows, where each color represents one of the three clusters(Setosa, Virginica and Versicolor)






```python
#K means clustering
def kMeans(x,k):
    from copy import deepcopy
    c = initialCentroid1(x,k)
    # To store the value of centroids when it updates
    cOld = np.zeros(c.shape).astype("float")
    # Cluster Lables(0, 1, 2)
    clusters = np.zeros(x.shape[0])
    # Error func. - Distance between new centroids and old centroids
    error = euclideanDistance(c,cOld,None)
    # Loop will run till the error becomes zero
    while error.sum() != 0:
        # Assigning each value to its closest cluster
        i = 0
        for each in x.values:
            distances= euclideanDistance(np.vstack(each),c)
            cluster = np.argmin(distances)
            clusters[i] = cluster
            i = i+1
        # Storing the old centroid values
        cOld = deepcopy(c)
        # Finding the new centroids by taking the average value
        for i in range(k):
            points = [x.iloc[j] for j in range(len(x)) if clusters[j] == i]
            print(isinstance(points, list),points)
            c[i] = (np.mean(points, axis=0)).ravel()
        error = euclideanDistance(c, cOld)
    return(clusters,c)
```

The task is to implement clustering using k-means, which can be achieved by initially assuming random clusters centers and grouping the data points based on distance between the centers and the points themselves. Later we recompute the cluster centers and group the points till we reach the best distance between centers and data points. The distance measure used is euclidean distance. The results are pretty close to the original classification, the predicted classes based on petal and sepal widths are

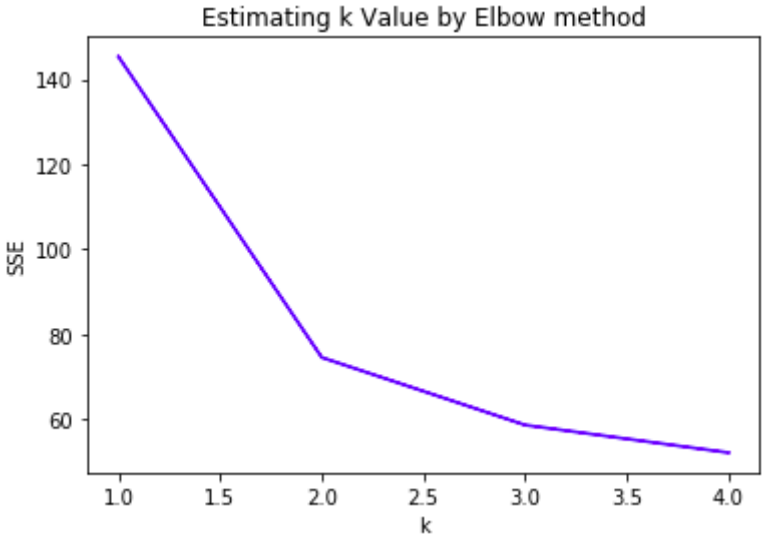


Computing the accuracy for unsupervised models like clustering is a complex task, as the system's assumption of group names may be vary with that of human assumption. Since, the data is small, assuming the classes from the graphs the accuracy can be manually computed by matching the labels assumed initially to the ones predicted; Which is about **83%**

```
#Adjusting the clusters, so as to accomodate computation of accuracy
original = ['A' if l == 1 else 'B' if l == 2 else 'C' for l in y['class']]
predict = ['A' if l == 2 else 'B' if l == 3 else 'C' for l in predicted]
```

The optimal k value can be obtained by several methods, one such method is Elbow method, where the each value for k, the SSE is calculated. As k increases, SSE decreases and the graph looks similar to that of an elbow, the k value at the elbow of the curve is the optimal value where thee SSE is neither too high nor approaching zero. For this dataset, 3 is the optimal value of k(no. of clusters)

```
#Optimal criteria for k
i,SSE = [],[]
for k in range(1,5):
    predicted,centroid = kMeans(x,k)
    predicted = np.array(predicted, dtype=np.int32)
    sse = 0
    for every,j in zip(x.values,predicted):
        sse = sse + (euclideanDistance(every,centroid[[j]]))
    i.append(k)
    SSE.append(sse)
```



Exercise 2: Cluster news articles

Dataset : 20 News Groups

This dataset contains newsgroup documents for twenty different categories. The dataset itself is quiet huge and is divided into train and test. The data is read into python lists by the following code snippet.

```
import os
#20 folder names
folder = ['a','b','c','e','f','g','h','i','j','k','l','m','n','o','p','q','r','s','t']

#Reading files in a list(train)
train = []
for each in folder:
    all_files = os.listdir(r'/home/kritz/Documents/ML_Lab/Exercise11/20news-bydate-train/'+each+'/')
    for i in all_files:
        with open(r'/home/kritz/Documents/ML_Lab/Exercise11/20news-bydate-train/'+each+'/'+i,'rb') as f:
            lines = f.read()
            train.append(lines)

#Reading files in a list(test)
test = []
for each in folder:
    all_files = os.listdir(r'/home/kritz/Documents/ML_Lab/Exercise11/20news-bydate-test/'+each+'/')
    for i in all_files:
        with open(r'/home/kritz/Documents/ML_Lab/Exercise11/20news-bydate-test/'+each+'/'+i,'rb') as f:
            lines = f.read()
            test.append(lines)
```

Since, the data is text it has to preprocessed in order to perform operations on it, first the data has to be cleaned. The following are done using inbuilt NLTK packages in python:

i)**Stemming** : The words are reduced to ground form so as avoid considering the conjugated verbs as different words

ii)**Stop words** : Some of the commonly occurring words like, “this”,”and”,etc..., which do not add a lot of value to the data are removed so as to increase the computationally efficiency.

The words cannot be used as such, so they are converted into frequency vectors like TFID, here the maximum number of words is limited which will affect the accuracy of the results but considering the naive approach of k-means implemented and capacity of the system it is limited.

```
#Text processing libraries
from nltk.stem import PorterStemmer
from nltk import word_tokenize
from sklearn.feature_extraction.text import TfidfVectorizer

train = pd.DataFrame([str(i) for i in train])
test = pd.DataFrame([str(i) for i in test])

#Stemming: reducing words to ground form
ps = PorterStemmer()
train['stemmed'] = train.apply(lambda x: [ps.stem(y) for y in x])
test['stemmed'] = test.apply(lambda x: [ps.stem(y) for y in x])

#Converting to Tfid vectors
tfid = TfidfVectorizer(max_df=0.5, max_features=1000, stop_words='english')
tfidTrainX = tfid.fit_transform(train.stemmed.dropna())
tfidTestX = tfid.transform(test.stemmed.dropna())


Train = pd.DataFrame(tfidTrainX.todense())
Test = pd.DataFrame(tfidTestX.todense())
```

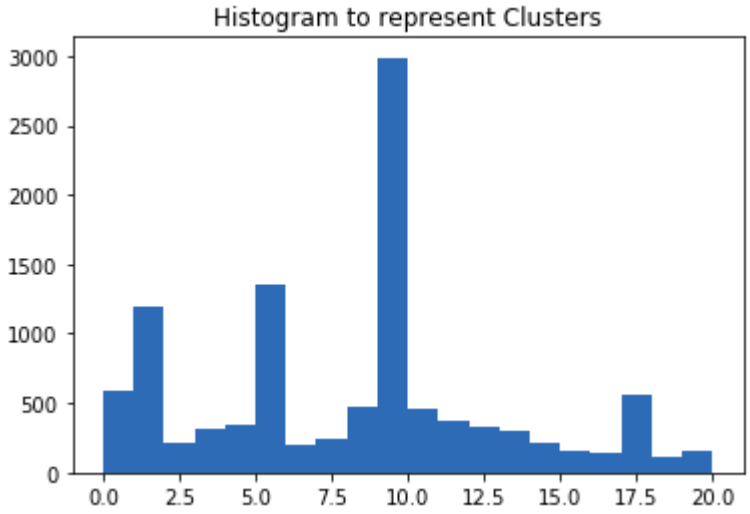
Considering there are 20 groups, the initial centroids are computed for huge dimensions,

```
#Generating Centroids for the larger datset
def initialCentroid1(x,k):
    c = []
    for i in range(1000):
        c.append ([np.random.uniform(0,999) for i in range(k)])
    d = np.array(c, dtype=np.float32)
    return(d)
```

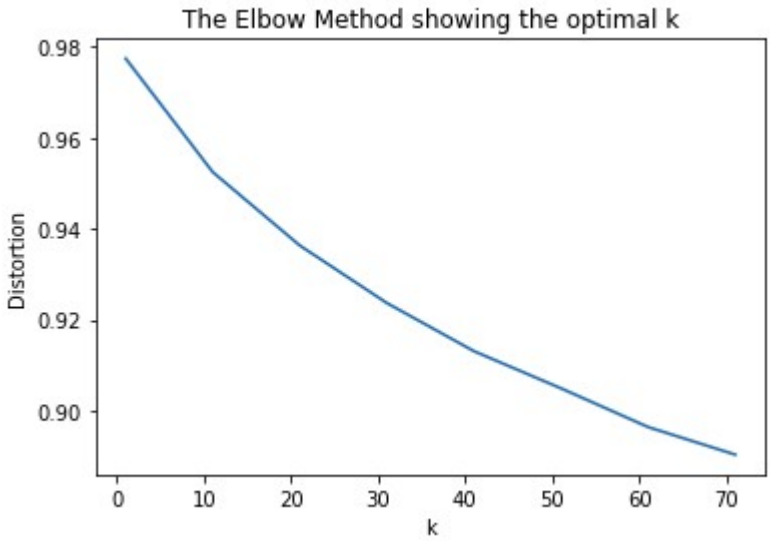
The algorithm took a considerable amount of time, considering it has to compute the distance for every vector each time, the scikit implementation of the same was way more efficient than the normal implementation. It was able to complete the learning in 34 seconds which took few hours with the code used earlier and sometimes ran into memory issues. The scikit version is way more efficient, it also has a batch variation which minimizes the same objective function in a faster time.

```
from sklearn.cluster import KMeans

#Number of clusters
num_clusters = 20

#Defining model
km = KMeans(n_clusters=num_clusters)

#Fitting model
%time km.fit(Train)
clusters = km.labels_.tolist()
```

Considering the huge amount of data, it is difficult to match with the original classification to compute accuracy, and the pre-processing and limiting the number of words to accommodate the algorithm also will contribute to a reduced accuracy. A histogram representing the clusters is as follows and it varies hugely across them.



The elbow method for optimal k value yielded the following results, though the data is grouped into 20 entities, the optimal k value goes somewhere near 60, showing a huge variation in the dataset we have.



Note : The LibSvm files are part of the zip folder.