Lab Course Machine Learning
Exercise 10

Exercise 1: Implement Kmeans Cluster Algorithm:

Function name : read_file
Parameter : name of the file to be converted to svm format

This function is used to convert the dataset which is in libsvm format to datafram for kmeans .

```
def read_file(filename):
    file = open(filename)
    lines = file.readlines()
    d1 = pd.DataFrame(np.zeros((150, 5)))
    for ind, line in enumerate(lines):
        line = line.split()
        d1[0][ind] = line[0]
        for i in line[1:]:
            d1[int(i[0])][ind] = i[2:]
    return d1
```

Function name : dist_matrix
Parameter : dataframe, initial centroid

This function is used to find the euclidean distance between two data points in order to get the initial centroids.

```
def dist_matrix(ftrd1,index):
    distarray = np.zeros((1,150))
    for ind,i in enumerate(ftrd1.values):
        distarray[0][ind] = np.sqrt(np.sum((ftrd1.values[index]-i)**2))
    return distarray
```

Function name : dist_mean
Parameter : dataframe, centroids

This function returns the mean and every other data points in the dataset.

```
def dist_mean(ftrd1,mean):
    distarray = np.zeros((150,len(mean)))
    for mean_i,i in enumerate(mean):
        for ind, j in enumerate(ftrd1.values):
            distarray[ind][mean_i] = np.sqrt(np.sum((i-j)**2))
    return distarray
```

Function name : get_mean
Parameter : cluster labels, data with labels, data, number of clusters

After each iteration, this function returns the mean of clustered datapoints and checks it with the labels of next dataset to conclude the centroids of the clusters.

```
def get_mean(cluster_label,d1,ftrd1,cluster):
    mean = list()
    for i in range(0,cluster):
        features = len(ftrd1.columns)
        dframe = ftrd1[d1["Cluster"]==i]
        average = dframe.mean(axis=0)
        mean.append(np.array(average))
    return mean
```

Function name : get_clusters
Parameter : data, number of clusters

Once the kmeans algorithm is converged, for the final centroids of the clusters, this function groups the datapoints into clusters and returns the same.

```
def get_clusters(ftrd1,d1,clusters):
    final_clusters = dict()
    for i in range(0,clusters):
        final_clusters[i] = ftrd1[d1.Cluster == i]
    return final_clusters
```

Function name : get_clusters
Parameter : data, number of clusters

This function performs the Kmeans clustering algorithm for the provided datasets.

```
def k_means(ftrd1,index,clusters):
    cluster_mean = initial_centroids(ftrd1,index,clusters)
    cluster_old = list()
    for it in range(0,100):
        cluster_new = list()
        mean_dist = dist_mean(ftrd1,cluster_mean)
        for mindist in mean_dist:
            minimum = min(mindist)
            if Counter(mindist)[minimum] > 1:
                rand = [r for r, x in enumerate(mindist) if x == minimum]
                cluster_new.append(random.choice(rand))
            else:
                cluster_new.append(list(mindist).index(minimum))
        d1["Cluster"] = cluster_new
        cluster_mean = get_mean(cluster_new,d1,ftrd1,clusters)
        if cluster_new == cluster_old:
            print("Clustered, Iteration: ",it)
            return cluster_mean
        else:
            cluster_old = cluster_new
    return cluster_mean,np.sum(dist_mean(ftrd1,cluster_mean))
```

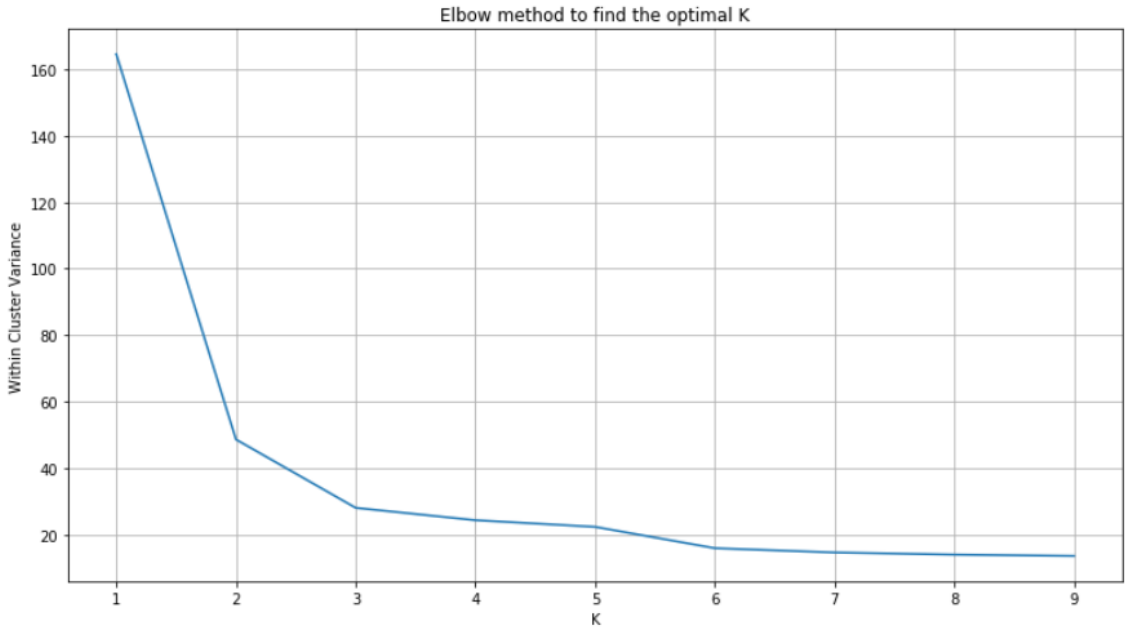
```
start = time.time()
clusters = np.arange(1,10)
variance = list()
for k in clusters:
    var = 0
    print("k: ",k)
    clustermean = k_means(ftrd1,index,k)
    final_clusters = get_clusters(ftrd1,d1,k)
    error = 0
    for i in final_clusters:
        for j in final_clusters[i].values:
            error += np.sum(pow((clustermean[i] - j) ,2))
    variance.append(error)
end = time.time()
```

Elbow method:

Reference: ***https://www.datanovia.com/en/lessons/determining-the-optimal-number-of-clusters-3-must-know-methods/#elbow-method***

1. Compute clustering algorithm (e.g., k-means clustering) for different values of k. For instance, by varying k from 1 to 10 clusters.
2. For each k, calculate the total within-cluster sum of square (wss).
3. Plot the curve of wss according to the number of clusters k.
4. The location of a bend (knee) in the plot is generally considered as an indicator of the appropriate number of clusters.

To find the optimum K for the Kmeans clustering algorithm, I have chosen the elbow criterion. This method calculates the within cluster variances (SSE) for every cluster depending on the number of cluster and the same is plotted and relevant K value is chosen. For K with a range of 1 to 9, the computation time was 4 seconds.

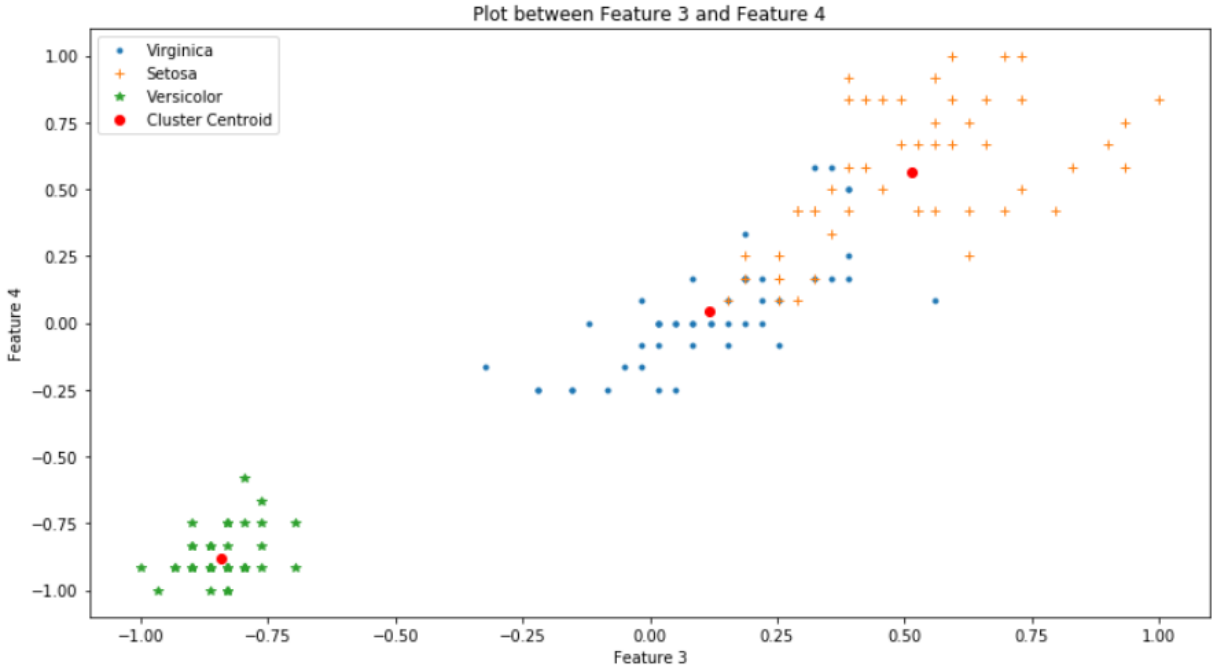


At K: 3

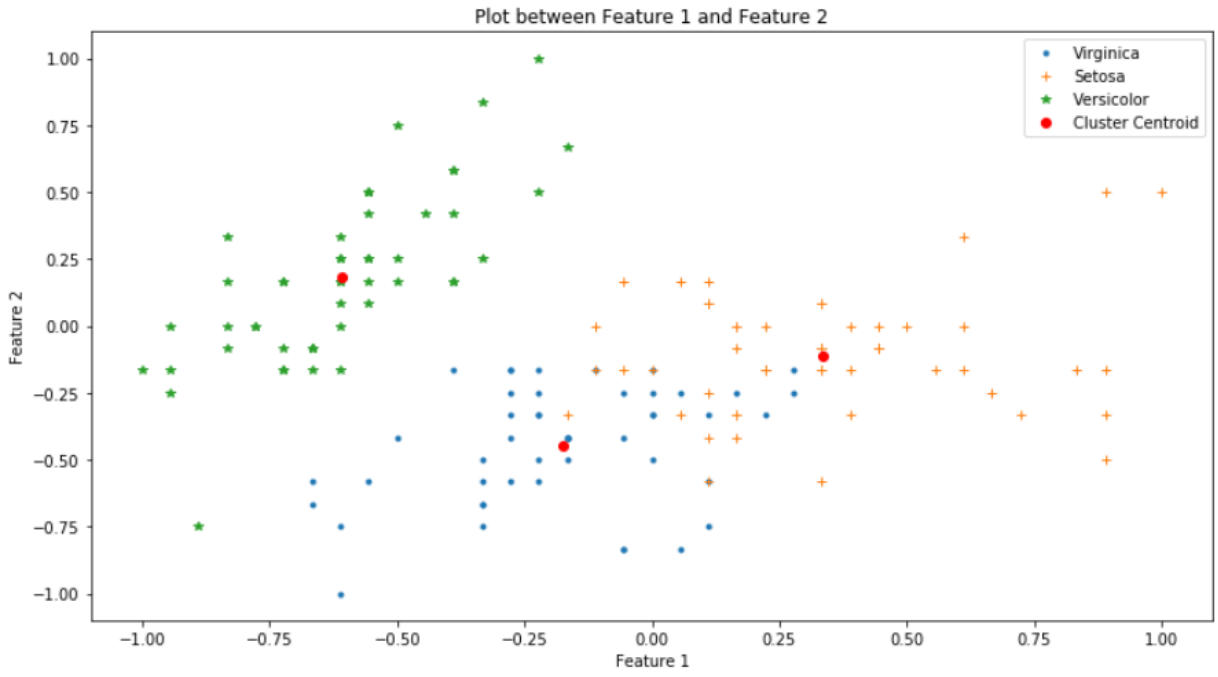
Cluster centroids

```
[array([-0.17592596, -0.44618056,  0.11793781,  0.04166668]),
 array([ 0.33546999, -0.11378209,  0.5143415 ,  0.56410263]),
 array([-0.60777778,  0.18166667, -0.84271174, -0.88000014])]
```

From the elbow method, it can be observed that when K is 3 there is a mild change in the pattern and hence it is chosen. Also the centroids/ means of the clusters are shown above.



.



Cluster News Articles – Kmeans Implementation:

```
path = "G:/DA - Hildeshim/ML Lab/Exercise 10/20news-bydate/20news-bydate-train/"
x = list()
y = list()
for newsgroup in os.listdir(path):
    for files in os.listdir(path+str(newsgroup)):
        with open(path+str(newsgroup)+"/"+str(files), 'rb') as file:
                x.append(file.read())
                y.append(newsgroup)
```

The above code is used to append the text within the document file by file and the corresponding labels are stored in the y list.

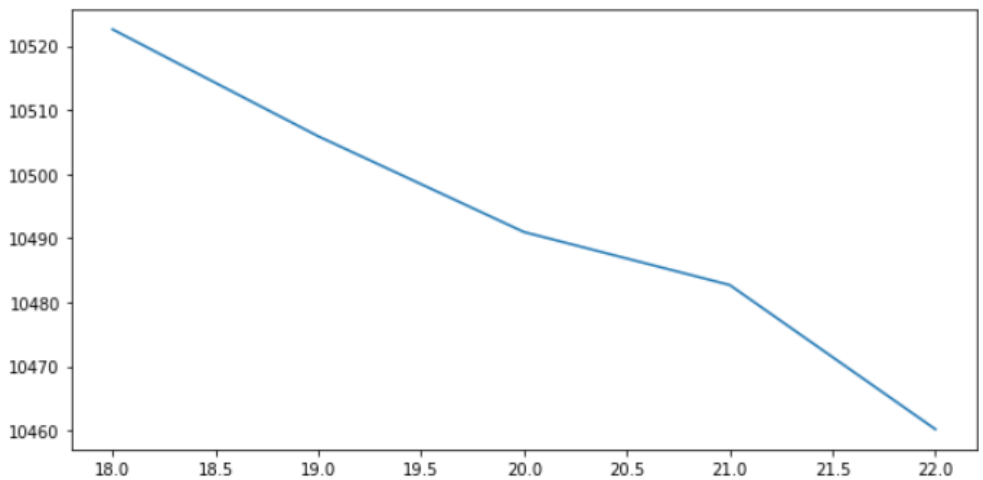
The text is converted into feature vector using TfIdf vectorizer with the tokenizers for prepocessings including removing stopwords, changing the case of the text and stemming of the words.

```
def Tokenizer(str_input):
    str_input = str_input.lower()
    words = word_tokenize(str_input)
    #remove stopwords
    stop_words = set(stopwords.words('english'))
    words = [w for w in words if not w in stop_words]
    #stem the words
    porter_stemmer=nltk.PorterStemmer()
    words = [porter_stemmer.stem(word) for word in words]
    return words
```

```
vectorizer = TfidfVectorizer(tokenizer=Tokenizer,min_df=0.001,max_df=0.3, max_features=100000)
xTrain = vectorizer.fit_transform(x)
xTrain = xTrain.todense()
```

To find the optimum K for the Kmeans clustering algorithm, I have chosen the elbow criterion. This method calculates the within cluster variances (SSE) for every cluster depending on the number of cluster and the same is plotted and relevant K value is chosen. For K with a range of 18 to 22, the computation time was 2 hour and 10 minutes.

Elbow method (K vs SSE)



K: 20

When K is set to 20, it took 37 iterations to converge with a total computation time of 50 minutes for 25941 features per document. The following is the precision, recall and F1 score of the predicted clusters of the test data with the given centroids of the clusters and it can be seen that the quality is very low and poor.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	319
1	0.00	0.00	0.00	389
2	0.05	1.00	0.10	394
3	0.00	0.00	0.00	392
4	0.00	0.00	0.00	385
5	0.00	0.00	0.00	395
6	0.00	0.00	0.00	390
7	0.00	0.00	0.00	396
8	0.00	0.00	0.00	398
9	0.00	0.00	0.00	397
10	0.00	0.00	0.00	399
11	0.00	0.00	0.00	396
12	0.00	0.00	0.00	393
13	0.00	0.00	0.00	396
14	0.00	0.00	0.00	394
15	0.00	0.00	0.00	398
16	0.00	0.00	0.00	364
17	0.00	0.00	0.00	376
18	0.00	0.00	0.00	310
19	0.00	0.00	0.00	251
micro avg	0.05	0.05	0.05	7532
macro avg	0.00	0.05	0.00	7532
weighted avg	0.00	0.05	0.01	7532

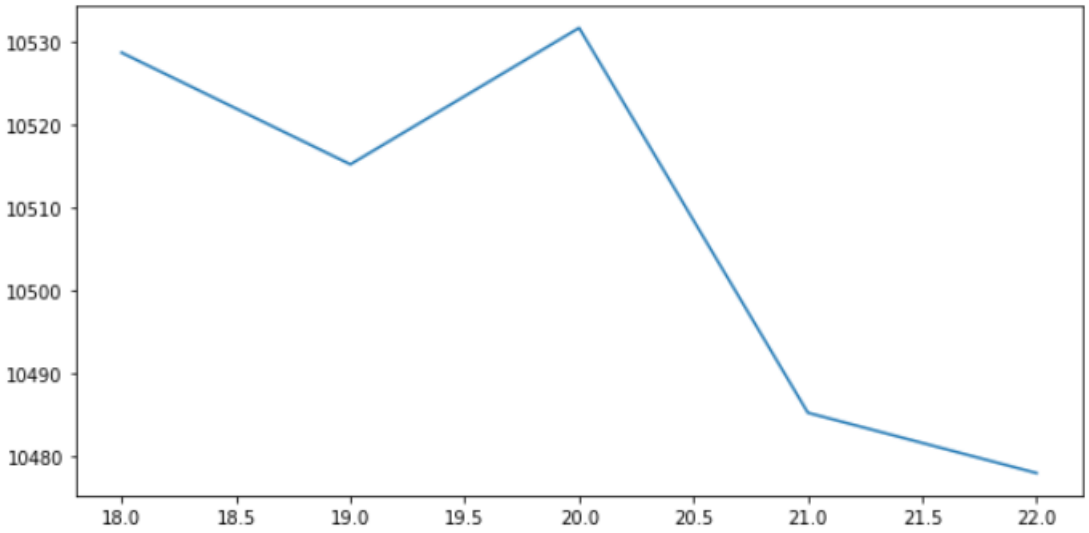
Cluster News Articles – Kmeans sklearn:

The similar preprocessing procedures are carried out for the implementation with sklearn but the change is that the algorithm provided by sklearn library is used here. For K with a range of 18 to 22, the computation time was 2 hour and 10 minutes.

```
start = time.time()
error = []
for i in range (18,23):
    km = KMeans(n_clusters=i,n_init=1)
    km.fit(xTrain,yTrain)
    print("K: ",i,"WCSE: ",km.inertia_)
    error.append(km.inertia_)
end = time.time()
print("\n\nTime taken: ", end - start)
```

```
K:  18 WCSE:  10528.651780640565
K:  19 WCSE:  10515.1695532143
K:  20 WCSE:  10531.635937274315
K:  21 WCSE:  10485.193039850697
K:  22 WCSE:  10477.914094172254


Time taken:  1016.1474456787109
```



K: 20

When K is set to 20, it 5 minutes for 25941 features per document. The following is the precision, recall and F1 score of the predicted clusters of the test data with the given centroids of the clusters and it can be seen that the quality is very low and poor and better compared to the self-implementation.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	319
1	0.00	0.00	0.00	389
2	0.00	0.00	0.00	394
3	0.00	0.00	0.00	392
4	0.00	0.00	0.00	385
5	0.00	0.00	0.00	395
6	0.05	0.93	0.09	390
7	0.00	0.00	0.00	396
8	0.00	0.00	0.00	398
9	0.00	0.00	0.00	397
10	0.00	0.00	0.00	399
11	0.02	0.00	0.00	396
12	0.00	0.00	0.00	393
13	0.00	0.00	0.00	396
14	0.50	0.00	0.01	394
15	0.00	0.00	0.00	398
16	0.04	0.01	0.01	364
17	0.00	0.00	0.00	376
18	0.00	0.00	0.00	310
19	0.00	0.00	0.00	251
micro avg	0.05	0.05	0.05	7532
macro avg	0.03	0.05	0.01	7532
weighted avg	0.03	0.05	0.01	7532

It can be concluded that the kmeans by sklearn performed better than my algorithm. It was fast in computation and the clustering is good compared to my algorithm.