Lab Course Machine Learning | Kalaiselvan Panneerselvam

Exercise 10 | 278920

Exercise 1: Implement K Means clustering algorithm

Loading IRIS Dataset

```
file = open("iris.scale.txt")
df=pd.DataFrame(np.zeros((150,5)))
for ind,line in enumerate(file):
    line=line.split()
    df[0][ind] = line[0]
    for i in line[1:]:
        df[int(i[0])][ind] = i[2:]
```

	1	2	3	4
0	-0.555556	0.250000	-0.864407	-0.916667
1	-0.666667	-0.166667	-0.864407	-0.916667
2	-0.777778	0.000000	-0.898305	-0.916667
3	-0.833333	-0.083333	-0.830508	-0.916667

```
x.shape
```

(150, 4)

Implement K Means clustering algorithm

```
def centroids(k):
    centroids=[]
    c = x[np.random.randint(0,len(x))]
    centroids.append(c)
    for i in range(k-1):
        c_new = x[max_dist(c)]
        if c_new not in np.unique(centroids, axis=0):
            centroids.append(c_new)
            c=c_new
        else :
            c_new = x[np.random.randint(0,len(x))]
            centroids.append(c_new)
    return centroids
def euclideanDistance(a, b):
    a = np.array(a)
    b = np.array(b)
    return np.sqrt(np.sum(np.square(a - b)))
def clusters(centroids):
    clusters = np.zeros(len(x))
    for j in range(len(x)):
        dist=[]
        for i in iter(list(centroids)):
            dist.append(euclideanDistance(x[j], i))
        cluster = np.argmin(dist)
        clusters[j] = cluster
    return clusters
def max_dist(c):
    distances=[]
    for i in range(len(x)):
        distances.append(euclideanDistance(x[i], c))
    return np.argmax(distances)
def load_data(dir):
    y = []
    x = []
    for fol in os.listdir(dir):
        for f in os.listdir(dir+"/"+str(fol)):
            with open(dir+"/"+str(fol)+"/"+str(f), 'rb') as file:
                x.append(file.read().decode('cp1252').encode("utf-8"))
                y.append(fol)
    return x, y
```

Optimal value of K :

K-means is a simple unsupervised machine learning algorithm that groups a dataset into a user-specified number (k) of clusters. The algorithm is somewhat naive--it clusters the data into k clusters, even if k is not the right number of clusters to use. One method to validate the number of clusters is the elbow method. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of k and for each value of k calculate the sum of squared errors (SSE).

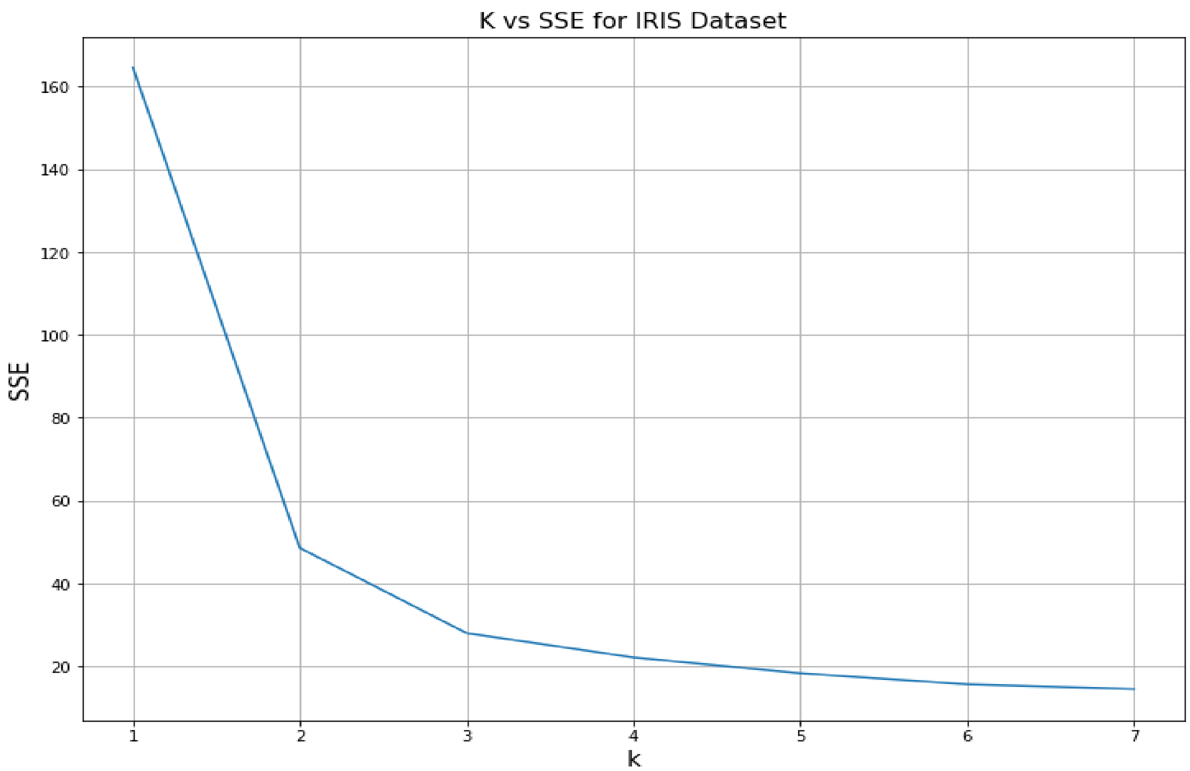
For IRIS dataset, I calculated k from 1 to 8 in which optimal K is tend to be 3.

Plot of K vs SSE using my implementation of SKLearn function

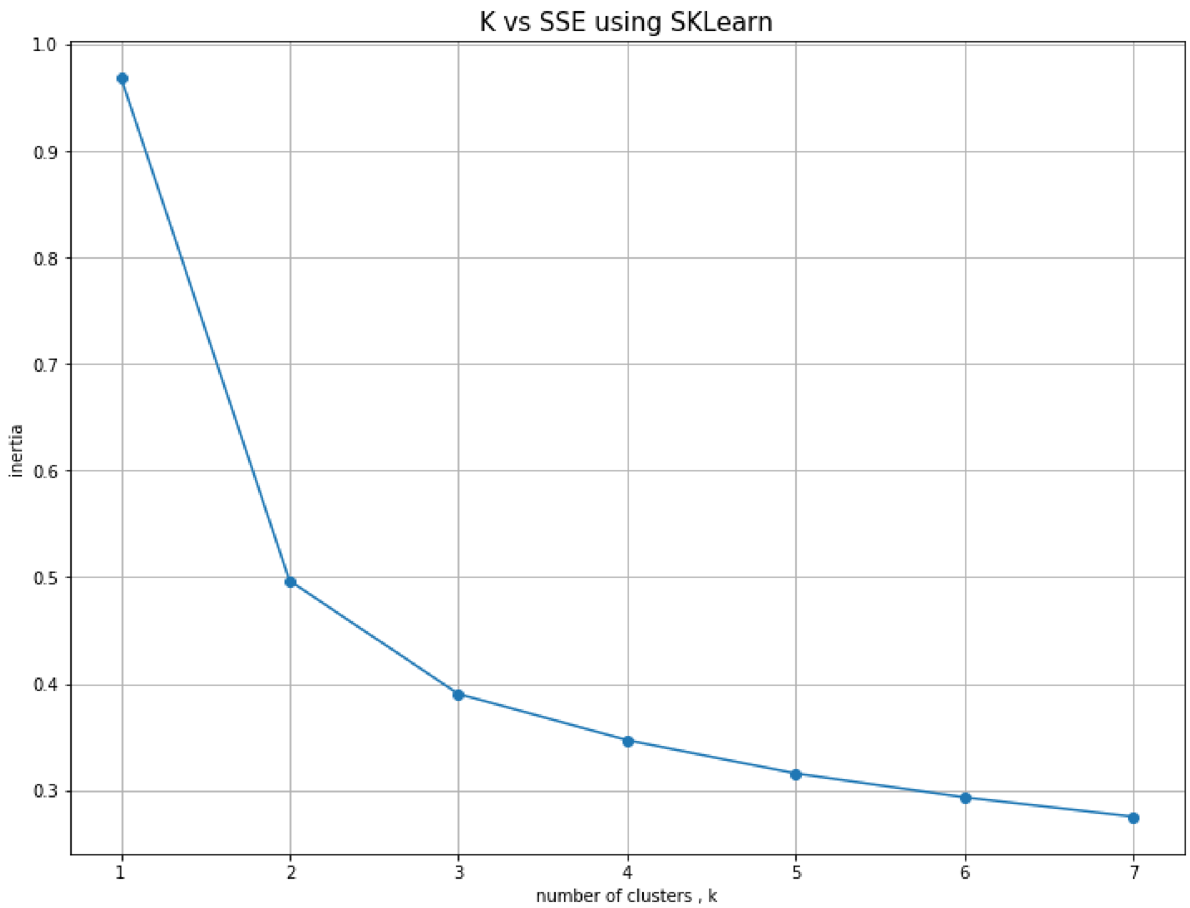
```
Converged in Iterations 1

K value :2
48.574745171474284


Converged in Iterations 4

K value :3
27.992453528801588


Converged in Iterations 4

K value :4
22.13132224532135


Converged in Iterations 7

K value :5
18.322999347220673


Converged in Iterations 6

K value :6
15.663208265337682


Converged in Iterations 9

K value :7
14.508808674682175


Converged in Iterations 8
```

```
54]: sse_list

54]: [164.5526773463816,
      48.574745171474284,
      27.992453528801588,
      22.13132224532135,
      18.322999347220673,
      15.663208265337682,
      14.508808674682175]
```

Plots of K vs SSE for IRIS Dataset using my own K-means implementation



Plots of K vs SSE for IRIS Dataset using SKLearn implementation



Exercise 2: Cluster news articles(10 Points)

Loading 20Newsgroups dataset Dataset

```
xT, yT = load_data("20news-bydate/20news-bydate-train")
xtest, ytest = load_data("20news-bydate/20news-bydate-test")
```

Preprocessing the dataset

```
v = TfidfVectorizer(stop_words = 'english',lowercase = True, ngram_range=(1, 3))
xTrain = v.fit_transform(xT)
xTest = v.transform(xtest)
```

```
xTrain.shape
```

```
(11314, 32659)
```

```
xTest.shape
```

```
(7532, 32659)
```

```
]: xt=xTrain[:1000]
   x=xt.todense()
   x.shape
]: (1000, 32659)
```

```
]: x=np.nan_to_num(x)
   x.shape
]: (1000, 32659)
```

I could not process full dataset as it throws Memory Error as my system has only 4GB of RAM. So I processed only first 1000 rows with K=20 which is the optimal value of K for News Article Dataset. It took more than two hour to converge. So I implemented the K-means using SKLearn function.

```
import datetime
start = datetime.datetime.now()
ks = range(8, 25)
inertia = []
for i in ks:
    kmeans = KMeans(n_clusters=i, n_init = 1)
    kmeans.fit(xTrain, yTrain)
    print(kmeans.inertia_)
    inertia.append(kmeans.inertia_)
end = datetime.datetime.now()
```

```
11026.60809967372
10991.463730172227
10983.636443475085
10947.170146492641
10911.6965696456
10913.49172160891
10874.1048004594
10895.676236814581
10844.932107040988
10848.185035432509
10796.135453128783
10813.064981249194
10794.485791735386
10796.501476474363
10774.161819095198
10736.441985384521
10738.790668587344
```

```python
print("The time taken is "+str(end-start))
```

```
The time taken is 0:31:22.146380
```

Plot of K vs SSE for News Article Dataset