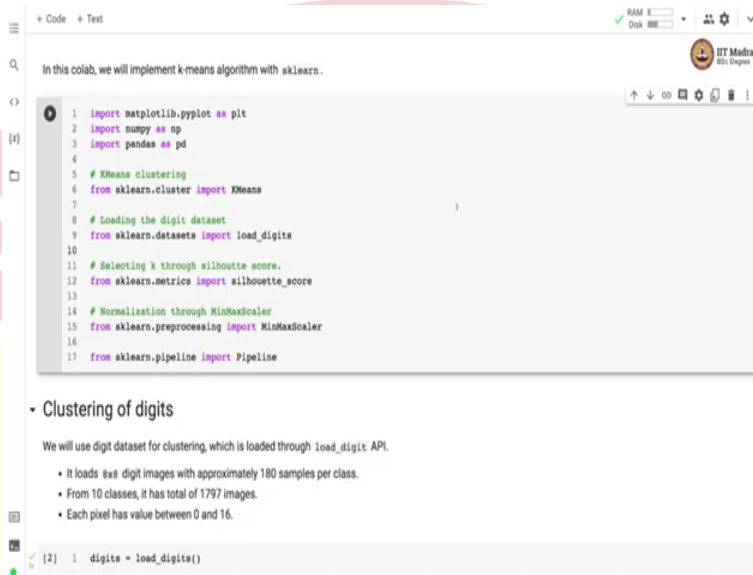


IIT Madras

ONLINE DEGREE

Machine Learning Practice
Indian Institute of Technology, Madras
Programming and Data Science
K-means clustering on Digital Dataset

(Refer Slide Time: 00:10)



```
1 import matplotlib.pyplot as plt
2 import numpy as np
3 import pandas as pd
4
5 # Kmeans clustering
6 from sklearn.cluster import KMeans
7
8 # Loading the digit dataset
9 from sklearn.datasets import load_digits
10
11 # Selecting k through silhouette score.
12 from sklearn.metrics import silhouette_score
13
14 # Normalization through MinMaxScaler
15 from sklearn.preprocessing import MinMaxScaler
16
17 from sklearn.pipeline import Pipeline
```

Clustering of digits

We will use digit dataset for clustering, which is loaded through `load_digit` API.

- It loads `axis` digit images with approximately 180 samples per class.
- From 10 classes, it has total of 1797 images.
- Each pixel has value between 0 and 16.

```
[2] 1 digits = load_digits()
```

Namaste! Welcome to the next video of Machine Learning Practice Course. In this video we will implement k-means algorithm with a sklearn. The first import basic Python libraries like numpy and pandas and matplotlib.pyplot for plotting. K-means clustering is implemented as part of sklearn.cluster module. Then for this demonstration we use the digit dataset which is loaded to `load_digit` API from sklearn.datasets module.

We will be selecting the optimal K through `silhouette_score` which is implemented as part of sklearn.metrics module and will normalize with `MinMaxScaler`. Finally, we use pipeline to combine the data pre-processing and clustering part.

(Refer Slide Time: 01:01)

The screenshot displays a Jupyter Notebook interface with a code editor on the left and a help window on the right. The code editor contains the following content:

```
[1] 14 # Normalization through MinMaxScaler
    15 from sklearn.preprocessing import MinMaxScaler
    16
    17 from sklearn.pipeline import Pipeline
```

Clustering of digits

We will use digit dataset for clustering, which is loaded through `load_digits` API.

- It loads `8x8` digit images with approximately 180 samples per class.
- From 10 classes, it has total of 1797 images.
- Each pixel has value between 0 and 16.

```
[2] 1 digits = load_digits()
```

Let's quickly check `KMeans` class as implemented in `sklearn.cluster` module.

```
1 KMeans
```

Some of the important parameters are as follows:

- `init`
- `n_init`
- `max_iter`
- `random_state`

Since `KMeans` algorithm is susceptible to local minima, we perform multiple `KMeans` fit and select the ones with the lowest value of sum of squared error.

The help window on the right shows the documentation for `KMeans`:

Init signature: `KMeans(*args, **kwargs)`
Docstring:
K-Means clustering.

Read more in the :ref:`User Guide <k_means>`.

Parameters

`n_clusters` : int, default=8
The number of clusters to form as well as the number of centroids to generate.

`init` : {'k-means++', 'random'}, callable or array-like of shape (n_clusters, n_features), default='k-means++'
Method for initialization:

- 'k-means++': selects initial cluster centers for k-mean clustering in a smart way to speed up convergence. See section Notes in `K_init` for more details.
- 'random': choose 'n_clusters' observations (rows) at random from data for the initial centroids.

If an array is passed, it should be of shape (n_clusters, n_features) and gives the initial centers.

If a callable is passed, it should take arguments `X`, `n_clusters` and a random state and return an initialization.

The screenshot displays a Jupyter Notebook interface with two cells. The first cell contains code for normalization and pipeline import:

```
[1] 14 # Normalization through MinMaxScaler
    15 from sklearn.preprocessing import MinMaxScaler
    16
    17 from sklearn.pipeline import Pipeline
```

The second cell is titled "Clustering of digits" and contains explanatory text and code:

```
[2] 1 digits = load_digits()
```

Below the code, the notebook lists important parameters for the `KMeans` class:

- `init`
- `n_init`
- `max_iter`
- `random_state`

The right-hand pane shows the documentation for the `KMeans` class, detailing parameters such as `n_clusters`, `n_init`, `max_iter`, `tol`, `verbose`, `random_state`, `copy_x`, and `algorithm`.

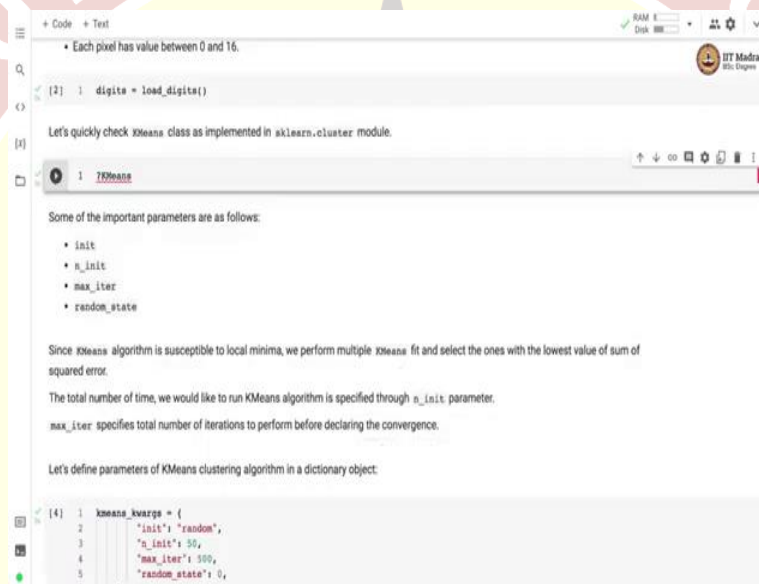
We will be using the digit dataset for clustering which is loaded through `load_digit` API. It loads 8×8 digit images with approximately 180 samples per class. There are 10 classes and it has got in all 1797 images. Each pixel has value between 0 and 16. Let us quickly check the k-means class as implemented at `sklearn.cluster` module. So, you can see the documentation of k-means class by calling k-means by prefixing it with the ?.

So, you can see that in k-means it takes parameters like number of clusters to form, then there is initialization parameter which could be `k-means++` or `random` `k-means++` selects initial cluster centers for k-means in a smart way to speed up the convergence, whereas random basically

initializes the cluster centers randomly. Then there is `n_init` parameter, which shows the number of times the k-means algorithm will be run with different centroid seeds.

The final result will be the best output of the different runs in terms of the sum of squared error and `max_iter` specifies the maximum number of iterations that we want to run the single k-means algorithm for and finally, `random_state` that determines the random number generation for centroid initialization.

(Refer Slide Time: 02:40)



The screenshot shows a Jupyter Notebook interface with a code cell containing the following text and code:

```
[2]: 1 digit = load_digits()

Let's quickly check KMeans class as implemented in sklearn.cluster module.

1 KMeans

Some of the important parameters are as follows:

• init
• n_init
• max_iter
• random_state

Since KMeans algorithm is susceptible to local minima, we perform multiple KMeans fit and select the ones with the lowest value of sum of squared error.

The total number of time, we would like to run KMeans algorithm is specified through n_init parameter.

max_iter specifies total number of iterations to perform before declaring the convergence.

Let's define parameters of KMeans clustering algorithm in a dictionary object:

[4]: 1 kmeans_kwargs = {
2     "init": "random",
3     "n_init": 50,
4     "max_iter": 500,
5     "random_state": 0,
```

Since k-means algorithm is susceptible to local minima, we perform multiple k-means run and select the one with the lowest sum of squared error. The total number of time we would like to run k-means algorithm is specified through `n_init` parameter. `max_iter` specifies the total number of iterations to perform before declaring convergence in any of these runs.

(Refer Slide Time: 03:04)



```
+ Code + Text
The total number of time, we would like to run KMeans algorithm is specified through n_init parameter.
max_iter specifies total number of iterations to perform before declaring the convergence.

Let's define parameters of KMeans clustering algorithm in a dictionary object:

[4] 1 kmeans_kwargs = {
2     'init': 'random',
3     'n_init': 50,
4     'max_iter': 500,
5     'random_state': 0,
6 }

Let's define a pipeline with two stages:
• preprocessing for feature scaling with MinMaxScaler.
• clustering with KMeans clustering algorithm.

[5] 1 pipeline = Pipeline([('Preprocess', MinMaxScaler()),
2     ('Clustering', KMeans(n_clusters=10, **kmeans_kwargs))])
3 pipeline.fit(digits.data)

Pipeline(steps=[('Preprocess', MinMaxScaler()),
                 ('Clustering',
                  KMeans(init='random', max_iter=500, n_clusters=10, n_init=50,
                        random_state=0))])

The cluster centroids can be accessed via cluster_centers_ member variable of KMeans class.

[6] 1 cluster_centers = pipeline[-1].cluster_centers_
```

Let us first define all the k-means clustering algorithm parameter in a dictionary object. We will be using random initialization. We will be running k-means again, and again for 50 times, and in each iteration, we will be running k-means for 500 iterations, and random_state is set to 0. Let us define a pipeline with 2 stages. The first one is pre-processing for future scaling with MinMaxScaler. And second is clustering with k-means clustering algorithm.

Here we have selected number of clusters = 10. And we have specified the other arguments for k-means through this dictionary object. We call the fit function on pipeline with the digit dataset.

(Refer Slide Time: 03:52)

```
+ Code + Test
• preprocessing for feature scaling with MinMaxScaler.
• clustering with KMeans clustering algorithm.

1 pipeline = Pipeline([("Preprocess", MinMaxScaler()),
2 ("Clustering", KMeans(n_clusters=10, **kmeans_kwargs))])
3 pipeline.fit(digits.data)

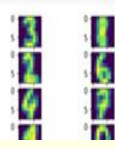
4 Pipeline(steps=[("Preprocess", MinMaxScaler()),
5 ("Clustering",
6 KMeans(init='random', max_iter=500, n_clusters=10, n_init=50,
7 random_state=0))])

The cluster centroids can be accessed via cluster_centers_ member variable of KMeans class.

8 cluster_centers = pipeline[-1].cluster_centers_

Let's display cluster centroids:

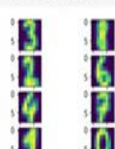
9 # displaying centroids
10 fig, ax = plt.subplots(5, 2, figsize=(4, 4))
11 for i,j in zip(ax.flat, cluster_centers.reshape(10,8,8)):
12     i.imshow(j)
```



```
+ Code + Test
8 cluster_centers = pipeline[-1].cluster_centers_

Let's display cluster centroids:

9 # displaying centroids
10 fig, ax = plt.subplots(5, 2, figsize=(4, 4))
11 for i,j in zip(ax.flat, cluster_centers.reshape(10,8,8)):
12     i.imshow(j)
```



In this case, the number of clusters were known. Hence we set $k=10$ and got the clusters.

For deciding the optimal number of clusters through elbow and silhouette, we will pretend that we do not know the number of clusters in the data and we will try to discover the optimal number of clusters through these two methods one by one:

- Elbow method

We can access the cluster centroids after training the k-means clustering algorithm. We can access them via `cluster_centers_` member variable of k-means class. So, k-means class is available at the last stage of pipelines. That is why it is `pipeline[-1].cluster_centers_`, give us the cluster centers. We display this cluster centroids in form of the digit images, as you can see over here. So, in this case, the number of clusters were known, hence, we set $K = 10$ and got these clusters.

(Refer Slide Time: 04:39)



```
[8] 1 #Identifying the correct number of clusters
2 sse_digit = []
3
4 scaled_digits = MinMaxScaler().fit_transform(digits.data)
5 for k in range(1,12):
6     kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
7     kmeans.fit(scaled_digits)
8     sse_digit.append(kmeans.inertia_)
```

Note that the SSE for a given clustering output is obtained through inertia_ member variable.

```
[9] 1 plt.plot(range(1, 12), sse_digit)
2 plt.xticks(range(1, 12))
3 plt.xlabel('Number of Clusters')
4 plt.ylabel('SSE')
5 plt.show()
```

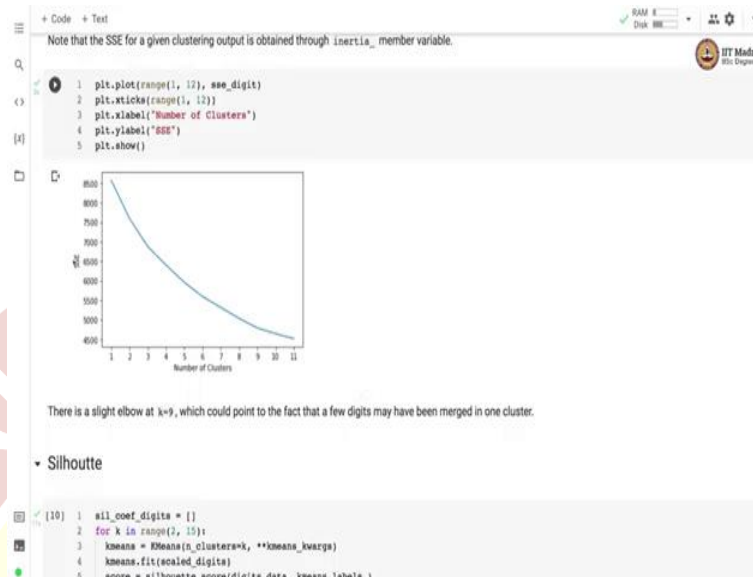
The plot shows a line graph where the x-axis is 'Number of Clusters' (ranging from 1 to 12) and the y-axis is 'SSE' (ranging from 8000 to 8500). The line starts at approximately 8450 for 1 cluster and decreases sharply to about 8050 for 2 clusters, then continues to decrease more gradually, reaching approximately 8000 for 12 clusters.

For deciding the optimal number of clusters, we generally use Elbow method and silhouette method. So, we basically trained the elbow and silhouette method and find out the actual number of K that comes out of these 2 methods. Here, for some time, we will pretend that we do not know the number of clusters and this will usually be the case when you are doing clustering in real life world, where we will not be knowing the total number of clusters, and we are supposed to come up with the optimal number of K.

And for that we use elbow and silhouette as 2 methods, so one of the 2 methods you need to employ. So, let us see how to use Elbow method. So, in Elbow method, what we do is we keep track of sum of squared error in a list. And here what we do is we first scale the digits by applying MinMaxScaler and this scale digits are used for clustering with k-means clustering algorithm, but number of clusters now varies in the range between 1-12.

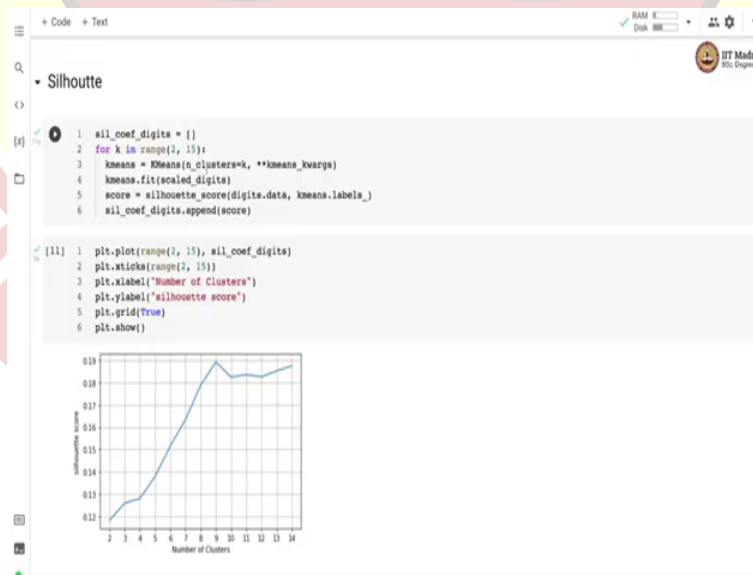
And the sum of squared error that we obtained for different values of K is appended to this particular list. And sum of squared error is obtained by accessing inertia_ member variable of the k-means object.

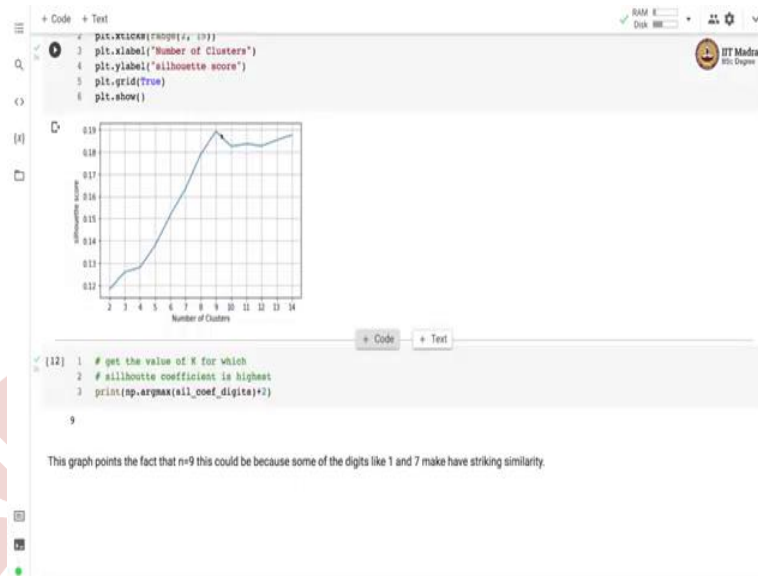
(Refer Slide Time: 06:01)



Now, you can see that, we have plotted the sum of square error against the number of clusters. So, we have number of clusters on x-axis and sum of square error on y-axis, and you can see that there is a slight elbow at $K = 9$. And this could point to the fact that few digits may have been merged in a single cluster.

(Refer Slide Time: 06:25)





The second method is a silhouette method and in silhouette method we basically calculate the silhouette _score based on the data and the labels that we obtained to k-means clustering algorithm. So, here again, we trained a k-means clustering algorithm with different number of clusters between in the range between 2-15 and then we calculate silhouette _score for each clustering solution. And the silhouette _score is appended to the list.

So, this graph also points the fact that $K = 9$, maybe the optimal number of clusters, which was also revealed by Elbow Method. And this could happen because some digits like 1 and 7 have striking similarity, and they have been merged in the single cluster. So, this was a short demonstration of how to use k-means clustering algorithm and find out optimal value of K through elbow and silhouette methods.