

DELHI TECHNOLOGICAL UNIVERSITY

[Formerly Delhi College of Engineering]

2018-2022



PATTERN RECOGNITION (CO324)

**Dept. Of Computer Science And
Engineering**

ASSIGNMENT 1

Prepared by :

Yash Gandhi 2K18/CO/402

Ashi Gupta 2K18/SE/040

Submitted to : Dr. Anurag Goel

KNN ALGORITHM

K-nearest neighbors (KNN) algorithm is a type of supervised ML algorithm which can be used for both classification as well as regression predictive problems. However, it is mainly used for classification predictive problems in industry. The following two properties would define KNN well –

- **Lazy learning algorithm** – KNN is a lazy learning algorithm because it does not have a specialized training phase and uses all the data for training while classification.
- **Non-parametric learning algorithm** – KNN is also a non-parametric learning algorithm because it doesn't assume anything about the underlying data.

Working of KNN Algorithm

K-nearest neighbors (KNN) algorithm uses 'feature similarity' to predict the values of new datapoints which further means that the new data point will be assigned a value based on how closely it matches the points in the training set. We can understand its working with the help of following steps –

Step 1 – For implementing any algorithm, we need dataset. So during the first step of KNN, we must load the training as well as test data.

Step 2 – Next, we need to choose the value of K i.e. the nearest data points. K can be any integer.

Step 3 – For each point in the test data do the following –

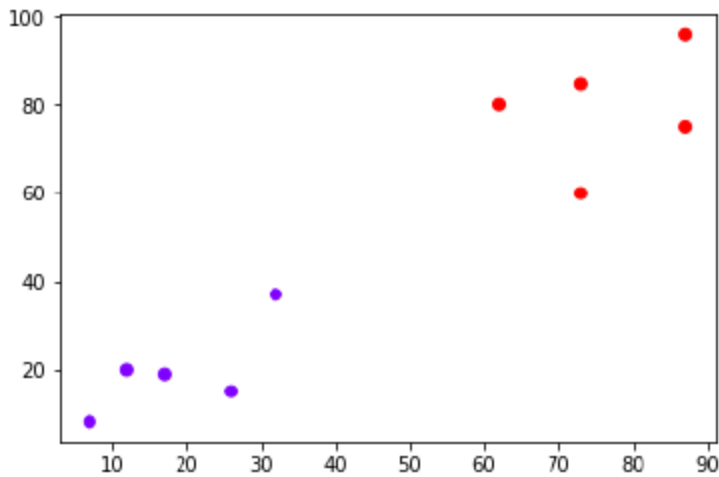
- **3.1** – Calculate the distance between test data and each row of training data with the help of any of the method namely: Euclidean, Manhattan or Hamming distance. The most commonly used method to calculate distance is Euclidean.
- **3.2** – Now, based on the distance value, sort them in ascending order.
- **3.3** – Next, it will choose the top K rows from the sorted array.
- **3.4** – Now, it will assign a class to the test point based on most frequent class of these rows.

Step 4 – End

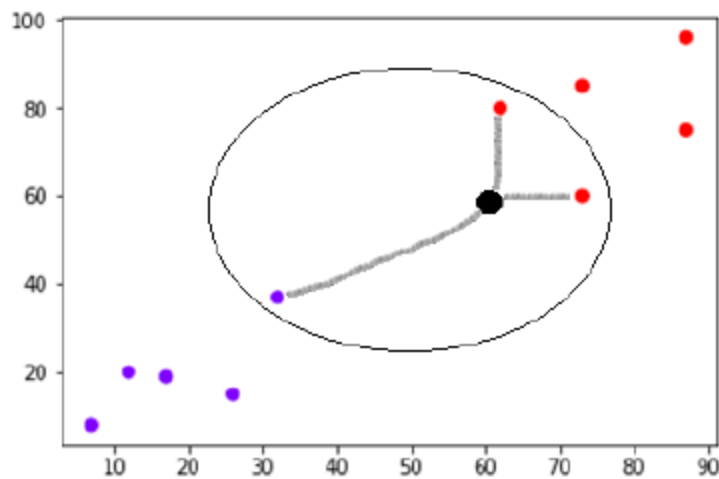
Example

The following is an example to understand the concept of K and working of KNN algorithm –

Suppose we have a dataset which can be plotted as follows –



Now, we need to classify new data point with black dot (at point 60,60) into blue or red class. We are assuming $K = 3$ i.e. it would find three nearest data points. It is shown in the next diagram –



We can see in the above diagram the three nearest neighbors of the data point with black dot. Among those three, two of them lies in Red class hence the black dot will also be assigned in red class.

K Means ALGORITHM

K-means clustering algorithm computes the centroids and iterates until we it finds optimal centroid. It assumes that the number of clusters are already known. It is also called **flat clustering** algorithm. The number of clusters identified from data by algorithm is represented by 'K' in K-means.

In this algorithm, the data points are assigned to a cluster in such a manner that the sum of the squared distance between the data points and centroid would be minimum. It is to be understood that less variation within the clusters will lead to more similar data points within same cluster.

Working of K-Means Algorithm

We can understand the working of K-Means clustering algorithm with the help of following steps –

Step 1 – First, we need to specify the number of clusters, K, need to be generated by this algorithm.

Step 2 – Next, randomly select K data points and assign each data point to a cluster. In simple words, classify the data based on the number of data points.

Step 3 – Now it will compute the cluster centroids.

Step 4 – Next, keep iterating the following until we find optimal centroid which is the assignment of data points to the clusters that are not changing any more

- **4.1** – First, the sum of squared distance between data points and centroids would be computed.
- **4.2** – Now, we have to assign each data point to the cluster that is closer than other cluster (centroid).
- **4.3** – At last compute the centroids for the clusters by taking the average of all data points of that cluster.

K-means follows **Expectation-Maximization** approach to solve the problem. The Expectation-step is used for assigning the data points to the closest cluster and the Maximization-step is used for computing the centroid of each cluster.

While working with K-means algorithm we need to take care of the following things –

- While working with clustering algorithms including K-Means, it is recommended to standardize the data because such algorithms use distance-based measurement to determine the similarity between data points.
- Due to the iterative nature of K-Means and random initialization of centroids, K-Means may stick in a local optimum and may not converge to global optimum. That is why it is recommended to use different initializations of centroids.

QUESTION 1:

Dataset Description:

The database consists of the multi-spectral values of pixels in 3x3 neighbourhoods in a satellite image, and the classification associated with the central pixel in each neighbourhood. The aim is to predict this classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

The Landsat satellite data is one of the many sources of information available for a scene. One frame of Landsat MSS imagery consists of four digital images of the same scene in different spectral bands. Two of these are in the visible region (corresponding approximately to green and red regions of the visible spectrum) and two are in the (near) infra-red. Each pixel is a 8-bit binary word, with 0 corresponding to black and 255 to white. The spatial resolution of a pixel is about 80m x 80m. Each image contains 2340 x 3380 such pixels.

The database is a (tiny) sub-area of a scene, consisting of 82 x 100 pixels. Each line of data corresponds to a 3x3 square neighborhood of pixels completely contained within the 82x100 sub-area. Each line contains the pixel values in the four spectral bands (converted to ASCII) of each of the 9 pixels in the 3x3 neighborhood and a number indicating the classification label of the central pixel. The number is a code for the following classes:

Data Set Characteristics:	Multivariate	Number of Instances:	6435	Area:	Physical
Attribute Characteristics:	Integer	Number of Attributes:	36	Date Donated	1993-02-13
Associated Tasks:	Classification	Missing Values?	N/A	Number of Web Hits:	142648

Number Class

- 1 red soil
- 2 cotton crop
- 3 grey soil
- 4 damp grey soil
- 5 soil with vegetation stubble
- 6 mixture class (all types present)
- 7 very damp grey soil

Attribute Information:

The attributes are numerical, in the range 0 to 255.

Methodology:

The problem involves building a KNN classifier from scratch for the Landsat satellite dataset, in which the optimal value of k is calculated using grid search algorithm and then that value of k parameter is used to find the accuracy of the model and the classification report. Finally this model is tested with standard sklearn library's KNN model and then we analyze the results obtained.

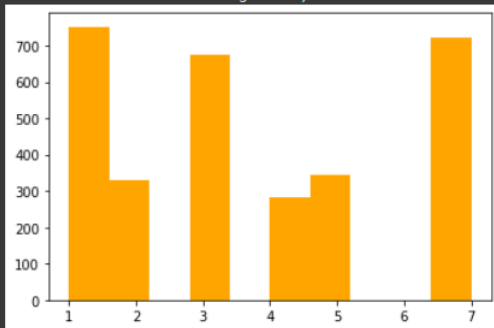
Train and Validation split:

Dataset is given in train and test files for training and testing the ratio used is 70:30 for training and validation sets. The train data is further split into training and validation set as shown:

```
[ ] x_train, x_val, y_train, y_val = train_test_split(train[train.columns[:-1]].to_numpy(), train[train.columns[-1]].to_numpy(), test_size=0.30)
```

```
[ ] plt.hist(y_train,color='orange')
```

```
(array([752., 329., 0., 673., 0., 283., 345., 0., 0., 722.]),  
 array([1. , 1.6, 2.2, 2.8, 3.4, 4. , 4.6, 5.2, 5.8, 6.4, 7. ]),  
 <a list of 10 Patch objects>)
```



Finding optimal value of k :

Grid search is used to find the optimum value of k in which there are 2 functions which return accuracy for a particular k value when prediction is true by calculating the mean value And second one returns optimal value of k from a given range of values. Both are implemented from scratch.

```
# return accuracy for a particular k value when prediction is true by calculating the mean value
```

```
def get_acc(prediction, y_true):  
    N1 = prediction.shape[0]  
    accuracy = np.sum(prediction == y_true) / N1  
    return accuracy
```

```

# returns optimal value of k from a given range of values
def kselector(x_val, y_val, x_train, y_train, k_values):
    accuracies = []

    for k in tqdm(k_values):
        prediction = predict(x_val, x_train, y_train, k)

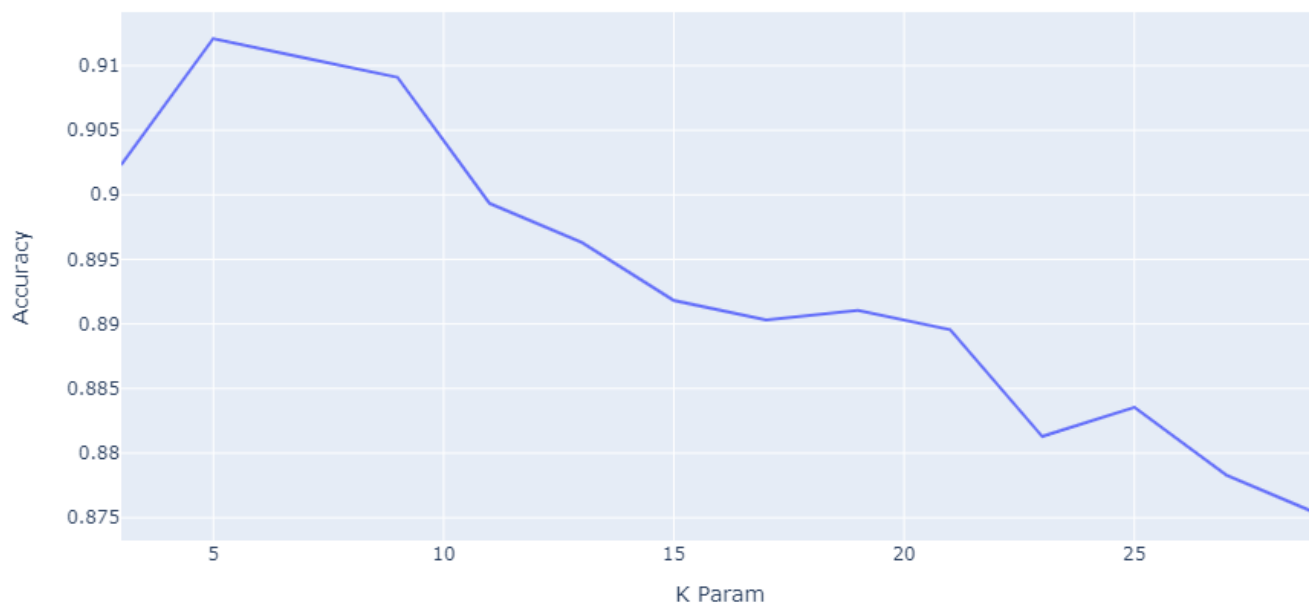
        acc = get_acc(prediction, y_val)
        accuracies.append(acc)

    k_optimal = k_values[accuracies.index(max(accuracies))]

    return k_optimal, accuracies

```

Accuracy for k nearest neighbors



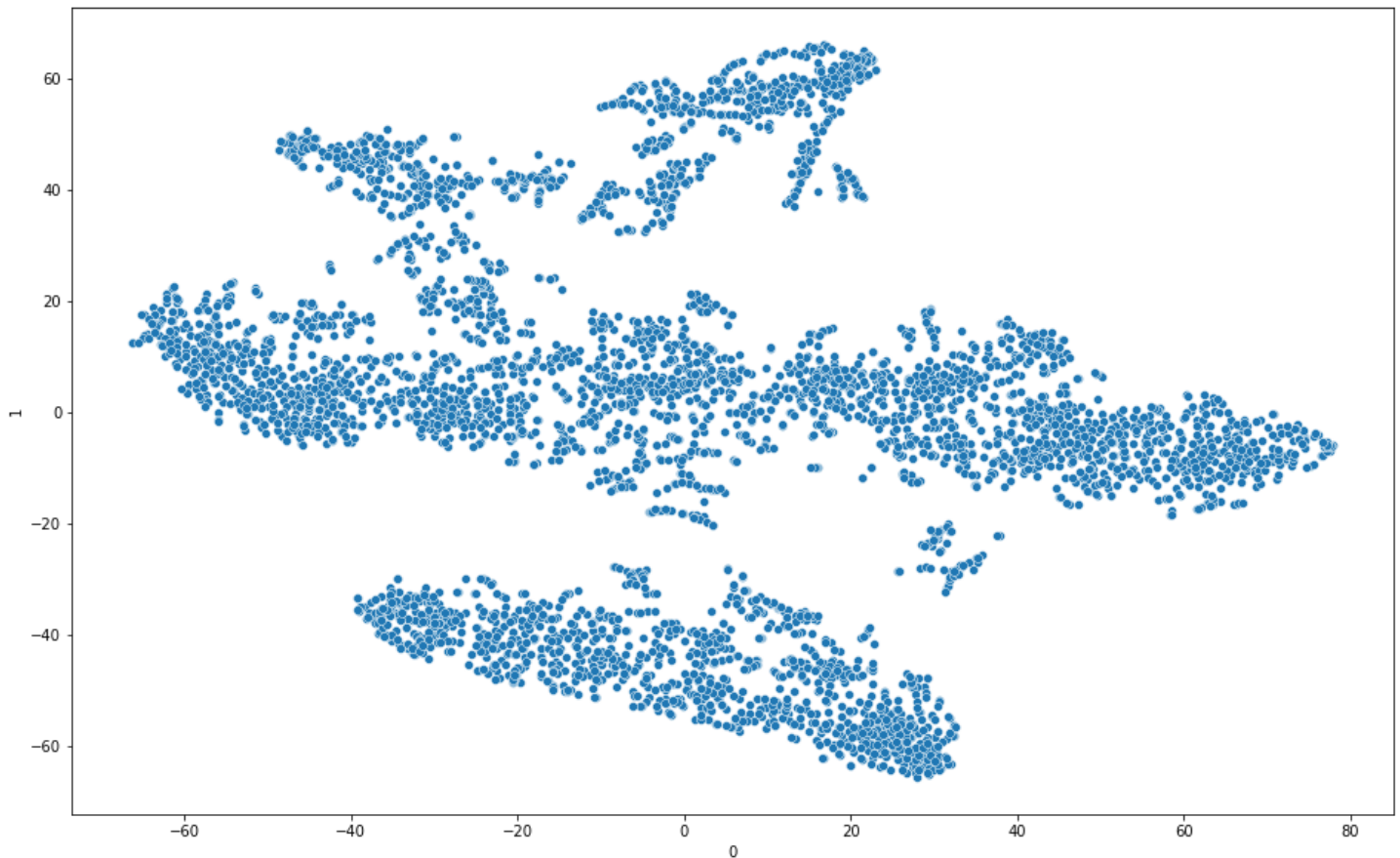
The graph shows the variation of model accuracy with the changing parameter k. Here the low value is chosen to be 3. A small value of **k** means that noise will have a **higher** influence on the result and a **large** value make it computationally expensive. Thus we can see that the optimum value of k here is 5 as it's giving highest accuracy.

TSNE Plot:

The TSNE plot is similar to a PCA, but with some important differences. The TSNE plot was made as shown and it was plotted using scatter plot module.

```
tsne = TSNE(n_components=2, perplexity=40, random_state=4).fit_transform(train)
```

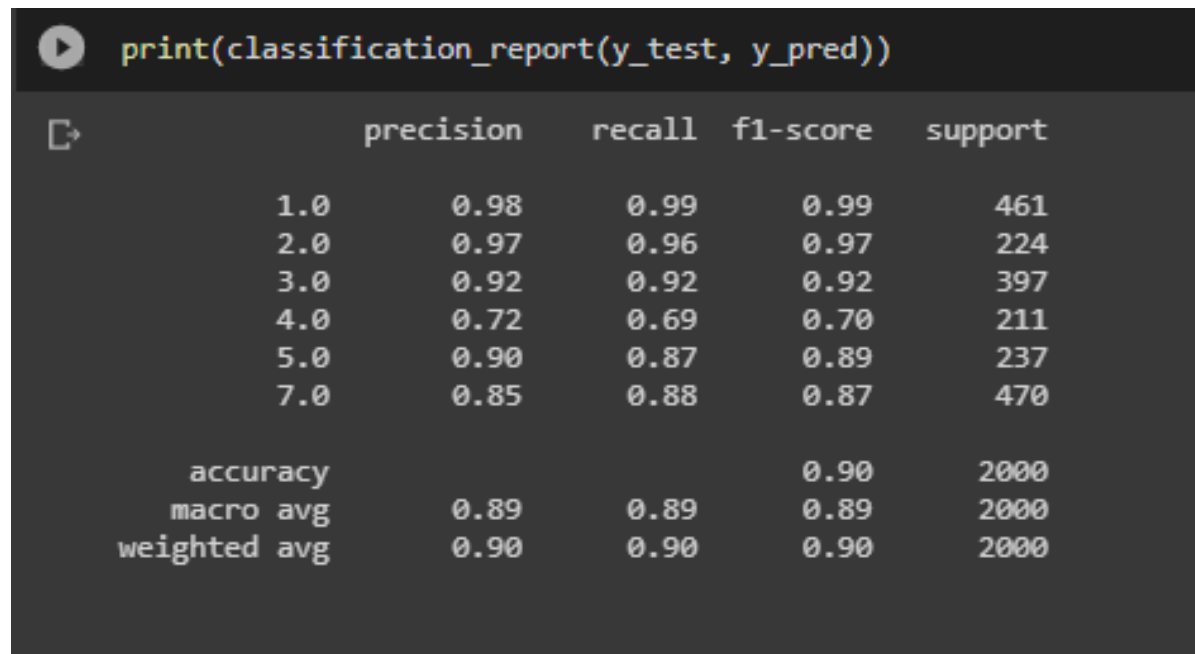
```
plt.figure(figsize=(16,10))  
# colorsIdx = {'A': 'rgb(215,48,39)', 'B': 'rgb(215,148,39)'}  
# cols      = ['orange', 'blue']  
sns.scatterplot(  
    x=0, y=1,  
    palette=sns.color_palette("hls", 10),  
    data=tsne_df,  
    legend="full",  
    alpha=1  
)
```



Classification report and results:

The accuracy of our model comes out to be **0.9012 for Testing and for 0.906 Validation** whereas the accuracy for standard sklearn's kNN model is **0.925 for Testing and for 0.9113 Validation** for the same dataset. Which indicates that our model performed well in classification when testing the model for the optimum k value to be 5.

Similarly, training the model for a different random state and slightly changing some related parameters yielded a different optimum value of k and for k=3. The testig accuracy in that case comes out to be 0.932.

A screenshot of a Jupyter Notebook cell. At the top, there is a play button icon followed by the code `print(classification_report(y_test, y_pred))`. Below the code, the output is a classification report table. The table has five columns: 'precision', 'recall', 'f1-score', and 'support'. The rows represent different classes (1.0, 2.0, 3.0, 4.0, 5.0, 7.0) and summary metrics (accuracy, macro avg, weighted avg).

	precision	recall	f1-score	support
1.0	0.98	0.99	0.99	461
2.0	0.97	0.96	0.97	224
3.0	0.92	0.92	0.92	397
4.0	0.72	0.69	0.70	211
5.0	0.90	0.87	0.89	237
7.0	0.85	0.88	0.87	470
accuracy			0.90	2000
macro avg	0.89	0.89	0.89	2000
weighted avg	0.90	0.90	0.90	2000

Classification report for our model

```
print(classification_report(y_test, y_pred_skl))
```

	precision	recall	f1-score	support
1.0	0.98	0.99	0.99	461
2.0	0.97	0.96	0.96	224
3.0	0.89	0.92	0.91	397
4.0	0.70	0.65	0.67	211
5.0	0.91	0.87	0.89	237
7.0	0.85	0.86	0.86	470
accuracy			0.89	2000
macro avg	0.88	0.88	0.88	2000
weighted avg	0.89	0.89	0.89	2000

```
[ ] print("Accuracy for the model from sklearn is",accuracy_score(y_test, y_pred_skl))  
    print("Accuracy for the model from scratch is",accuracy_score(y_test, y_pred))
```

```
Accuracy for the model from sklearn is 0.894  
Accuracy for the model from scratch is 0.9015
```

Classification Report for sklearn

```
print("Testing Accuracy for sklearn:",accuracy_score(y_test, y_pred_skl))  
print("Testing Accuracy for our model:",accuracy_score(y_test, y_pred))  
print("\n")  
print("Validation Accuracy for sklearn:",accuracy_score(y_val, y_valPred_skl))  
print("Validation Accuracy for our model:",accuracy_score(y_val, y_valPred))
```

```
Testing Accuracy for sklearn: 0.897  
Testing Accuracy for our model: 0.8975
```

```
Validation Accuracy for sklearn: 0.9113448534936138  
Validation Accuracy for our model: 0.9060856498873028
```

QUESTION 2:

Dataset Description:

The Iris Dataset contains four features (length and width of sepals and petals) of 50 samples of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). The rows being the samples and the columns being: Sepal Length, Sepal Width, Petal Length and Petal Width.

The dataset is often used in data mining, classification and clustering examples and to test algorithms.



Iris Versicolor



Iris Setosa



Iris Virginica

Attribute Information:

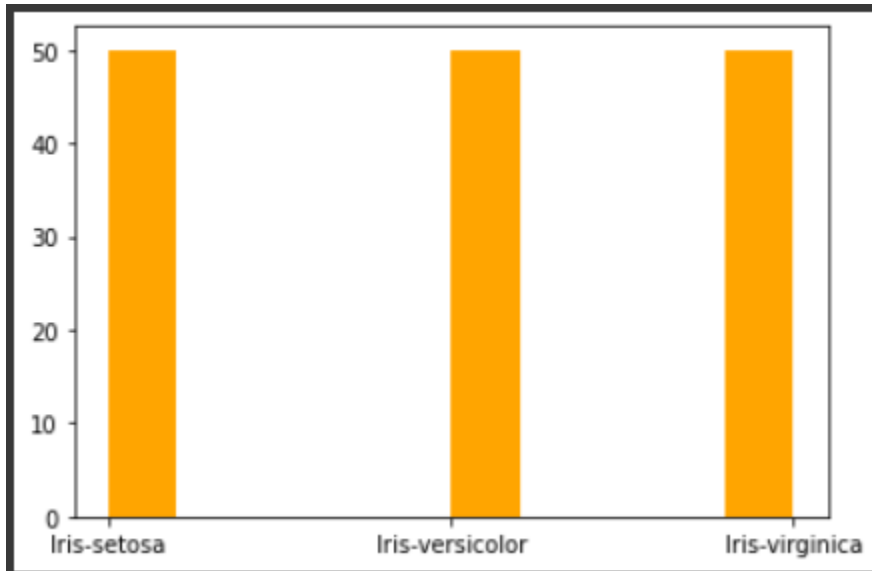
1. sepal length in cm
2. sepal width in cm
3. petal length in cm
4. petal width in cm
5. class:
 - Iris Setosa
 - Iris Versicolour
 - Iris Virginica

Data Set Characteristics:	Multivariate	Number of Instances:	150	Area:	Life
Attribute Characteristics:	Real	Number of Attributes:	4	Date Donated	1988-07-01
Associated Tasks:	Classification	Missing Values?	No	Number of Web Hits:	3917681

Train and Validation split:

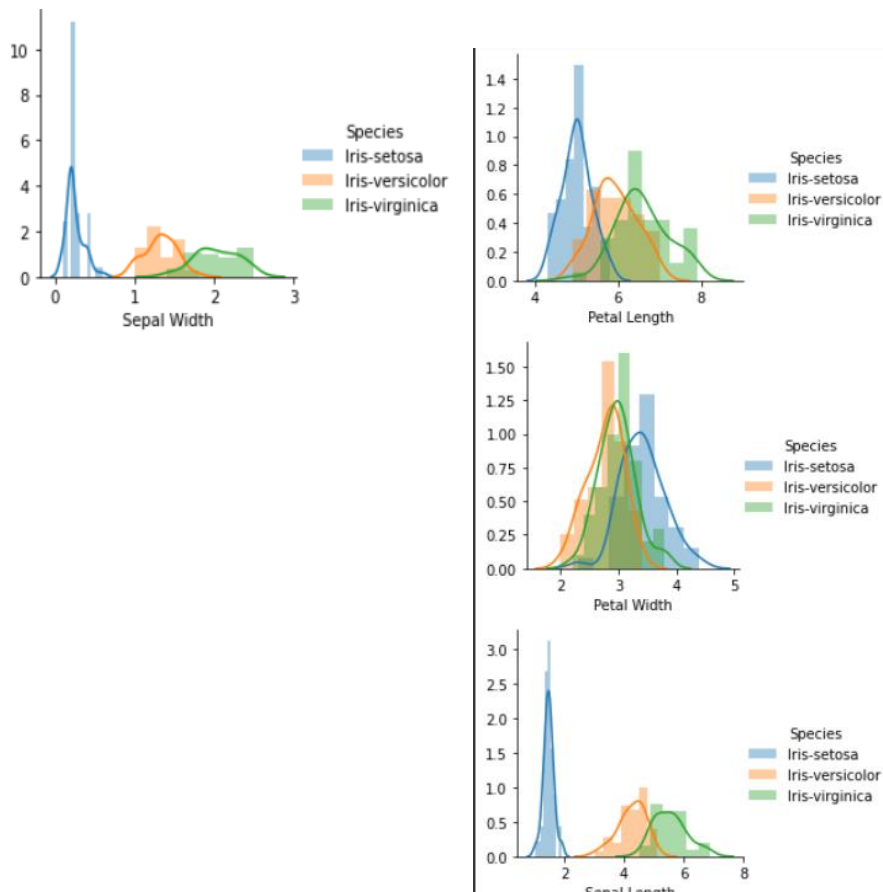
Dataset is given in train and test files for training and testing the ratio used is 70:30 for training and validation sets. The train data is further split into training and validation set as shown:

```
plt.hist(dataFrame["Species"],color="Orange")  
plt.show()
```



Distribution plots : Plotting each flower to a histogram

```
import seaborn as sns
sns.FacetGrid(dataFrame,hue="Species",size=3).map(sns.distplot,"Petal Length").add_legend()
sns.FacetGrid(dataFrame,hue="Species",size=3).map(sns.distplot,"Petal Width").add_legend()
sns.FacetGrid(dataFrame,hue="Species",size=3).map(sns.distplot,"Sepal Length").add_legend()
sns.FacetGrid(dataFrame,hue="Species",size=3).map(sns.distplot,"Sepal Width").add_legend()
plt.show()
```



Finding the optimum number of clusters for k-means classification

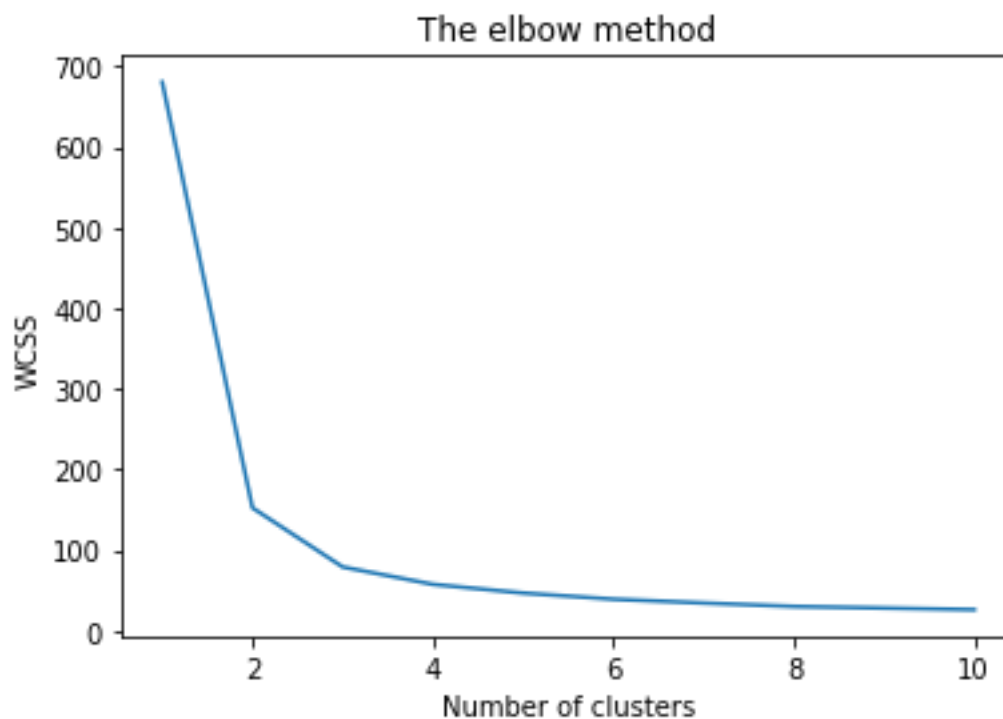
```
from sklearn.cluster import KMeans
wcss = []

for i in range(1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
    kmeans.fit(attributes)
    wcss.append(kmeans.inertia_)
```

Using the elbow method to determine the optimal number of clusters for k-means clustering

```
#Plotting the results onto a line graph, allowing us to observe 'The elbow'
plt.plot(range(1, 11), wcss)
plt.title('The elbow method')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS') #within cluster sum of squares
plt.show()
```

The elbow method allows us to pick the optimum amount of clusters for classification



To determine the optimal number of clusters, we have to select the value of k at the “elbow” i.e. the point after which the distortion/inertia start decreasing in a linear fashion. In the above graph, it is when the within cluster sum of squares (WCSS) doesn't decrease significantly with every iteration. Now we have the optimum amount of clusters, we can move on to applying K-means clustering to the Iris dataset. Considering the point of occurring of the elbow to be 3, i.e. **3 clusters will be the optimum solution for this problem.**

Implementing K-Means Clustering

```
#Applying kmeans to the dataset / Creating the kmeans classifier
kmeans = KMeans(n_clusters = 3, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_kmeans = kmeans.fit_predict(x_train)
```

Visualising the clusters

We'll calculate three clusters, get their centroids, and set some colors.

```
#Visualising the clusters
plt.scatter(x_train[y_kmeans == 0, 0], x_train[y_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(x_train[y_kmeans == 1, 0], x_train[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')
plt.scatter(x_train[y_kmeans == 2, 0], x_train[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

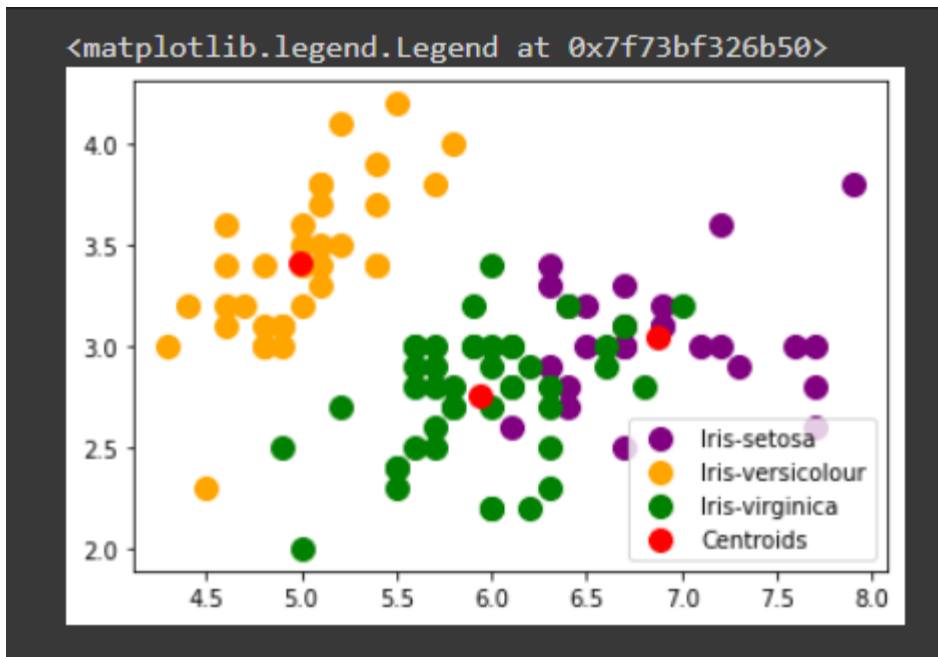
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s = 100, c = 'red', label = 'Centroids')

plt.legend()
```

Then we can pass the fields we used to create the cluster to Matplotlib's scatter and use the 'c' column we created to paint the points in our chart according to their cluster.

```
#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:, 1], s = 100, c = 'red', label = 'Centroids')

plt.legend()
```

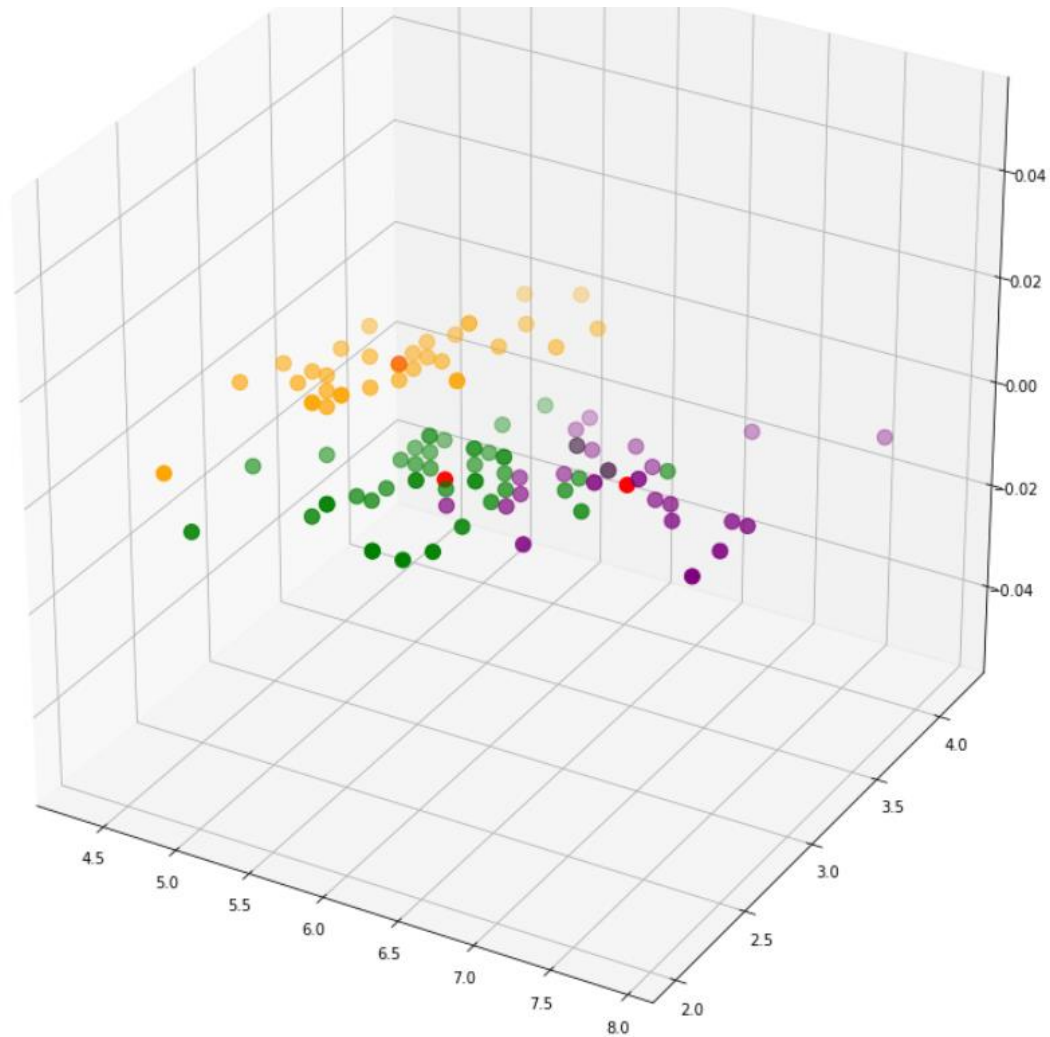


3d scatter plot using matplotlib

```
[ ] # 3d scatterplot using matplotlib

fig = plt.figure(figsize = (15,15))
ax = fig.add_subplot(111, projection='3d')
plt.scatter(x_train[y_kmeans == 0, 0], x_train[y_kmeans == 0, 1], s = 100, c = 'purple', label = 'Iris-setosa')
plt.scatter(x_train[y_kmeans == 1, 0], x_train[y_kmeans == 1, 1], s = 100, c = 'orange', label = 'Iris-versicolour')
plt.scatter(x_train[y_kmeans == 2, 0], x_train[y_kmeans == 2, 1], s = 100, c = 'green', label = 'Iris-virginica')

#Plotting the centroids of the clusters
plt.scatter(kmeans.cluster_centers[:, 0], kmeans.cluster_centers[:,1], s = 100, c = 'red', label = 'Centroids')
plt.show()
```

Model accuracy check and Classification report

Converting true labels to their respective class numbers:

- Virginica Class 1
- Setosa Class 0
- Versicolor Class 2

Classification report for training set :

	precision	recall	f1-score	support
Iris-setosa	1.00000	1.00000	1.00000	35
Iris-versicolor	0.75000	0.94286	0.83544	35
Iris-virginica	0.92308	0.68571	0.78689	35
accuracy			0.87619	105
macro avg	0.89103	0.87619	0.87411	105
weighted avg	0.89103	0.87619	0.87411	105

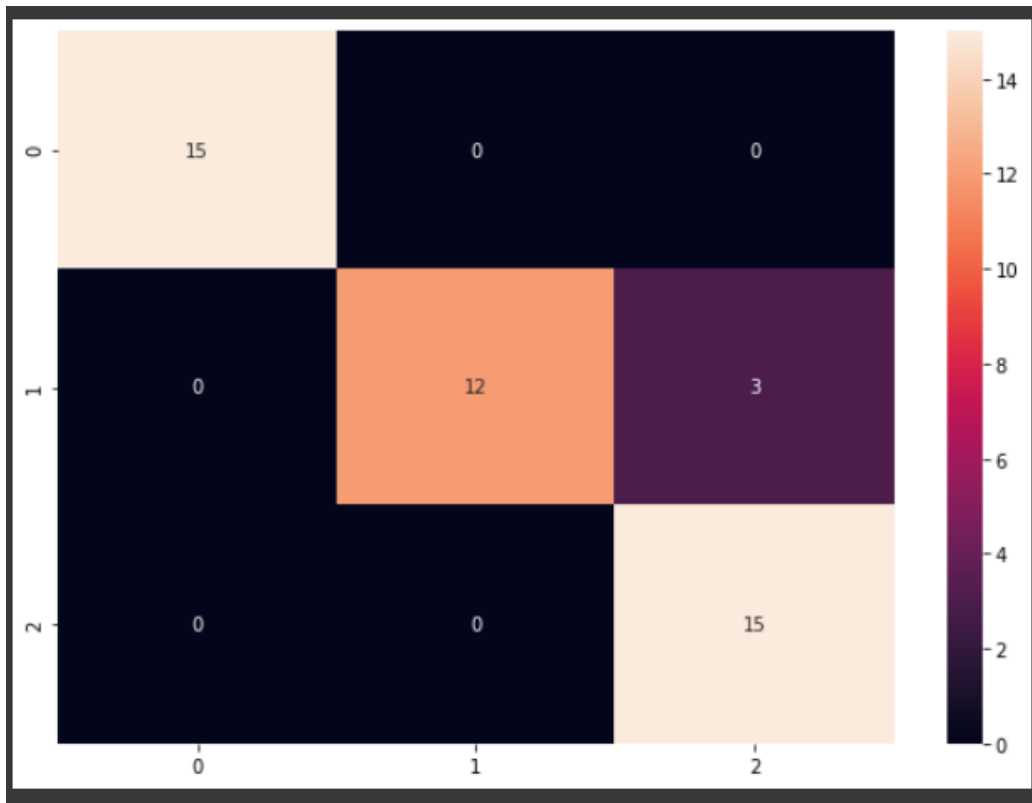
The accuracy for training set is 0.8761

Classification report for validation set :

	precision	recall	f1-score	support
Iris-setosa	1.00000	1.00000	1.00000	15
Iris-versicolor	1.00000	0.80000	0.88889	15
Iris-virginica	0.83333	1.00000	0.90909	15
accuracy			0.93333	45
macro avg	0.94444	0.93333	0.93266	45
weighted avg	0.94444	0.93333	0.93266	45

The accuracy for validation set is 0.9333

Following is the confusion matrix:



Yash Gandhi 2K18/CO/402

Ashi Gupta 2K18/SE/040

