University of Pittsburgh

ECE 2195: Special Topics – Computers Machine Learning

Implementation Workflow and K-Nearest Neighbor Classification in Python

Mai Abdelhakim, PhD
ECE Department
Swanson School of Engineering
University of Pittsburgh
maia@pitt.edu

Revisit K-Nearest Neighbor (KNN)

- Store the training set

- Prediction for a new data: algorithm finds K points in the training set that are nearest to the new data point

- Then make prediction using **majority** class among the neighbors

  - In an exercise problem, you will need to implement without using built-in functions

Implementation Steps

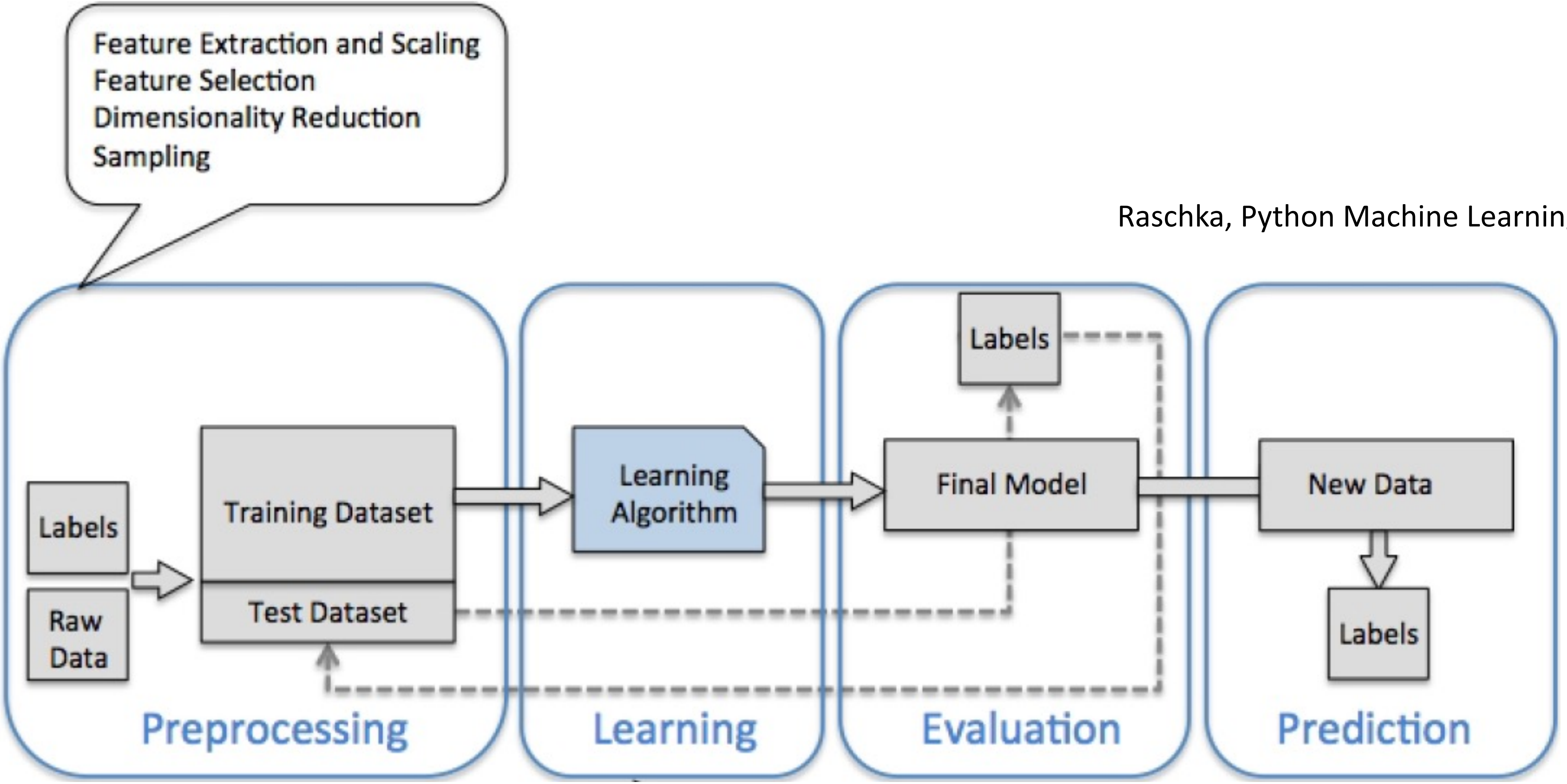
- Step 1: Get the data, and represent information by features (feature extraction) --- *preprocessing*

- Step 2: Split the Data to Training and Test Set (*Preprocessing step*)

- Step 3: Define your Model

- Step 4: Fit (Train) your Model using training data (*Learning*)
  Cross validate (later)

- Step 5: Performance Evaluation using test data (*Evaluation*)
  --You can then use for *prediction*



Raschka, Python Machine Learning

Example: Classify Iris Species

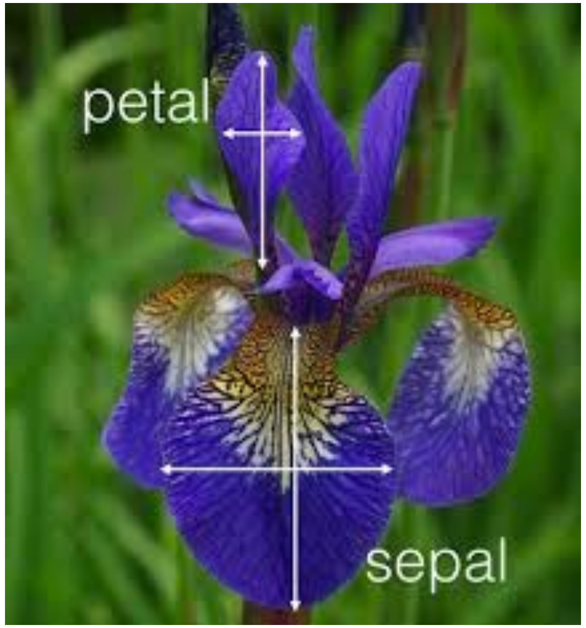
- Goal: Distinguish species of different Iris flowers
  - Species: (0) Setosa, (1) Versicolor, (2) Virginica

- Features:
  - length and width of sepal
  - Width and length of petal

- From measurements of Iris flowers whose species are known, develop a model that predict the species with new measurements

- **Supervised learning:** we know correct species of training data – called **label**

- **Classification problem**



Step 1: Get the Data

- Classical dataset in machine learning, included in Scikit-learn – **datasets module**

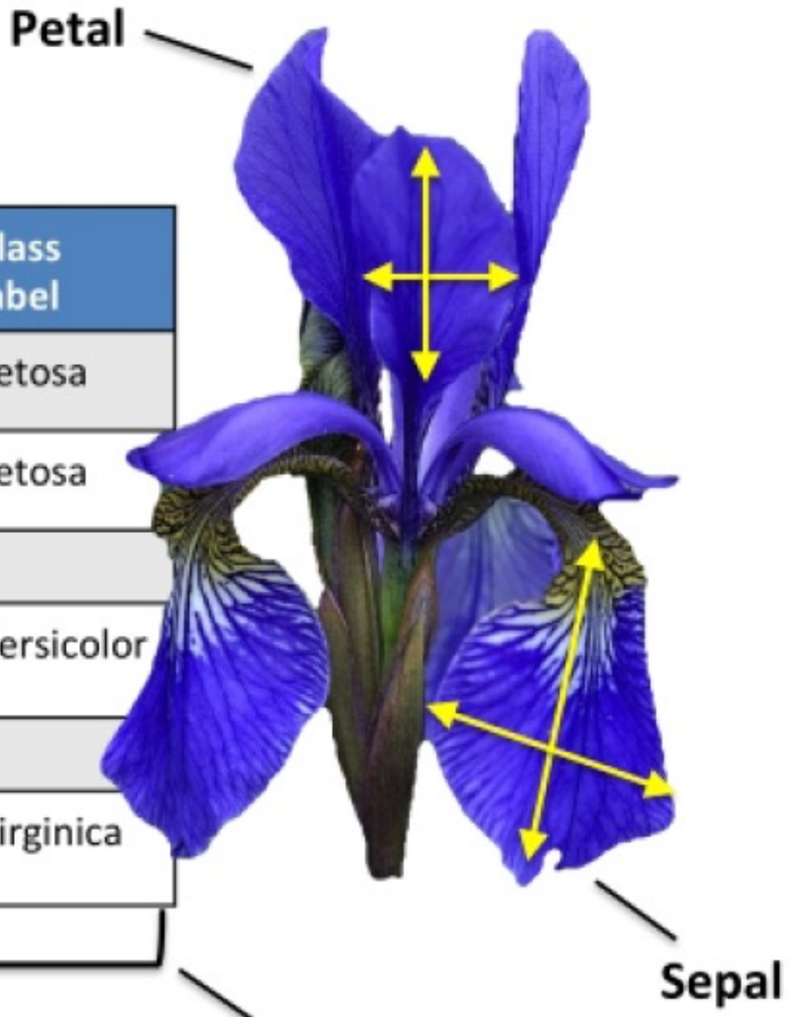
- In python type the following to import the dataset and save it in an object:

  from sklearn.datasets import load_iris #load the dataset
  iris_dataset=load_iris() #this is an object similar to a dictionary

- There are 150 samples in the dataset, collected by biologist Ronald Fisher

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica



Features
(attributes, measurements, dimensions)

Class labels
(targets)

Raschka, Python Machine Learning

- iris_dataset has a dictionary like type (called Bunch)

- Find the keys of the iris_datasets using **iris_dataset.keys():** ['target_names', 'data', 'target', 'DESCR', 'feature_names’])

  - DESCR: contains description of the dataset
  - target_names: array of strings containing the species
  - **target**: 0, 1, 2 corresponding to species (Setosa, Versicolor, Virginica)
  - **data**: contains the measurement of data (features/predictors)

- Understand the dataset by reading DESCR

Step 2: Split the Data to Training and Test Set

- Split the data to training (75%) and test (25%)

- Scikit-learn has a function that **shuffles** and **splits** the data

- In python:

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test =
train_test_split(iris_dataset['data'],iris_dataset['target'],random_state=100)
```

  - random_state is a seed to the random generator to ensure that same sequence is output every run

- Find the shape of text and train samples:
  - 112 samples (75% of data points) and 38 samples

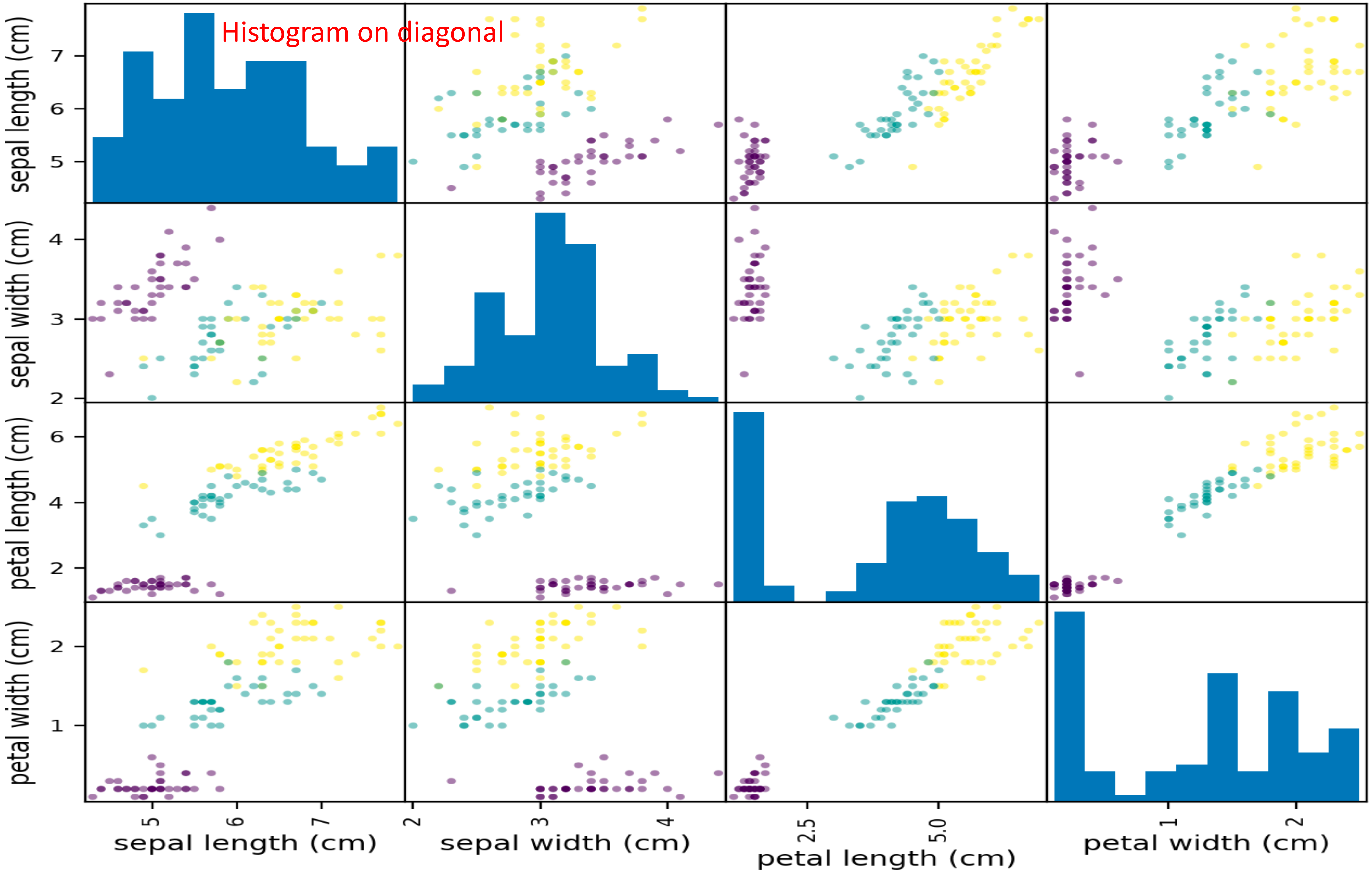
Inspect the data

- Inspect the data: anything missing? Sufficient samples from the classes?

- Use **scatter_plot** which is a function supported by **pandas** whose input has to be a **DataFrame**

- Python code:

```
import pandas as pd
iris_dataFrame=pd.DataFrame(X_train, columns=iris_dataset.feature_names)
#create a scatter_matrix for the dataframe
sm=pd.plotting.scatter_matrix(iris_dataFrame,c=Y_train, figsize=(15,15))
```



Different colors = different classes (species) Data can be distinguished through the features!

- You can draw pairplots with Seaborn as well

```
import seaborn as sns
sns.pairplot(iris_dataFrame)
```

Step 3: Define your Model
K-Nearest Neighbor from Scikit-Learn Functions

- In scikit-learn: KNeighborClassifier class in neighbors module
  from sklearn.neighbors import KNeighborsClassifier
- Then, **initiate class into an object**
  knn=**KNeighborsClassifier**(n_neighbors=k)

Step 4: Fit (Train) your Model

- **Fit the model using the training data:** by calling the fit object
  knn.fit(X_train, Y_train)

Step 5: Performance Evaluation

- Accuracy: the fraction of flowers for which the correct species **was predicted correctly**

  Accuracy=**knn.score**(X_test,Y_test)

- Alternatively, you can make predictions and then accuracy using the test set as follows:
  Y_predict=knn.predict(X_test)
  np.mean(Y_predict==Y_test)

Make predictions for a new sample

- **Make predictions:** predict label of new data
  - Define a new sample with: sepal length=5cm, sepal width=2.9cm , petal length: 1cm, petal width= 0.2cm.
    - X_new=np.array([5,2.9,1,0.2])

  - Call predict method:
    prediction=**knn.predict**(X_new.reshape(1,-1))

Feature Scaling Could be Essential

- Assume two features: one in the range 0 – 1, and another in the range of 100-10000.
  - The contribution to the Euclidean distance will be different!
- Thus, feature scaling is recommended in practice and would improve performance
- Done in the preprocessing step

Feature Scaling - MinMaxScaler

- MinMaxScaler: scales features to be in range 0 -1

```
from sklearn import preprocessing
scaler=preprocessing.MinMaxScaler().fit(X_train) #define scaler depending on the features in training data
X_train_transformed=scaler.transform(X_train) #apply scaling on training set
X_test_transformed=scaler.transform(X_test) #apply scaling on test set
```

Feature Scaling - StandardScaler

- StandardScaler: scales features so that they are all with zero mean and unit variance

```
from sklearn import preprocessing
scaler=preprocessing.StandardScaler().fit(X_train) #define scaler depending on the features in training data
X_train_transformed=scaler.transform(X_train) #apply scaling on training set
X_test_transformed=scaler.transform(X_test) #apply scaling on test set
```

Exercise

- Implement KNN without built in functions

- Then, you can use built in functions to assess different scenarios.

  - Classify the Iris species with KNN approach using the **first two feature only** and check the accuracy as **K changes**. Let K takes the values [1, 5, 10, 15]. No need to scale features.

  In the code, use **random_state=100** in **train_test_split**

    - Plot the accuracy and comment on your result

  - Use the Iris example, and find the accuracy of the KNN approach with K=5 when **different number of features** is used **without scaling**

    - Repeat when feature scaling with MinMaxScaler is used