**K-Nearest Neighbor**

**Load Data**

Loading the Iris dataset from scikit-learn. Here, the third column represents the petal length, and the fourth column the petal width of the flower samples. The classes are already converted to integer labels where 0=Iris-Setosa, 1=Iris-Versicolor, 2=Iris-Virginica.

import warnings

warnings.filterwarnings('ignore')

frommatplotlib.colors import ListedColormap

importmatplotlib.pyplot as plt

%matplotlib inline

fromsklearn import datasets

importnumpy as np

import pandas as pd

fromsklearn import tree

fromsklearn import metrics

iris = datasets.load\_iris()

X = iris.data

y = iris.target

print('Class labels:', np.unique(y))

**Output:**

Class labels: [0 1 2]

**Normalize data:** the unit of measurement might differ so lets normalize the data before building the model.

fromsklearn.preprocessing import StandardScaler

sc = StandardScaler()

sc.fit(X)

X = sc.transform(X)

**Split data into train and test**: Whenever we are using radom function its advised to use a seed to ensure the reproducibility of the results.

# split data into train and test

fromsklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=0)

defplot\_decision\_regions(X, y, classifier):

h = .02 # step size in the mesh

# setup marker generator and color map

markers = ('s', 'x', 'o', '^', 'v')

colors = ('red', 'blue', 'lightgreen', 'gray', 'cyan')

cmap = ListedColormap(colors[:len(np.unique(y))])

# plot the decision surface

x1\_min, x1\_max = X[:, 0].min() - 1, X[:, 0].max() + 1

x2\_min, x2\_max = X[:, 1].min() - 1, X[:, 1].max() + 1

xx1, xx2 = np.meshgrid(np.arange(x1\_min, x1\_max, h),

np.arange(x2\_min, x2\_max, h))

Z = classifier.predict(np.array([xx1.ravel(), xx2.ravel()]).T)

Z = Z.reshape(xx1.shape)

plt.contourf(xx1, xx2, Z, alpha=0.4, cmap=cmap)

plt.xlim(xx1.min(), xx1.max())

plt.ylim(xx2.min(), xx2.max())

foridx, cl in enumerate(np.unique(y)):

plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1],

alpha=0.8, c=cmap(idx),

marker=markers[idx], label=cl)

fromsklearn.neighbors import KNeighborsClassifier

clf = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski')

clf.fit(X\_train, y\_train)

# generate evaluation metrics

print("Train - Accuracy :", metrics.accuracy\_score(y\_train, clf.predict(X\_train)))

print("Train - Confusion matrix :",metrics.confusion\_matrix(y\_train, clf.predict(X\_train)))

print("Train - classification report :", metrics.classification\_report(y\_train, clf.predict(X\_train)))

print("Test - Accuracy :", metrics.accuracy\_score(y\_test, clf.predict(X\_test)))

print("Test - Confusion matrix :",metrics.confusion\_matrix(y\_test, clf.predict(X\_test)))

print("Test - classification report :", metrics.classification\_report(y\_test, clf.predict(X\_test)))

**Output:**

Train - Accuracy : 0.9714285714285714

Train - Confusion matrix : [[34 0 0]

[ 0 31 1]

[ 0 2 37]]

Train - classification report :

precision recall f1-score support

0 1.00 1.00 1.00 34

1 0.94 0.97 0.95 32

2 0.97 0.95 0.96 39

microavg 0.97 0.97 0.97 105

macroavg 0.97 0.97 0.97 105

weightedavg 0.97 0.97 0.97 105

Test - Accuracy : 0.9777777777777777

Test - Confusion matrix : [[16 0 0]

[ 0 17 1]

[ 0 0 11]]

Test - classification report :

precision recall f1-score support

0 1.00 1.00 1.00 16

1 1.00 0.94 0.97 18

2 0.92 1.00 0.96 11

microavg 0.98 0.98 0.98 45

macroavg 0.97 0.98 0.98 45

weightedavg 0.98 0.98 0.98 45

### Plot Decision Boundary

Let's consider a two class example to keep things simple.

**# Let's use sklearnmake\_classification function to create some test data.**

fromsklearn.datasets import make\_classification

X, y = make\_classification(

# the usual parameters

n\_samples=100, n\_features=2, n\_informative=2, n\_redundant=0, n\_classes=2,

# Set label 0 for 80% and 1 for rest 20% of observations

weights=[.8,.2],

)

# build a simple logistic regression model

clf = KNeighborsClassifier(n\_neighbors=5, p=2, metric='minkowski')

clf.fit(X, y)

**Output:**

KNeighborsClassifier(algorithm='auto', leaf\_size=30, metric='minkowski',

metric\_params=None, n\_jobs=None, n\_neighbors=5, p=2,

weights='uniform')

**# Plot the decision boundary**

plot\_decision\_regions(X, y, classifier=clf)

plt.xlabel('X1')

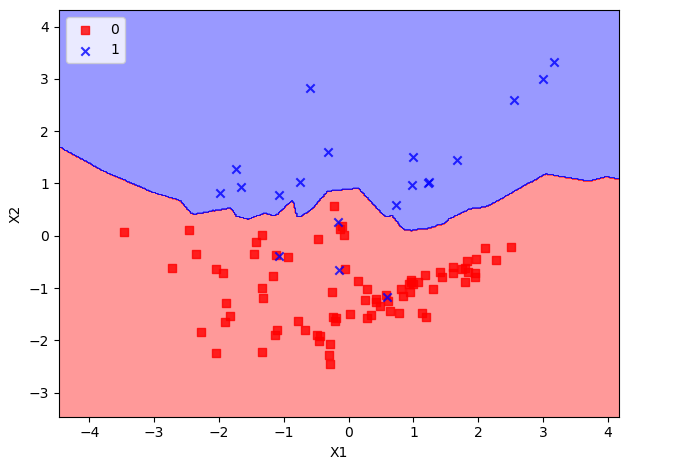
plt.ylabel('X2')

plt.legend(loc='upper left')

plt.tight\_layout()

plt.show()

**Output:**



**K-means Clustering**

Let's use unsupervised learning i.e., clustering to find out how many species exists. The goal here is to group all similar items into a cluster.

import warnings

warnings.filterwarnings('ignore')

frommatplotlib.colors import ListedColormap

importmatplotlib.pyplot as plt

%matplotlib inline

fromsklearn import datasets

importnumpy as np

import pandas as pd

fromsklearn.cluster import KMeans

fromsklearn.metrics import silhouette\_samples, silhouette\_score

fromsklearn import metrics

fromsklearn.preprocessing import StandardScaler

iris = datasets.load\_iris()

# Let's convert to dataframe

iris = pd.DataFrame(data= np.c\_[iris['data'], iris['target']],

columns= iris['feature\_names'] + ['species'])

# let's remove spaces from column name

iris.columns = iris.columns.str.replace(' ','')

iris.head()

X = iris.iloc[:,:3] # independent variables

y = iris.species # dependent variable

sc = StandardScaler()

sc.fit(X)

X = sc.transform(X)

# K Means Cluster

model = KMeans(n\_clusters=3, random\_state=11)

model.fit(X)

print(model.labels\_)

**Output:**

[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 1 1 1 0 1 1 1 1 1 1 1 1 2 2 2 0 2 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 2 0 0 0

2 2 2 2 0 0 0 0 0 0 0 2 2 0 0 0 0 2 0 0 0 0 0 0 0 0 2 0 2 2 2 2 0 2 2 2 2

2 2 0 0 2 2 2 2 0 2 0 2 0 2 2 0 2 2 2 2 2 2 2 0 2 2 2 0 2 2 2 0 2 2 2 0 2

2 0]

##**CONFUSION MATRIX Generation**

# since its unsupervised the labels have been assigned

# not in line with the actual lables so let's convert all the 1s to 0s and 0s #to 1s

# 2's look fine

iris['pred\_species'] = np.choose(model.labels\_, [1, 0, 2]).astype(np.int64)

print("Accuracy :", metrics.accuracy\_score(iris.species, iris.pred\_species))

print("Classification report :", metrics.classification\_report(iris.species, iris.pred\_species))

**Output:**

Accuracy : 0.8066666666666666

Classification report :

precision recall f1-score support

0.0 1.00 0.98 0.99 50

1.0 0.71 0.70 0.71 50

2.0 0.71 0.74 0.73 50

microavg 0.81 0.81 0.81 150

macroavg 0.81 0.81 0.81 150

weightedavg 0.81 0.81 0.81 150

**##PLOTTING:**

# Set the size of the plot

plt.figure(figsize=(10,7))

# Create a colormap for red, green and blue

cmap = ListedColormap(['r', 'g', 'b'])

# Plot Sepal

plt.subplot(2, 2, 1)

plt.scatter(iris['sepallength(cm)'], iris['sepalwidth(cm)'], c=cmap(iris.species), marker='o', s=50)

plt.xlabel('sepallength(cm)')

plt.ylabel('sepalwidth(cm)')

plt.title('Sepal (Actual)')

plt.subplot(2, 2, 2)

plt.scatter(iris['sepallength(cm)'], iris['sepalwidth(cm)'], c=cmap(iris.pred\_species), marker='o', s=50)

plt.xlabel('sepallength(cm)')

plt.ylabel('sepalwidth(cm)')

plt.title('Sepal (Predicted)')

plt.subplot(2, 2, 3)

plt.scatter(iris['petallength(cm)'], iris['petalwidth(cm)'], c=cmap(iris.species),marker='o', s=50)

plt.xlabel('petallength(cm)')

plt.ylabel('petalwidth(cm)')

plt.title('Petal (Actual)')

plt.subplot(2, 2, 4)

plt.scatter(iris['petallength(cm)'], iris['petalwidth(cm)'], c=cmap(iris.pred\_species),marker='o', s=50)

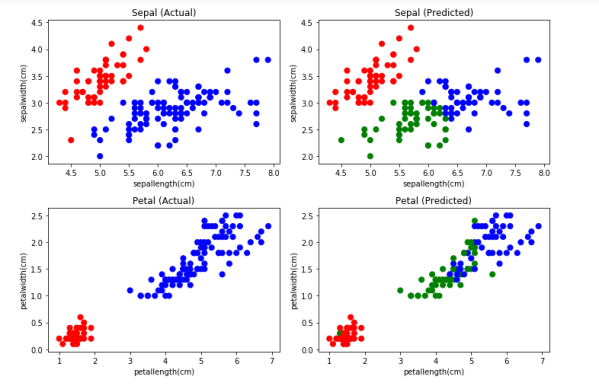
plt.xlabel('petallength(cm)')

plt.ylabel('petalwidth(cm)')

plt.title('Petal (Predicted)')

plt.tight\_layout()

**Output:**



### Finding value of K:

### fromscipy.spatial.distance import cdist, pdist

### fromsklearn.cluster import KMeans

### K = range(1,10)

### KM = [KMeans(n\_clusters=k).fit(X) for k in K]

### centroids = [k.cluster\_centers\_ for k in KM]

### D\_k = [cdist(X, cent, 'euclidean') for cent in centroids]

### cIdx = [np.argmin(D,axis=1) for D in D\_k]

### dist = [np.min(D,axis=1) for D in D\_k]

### avgWithinSS = [sum(d)/X.shape[0] for d in dist]

### # Total with-in sum of square

### wcss = [sum(d\*\*2) for d in dist]

### tss = sum(pdist(X)\*\*2)/X.shape[0]

### bss = tss-wcss

### varExplained = bss/tss\*100

### kIdx = 10-1

### ##### plot ###

### kIdx = 2

### # elbow curve

### # Set the size of the plot

### plt.figure(figsize=(10,4))

### plt.subplot(1, 2, 1)

### plt.plot(K, avgWithinSS, 'b\*-')

### plt.plot(K[kIdx], avgWithinSS[kIdx], marker='o', markersize=12,

### markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')

### plt.grid(True)

### plt.xlabel('Number of clusters')

### plt.ylabel('Average within-cluster sum of squares')

### plt.title('Elbow for KMeans clustering')

### plt.subplot(1, 2, 2)

### plt.plot(K, varExplained, 'b\*-')

### plt.plot(K[kIdx], varExplained[kIdx], marker='o', markersize=12,

### markeredgewidth=2, markeredgecolor='r', markerfacecolor='None')

### plt.grid(True)

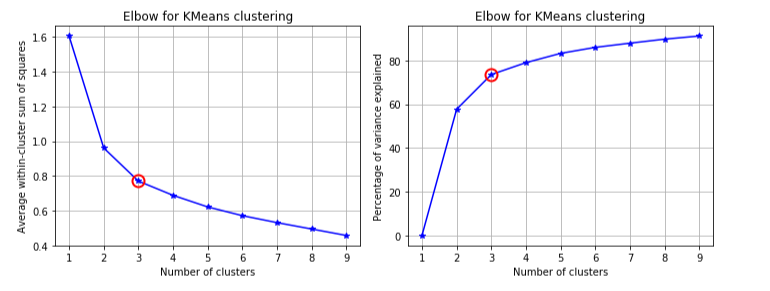
### plt.xlabel('Number of clusters')

### plt.ylabel('Percentage of variance explained')

### plt.title('Elbow for KMeans clustering')

### plt.tight\_layout()

**Output:**



**Average silhouette method**

fromsklearn.metrics import silhouette\_score

frommatplotlib import cm

score = []

forn\_clusters in range(2,10):

kmeans = KMeans(n\_clusters=n\_clusters)

kmeans.fit(X)

labels = kmeans.labels\_

centroids = kmeans.cluster\_centers\_

score.append(silhouette\_score(X, labels, metric='euclidean'))

# Set the size of the plot

plt.figure(figsize=(10,4))

plt.subplot(1, 2, 1)

plt.plot(score)

plt.grid(True)

plt.ylabel("Silouette Score")

plt.xlabel("k")

plt.title("Silouette for K-means")

# Initialize the clusterer with n\_clusters value and a random generator

model = KMeans(n\_clusters=3, init='k-means++', n\_init=10, random\_state=0)

model.fit\_predict(X)

cluster\_labels = np.unique(model.labels\_)

n\_clusters = cluster\_labels.shape[0]

# Compute the silhouette scores for each sample

silhouette\_vals = silhouette\_samples(X, model.labels\_)

plt.subplot(1, 2, 2)

# Get spectral values for colormap.

cmap = cm.get\_cmap("Spectral")

y\_lower, y\_upper = 0,0

yticks = []

for i, c in enumerate(cluster\_labels):

c\_silhouette\_vals = silhouette\_vals[cluster\_labels]

c\_silhouette\_vals.sort()

y\_upper += len(c\_silhouette\_vals)

color = cmap(float(i) / n\_clusters)

plt.barh(range(y\_lower, y\_upper), c\_silhouette\_vals, facecolor=color, edgecolor=color, alpha=0.7)

yticks.append((y\_lower + y\_upper) / 2)

y\_lower += len(c\_silhouette\_vals)

silhouette\_avg = np.mean(silhouette\_vals)

plt.yticks(yticks, cluster\_labels+1)

# The vertical line for average silhouette score of all the values

plt.axvline(x=silhouette\_avg, color="red", linestyle="--")

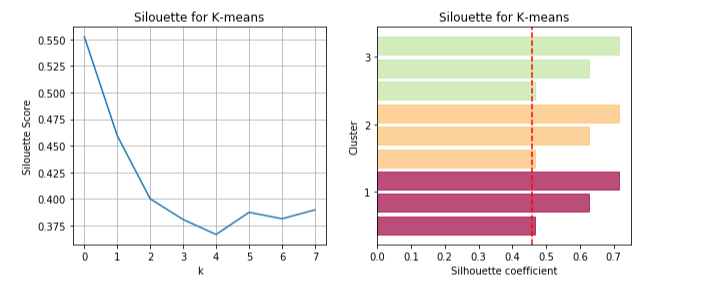
plt.ylabel('Cluster')

plt.xlabel('Silhouette coefficient')

plt.title("Silouette for K-means")

plt.show()

**Output:**



**Hierarchical Clustering**

fromsklearn.cluster import AgglomerativeClustering

# Agglomerative Cluster

model = AgglomerativeClustering(n\_clusters=3)

model.fit(X)

print(model.labels\_)

iris['pred\_species'] = model.labels\_

print("Accuracy :", metrics.accuracy\_score(iris.species, iris.pred\_species))

print("Classification report :", metrics.classification\_report(iris.species, iris.pred\_species))

**Output:**

[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 1 0 0 0 0 0 0 0 0 2 2 2 1 2 1 2 1 2 1 1 1 1 1 1 2 1 1 1 1 2 1 1 1

1 2 2 2 1 1 1 1 1 1 1 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 2 1 2 1 2 2 1 2 1 2 2

1 2 1 1 2 2 2 2 1 2 1 2 1 2 2 1 1 1 2 2 2 1 1 1 2 2 2 1 2 2 2 1 2 2 2 1 2

2 1]

Accuracy : 0.7733333333333333

Classification report :

precision recall f1-score support

0.0 1.00 0.98 0.99 50

1.0 0.64 0.74 0.69 50

2.0 0.70 0.60 0.65 50

microavg 0.77 0.77 0.77 150

macroavg 0.78 0.77 0.77 150

weightedavg 0.78 0.77 0.77 150

**PLOTTING ACTUAL Vs. PREDICTED**

# Set the size of the plot

plt.figure(figsize=(10,7))

# Create a colormap for red, green and blue

cmap = ListedColormap(['r', 'g', 'b'])

# Plot Sepal

plt.subplot(2, 2, 1)

plt.scatter(iris['sepallength(cm)'], iris['sepalwidth(cm)'], c=cmap(iris.species), marker='o', s=50)

plt.xlabel('sepallength(cm)')

plt.ylabel('sepalwidth(cm)')

plt.title('Sepal (Actual)')

plt.subplot(2, 2, 2)

plt.scatter(iris['sepallength(cm)'], iris['sepalwidth(cm)'], c=cmap(iris.pred\_species), marker='o', s=50)

plt.xlabel('sepallength(cm)')

plt.ylabel('sepalwidth(cm)')

plt.title('Sepal (Predicted)')

plt.subplot(2, 2, 3)

plt.scatter(iris['petallength(cm)'], iris['petalwidth(cm)'], c=cmap(iris.species),marker='o', s=50)

plt.xlabel('petallength(cm)')

plt.ylabel('petalwidth(cm)')

plt.title('Petal (Actual)')

plt.subplot(2, 2, 4)

plt.scatter(iris['petallength(cm)'], iris['petalwidth(cm)'], c=cmap(iris.pred\_species),marker='o', s=50)

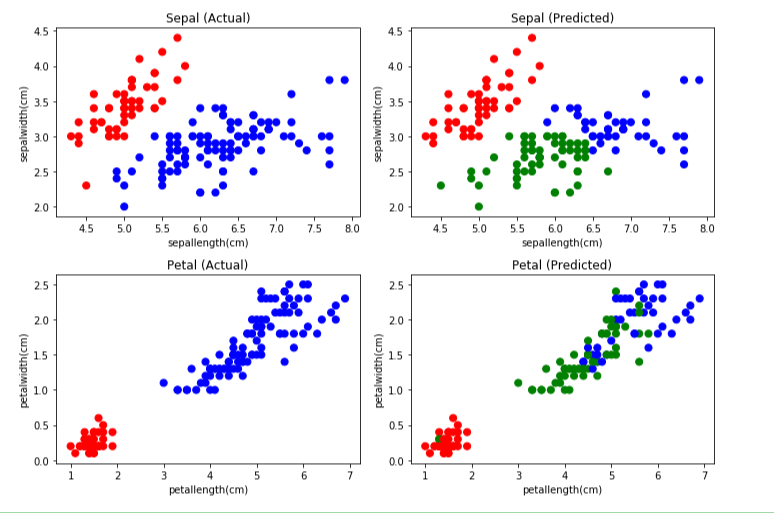
plt.xlabel('petallength(cm)')

plt.ylabel('petalwidth(cm)')

plt.title('Petal (Predicted)')

plt.tight\_layout()

**Output:**



**AGGLOMERATIVE HIERARCHICAL CLUSTERING**

fromscipy.cluster.hierarchy import cophenet, dendrogram, linkage

fromscipy.spatial.distance import pdist

# generate the linkage matrix

Z = linkage(X, 'ward')

c, coph\_dists = cophenet(Z, pdist(X))

# calculate full dendrogram

plt.figure(figsize=(25, 10))

plt.title('Agglomerative Hierarchical Clustering Dendrogram')

plt.xlabel('sample index')

plt.ylabel('distance')

dendrogram(

Z,

leaf\_rotation=90., # rotates the x axis labels

leaf\_font\_size=8., # font size for the x axis labels

)

plt.tight\_layout()

**Output:**

