**Basic Numpy Functions**

import numpy as np

np.zeros(6)

np.ones(6)

data = np.array([5,5,5,5,5,5])

data

np.arange(1,100)

np.arange(1,100,2)

np.arange(1,5).reshape(2,2)

np.arange(9,18).reshape(3,3)

np.identity(4)

np.random.rand()

np.linspace(1,10,10)

a1 =np.array([[[0,1,2]],[[3,4,5]],[[6,7,8],[2,3,4]]])

a2 =np.array([[[10,11,12]],[[13,14,15]],[[16,17,18],[12,13,14]]])

np.equal(a1,a2)

x = np.zeros(20)

x[6] = 5

x

np.flip(np.arange(0,100))

x = np.random.randint(0,800,size=(20,20))

max = np.max(x)

min = np.min(x)

print("Maximum is ", max)

print("Minimum is ", min)

np.mean(np.random.randint(0,2000,size=(50)))

x = np.ones((20,20))

x[1:-1,1:-1] = 0

x

x = np.empty((10,10))

x[-1:] = np.nan

x

x = np.random.randint(20,200,size=(4,4))

x[[1,2,3],[0,1,2]] = [9,8,7]

x

x = np.zeros((10, 10), dtype = int)

x[1::2, ::2] = 1

x[::2, 1::2] = 1

x

print(np.dtype(np.int16))

print(np.dtype(np.float64))

**Basic Pandas Functions**

import pandas as pd

ser = pd.Series([1.5,2.5,3.5,4.5,5.5,6.5])

ser

ser1 = pd.Series({"India": "New Delhi", "Japan": "Tokyo", "UK": "London"})

ser1

ser2 = pd.Series(np.linspace(1,5,5))

ser2

print(ser1.index)

print(ser1.values)

Countries = ['Australia', 'Iceland', 'India', 'United Kingdom', 'United States']

Capitals = ['Canberra', 'Reykjavik', 'New Delhi', 'London', 'Washington D.C']

s = pd.Series(Capitals, index=Countries)

s

print(len(ser1))

print(s.size)

print(s.shape)

print(s.head(2))

print(s.tail(2))

data\_df = {'Name': ['Arpit', 'Riya', 'Priyanka', 'Aman', 'Arpit', 'Rohan', 'Riya', 'Sakshi'], 'Employment Type': ['Full-time Employee', 'Part-time Employee', 'Intern', 'Intern', 'Full-time Employee', 'Part-time Employee', 'Part-time Employee', 'Full-time Employee'], 'Department': ['Administration', 'Marketing', 'Technical', 'Marketing', 'Administration', 'Technical', 'Marketing', 'Administration']}

df = pd.DataFrame(data\_df)

df.duplicated()

**Normalizing Values 2 types**

from sklearn.preprocessing import MinMaxScaler

from sklearn.preprocessing import StandardScaler

dataset = pd.read\_csv("./data.csv")

dataset = dataset.drop(['Gender', 'Index'], axis = 1)

dataset['Height'] = dataset['Height'].multiply(0.0328084)

dataset['Weight'] = dataset['Weight'].multiply(1000)

dataset.head(10)

scaler = MinMaxScaler(feature\_range=(-10,10))

rescaled = scaler.fit\_transform(dataset)

np.set\_printoptions(precision=3)

print(rescaled)

stdscaler = StandardScaler().fit(dataset)

rescaled = stdscaler.transform(dataset)

print(rescaled)

**DA2**

***FIND-S***

import csv

num\_attributes = 5

a = []

print("\n The Given Training Data Set \n")

with open('dataset.csv', 'r') as csvfile:

reader = csv.reader(csvfile)

for row in reader:

a.append (row)

print(row)

print("\n The initial value of hypothesis: ")

hypothesis = ['0'] \* num\_attributes

print(hypothesis)

for j in range(0,num\_attributes):

hypothesis[j] = a[1][j]

print("\n The a[1] value of hypothesis: ")

print(hypothesis)

print("\n Find S: Finding a Maximally Specific Hypothesis\n")

for i in range(0,len(a)):

if a[i][num\_attributes]=='Yes':

for j in range(0,num\_attributes):

if a[i][j]!=hypothesis[j]:

hypothesis[j]='?'

else :

hypothesis[j]= a[i][j]

print(" For Training instance No:{} the hypothesis is ".format(i), hypothesis)

print("\n The Maximally Specific Hypothesis for a given Training Examples :\n")

print(hypothesis)

**Candidate Elimination**

import numpy as np

import pandas as pd

import random

import csv

def g\_0(n):

return ("?",)\*n

def s\_0(n):

return ('0',)\*n

with open('dataset.csv') as csvFile:

examples = [tuple(line) for line in csv.reader(csvFile)]

def get\_domains(examples):

d = [set() for i in examples[0]]

for x in examples:

for i, xi in enumerate(x):

d[i].add(xi)

return [list(sorted(x)) for x in d]

def more\_general(h1, h2):

more\_general\_parts = []

for x, y in zip(h1, h2):

mg = x == "?" or (x != "0" and (x == y or y == "0"))

more\_general\_parts.append(mg)

return all(more\_general\_parts)

def fulfills(example, hypothesis):

return more\_general(hypothesis, example)

def min\_generalizations(h, x):

h\_new = list(h)

for i in range(len(h)):

if not fulfills(x[i:i+1], h[i:i+1]):

h\_new[i] = '?' if h[i] != '0' else x[i]

return [tuple(h\_new)]

def min\_specializations(h, domains, x):

results = []

for i in range(len(h)):

if h[i] == "?":

for val in domains[i]:

if x[i] != val:

h\_new = h[:i] + (val,) + h[i+1:]

results.append(h\_new)

elif h[i] != "0":

h\_new = h[:i] + ('0',) + h[i+1:]

results.append(h\_new)

return results

def candidate\_elimination(examples):

domains = get\_domains(examples)[:-1]

G = set([g\_0(len(domains))])

S = set([s\_0(len(domains))])

i=0

print("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

for xcx in examples:

i=i+1

x, cx = xcx[:-1], xcx[-1]

if cx=='Y':

G = {g for g in G if fulfills(x, g)}

S = generalize\_S(x, G, S)

else:

S = {s for s in S if not fulfills(x, s)}

G = specialize\_G(x, domains, G, S)

print("\n G[{0}]:".format(i),G)

print("\n S[{0}]:".format(i),S)

return

def generalize\_S(x, G, S):

S\_prev = list(S)

for s in S\_prev:

if s not in S:

continue

if not fulfills(x, s):

S.remove(s)

Splus = min\_generalizations(s, x)

S.update([h for h in Splus if any([more\_general(g,h) for g in G])])

S.difference\_update([h for h in S if any([more\_general(h, h1) for h1 in S if h != h1])])

return S

def specialize\_G(x, domains, G, S):

G\_prev = list(G)

for g in G\_prev:

if g not in G:

continue

if fulfills(x, g):

G.remove(g)

Gminus = min\_specializations(g, domains, x)

G.update([h for h in Gminus if any([more\_general(h, s)for s in S])])

G.difference\_update([h for h in G if any([more\_general(g1, h) for g1 in G if h != g1])])

return G

candidate\_elimination(examples)

#2 https://www.vtupulse.com/machine-learning/candidate-elimination-algorithm-in-python/

#3 https://github.com/profthyagu/Python-ANN-Backpropagation

#4 https://www.chegg.com/homework-help/questions-and-answers/question-1-regression-analysis-8-marks-market-research-real-estate-investments-carried-las-q28781725

**#ANN**

import random

from math import exp

from random import seed

# Initialize a network

def initialize\_network(n\_inputs, n\_hidden, n\_outputs):

network = list()

hidden\_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n\_inputs + 1)]} for i in range(n\_hidden)]

network.append(hidden\_layer)

output\_layer = [{'weights':[random.uniform(-0.5,0.5) for i in range(n\_hidden + 1)]} for i in range(n\_outputs)]

network.append(output\_layer)

i= 1

print("\n The initialised Neural Network:\n")

for layer in network:

j=1

for sub in layer:

print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)

j=j+1

i=i+1

return network

def activate(weights, inputs):

activation = weights[-1]

for i in range(len(weights)-1):

activation += weights[i] \* inputs[i]

return activation

def transfer(activation):

return 1.0 / (1.0 + exp(-activation))

def forward\_propagate(network, row):

inputs = row

for layer in network:

new\_inputs = []

for neuron in layer:

activation = activate(neuron['weights'], inputs)

neuron['output'] = transfer(activation)

new\_inputs.append(neuron['output'])

inputs = new\_inputs

return inputs

def transfer\_derivative(output):

return output \* (1.0 - output)

def backward\_propagate\_error(network, expected):

for i in reversed(range(len(network))):

layer = network[i]

errors = list()

if i != len(network)-1:

for j in range(len(layer)):

error = 0.0

for neuron in network[i + 1]:

error += (neuron['weights'][j] \* neuron['delta'])

errors.append(error)

else:

for j in range(len(layer)):

neuron = layer[j]

errors.append(expected[j] - neuron['output'])

for j in range(len(layer)):

neuron = layer[j]

neuron['delta'] = errors[j] \* transfer\_derivative(neuron['output'])

def update\_weights(network, row, l\_rate):

for i in range(len(network)):

inputs = row[:-1]

if i != 0:

inputs = [neuron['output'] for neuron in network[i - 1]]

for neuron in network[i]:

for j in range(len(inputs)):

neuron['weights'][j] += l\_rate \* neuron['delta'] \* inputs[j]

neuron['weights'][-1] += l\_rate \* neuron['delta']

def train\_network(network, train, l\_rate, n\_epoch, n\_outputs):

print("\n Network Training Begins:\n")

for epoch in range(n\_epoch):

sum\_error = 0

for row in train:

outputs = forward\_propagate(network, row)

expected = [0 for i in range(n\_outputs)]

expected[row[-1]] = 1

sum\_error += sum([(expected[i]-outputs[i])\*\*2 for i in range(len(expected))])

backward\_propagate\_error(network, expected)

update\_weights(network, row, l\_rate)

print('>epoch=%d, lrate=%.3f, error=%.3f' % (epoch, l\_rate, sum\_error))

print("\n Network Training Ends:\n")

seed(2)

dataset = [[2.7810836,2.550537003,0],

[1.465489372,2.362125076,0],

[3.396561688,4.400293529,0],

[1.38807019,1.850220317,0],

[3.06407232,3.005305973,0],

[7.627531214,2.759262235,1],

[5.332441248,2.088626775,1],

[6.922596716,1.77106367,1],

[8.675418651,-0.242068655,1],

[7.673756466,3.508563011,1]]

print("\n The input Data Set :\n",dataset)

n\_inputs = len(dataset[0]) - 1

print("\n Number of Inputs :\n",n\_inputs)

n\_outputs = len(set([row[-1] for row in dataset]))

print("\n Number of Outputs :\n",n\_outputs)

#Network Initialization

network = initialize\_network(n\_inputs, 2, n\_outputs)

# Training the Network

train\_network(network, dataset, 0.5, 20, n\_outputs)

print("\n Final Neural Network :")

i= 1

for layer in network:

j=1

for sub in layer:

print("\n Layer[%d] Node[%d]:\n" %(i,j),sub)

j=j+1

i=i+1

**Regression**

To plot the data and check for a linear relationship between the price and sales quantity attributes, we can use the scatter plot. In python, we can use the matplotlib library to create a scatter plot.

import matplotlib.pyplot as plt

# data

price = [160, 280, 180, 200, 260, 240, 220, 170]

sales\_quantity = [125, 120, 104, 85, 40, 80, 75, 79]

# create scatter plot

plt.scatter(price, sales\_quantity)

plt.xlabel('Price')

plt.ylabel('Sales Quantity')

plt.show()

From the scatter plot, we can see that there is a negative linear relationship between the price and sales quantity attributes, as the data points form a line with a negative slope.

To find the least square regression line, we can use the linregress() function from the scipy library.

from scipy.stats import linregress

# calculate linear regression

slope, intercept, r\_value, p\_value, std\_err = linregress(price, sales\_quantity)

# print results

print('slope:', slope)

print('intercept:', intercept)

print('r\_value:', r\_value)

print('p\_value:', p\_value)

print('std\_err:', std\_err)

To plot the regression line on the scatter plot, we can use the plot() function from matplotlib, and pass in the calculated slope and y-intercept values.

# create scatter plot

plt.scatter(price, sales\_quantity)

plt.xlabel('Price')

plt.ylabel('Sales Quantity')

# plot regression line

temp = []

for i in price:

temp.append(slope\*i + intercept)

plt.plot(price, temp, 'r')

plt.show()

**ID3 Decision Tree**

#https://www.kaggle.com/code/ankitmalik/decision-trees-from-scratch-id3

import math

import pandas as pd

from operator import itemgetter

class DecisionTree:

def \_\_init\_\_(self, df, target, positive, parent\_val, parent):

self.data = df

self.target = target

self.positive = positive

self.parent\_val = parent\_val

self.parent = parent

self.childs = []

self.decision = ''

def \_get\_entropy(self, data):

p = sum(data[self.target]==self.positive)

n = data.shape[0] - p

p\_ratio = p/(p+n)

n\_ratio = 1 - p\_ratio

entropy\_p = -p\_ratio\*math.log2(p\_ratio) if p\_ratio != 0 else 0

entropy\_n = - n\_ratio\*math.log2(n\_ratio) if n\_ratio !=0 else 0

return entropy\_p + entropy\_n

def \_get\_gain(self, feat):

avg\_info=0

for val in self.data[feat].unique():

avg\_info+=self.\_get\_entropy(self.data[self.data[feat] == val])\*sum(self.data[feat]==val)/self.data.shape[0]

return self.\_get\_entropy(df) - avg\_info

def \_get\_splitter(self):

self.splitter = max(self.gains, key = itemgetter(1))[0]

def update\_nodes(self):

self.features = [col for col in self.data.columns if col != self.target]

self.entropy = self.\_get\_entropy(self.data)

if self.entropy != 0:

self.gains = [(feat, self.\_get\_gain(feat)) for feat in self.features]

self.\_get\_splitter()

residual\_columns = [k for k in self.data.columns if k != self.splitter]

for val in self.data[self.splitter].unique():

df\_tmp = self.data[self.data[self.splitter]==val][residual\_columns]

tmp\_node = DecisionTree(df\_tmp, self.target, self.positive, val, self.splitter)

tmp\_node.update\_nodes()

self.childs.append(tmp\_node)

def print\_tree(n):

for child in n.childs:

if child:

print(child.\_\_dict\_\_.get('parent', ''))

print(child.\_\_dict\_\_.get('parent\_val', ''), '\n')

print\_tree(child)

df = pd.read\_csv('decisiontree.csv')

df

dt = DecisionTree(df, 'Play', 'Yes', '', '')

dt.update\_nodes()

print\_tree(dt)

#6 https://www.vtupulse.com/machine-learning/k-nearest-neighbour-algorithm-in-python/

#7 https://github.com/vlevko/ML\_codebasics

**#KNN**

import numpy as np

import pandas as pd

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split

from sklearn import metrics

names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'Class']

# Read dataset to pandas dataframe

dataset = pd.read\_csv("./irisdataset.csv", names=names)

X = dataset.iloc[:, :-1]

y = dataset.iloc[:, -1]

print(X.head())

Xtrain, Xtest, ytrain, ytest = train\_test\_split(X, y, test\_size=0.10)

classifier = KNeighborsClassifier(n\_neighbors=5).fit(Xtrain, ytrain)

ypred = classifier.predict(Xtest)

i = 0

print ("\n-------------------------------------------------------------------------")

print ('%-25s %-25s %-25s' % ('Original Label', 'Predicted Label', 'Correct/Wrong'))

print ("-------------------------------------------------------------------------")

for label in ytest:

print ('%-25s %-25s' % (label, ypred[i]), end="")

if (label == ypred[i]):

print (' %-25s' % ('Correct'))

else:

print (' %-25s' % ('Wrong'))

i = i + 1

print ("-------------------------------------------------------------------------")

print("\nConfusion Matrix:\n",metrics.confusion\_matrix(ytest, ypred))

print ("-------------------------------------------------------------------------")

print("\nClassification Report:\n",metrics.classification\_report(ytest, ypred))

print ("-------------------------------------------------------------------------")

print('Accuracy of the classifer is %0.2f' % metrics.accuracy\_score(ytest,ypred))

print ("-------------------------------------------------------------------------")

**#SVM**

import pandas as pd

from sklearn.svm import SVC

from sklearn.model\_selection import train\_test\_split

from sklearn.datasets import load\_digits

digits = load\_digits()

dir(digits)

digits.target\_names

digits.target

digits.data[0]

df = pd.DataFrame(digits.data,digits.target)

df.head()

df['target'] = digits.target

df.head(15)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(df.drop('target',axis='columns'),df.target,test\_size=0.2)

rbf\_model = SVC(kernel='rbf',gamma=0.002)

rbf\_model.fit(X\_train,y\_train)

rbf\_model.score(X\_test,y\_test)

linear\_model = SVC(kernel='linear',C=0.001)

linear\_model.fit(X\_train,y\_train)

linear\_model.score(X\_test,y\_test)

**Write a program to demonstrate the working of the decision tree based CART algorithm**

**using GINI index calculation. Use an appropriate data set for building the decision tree and**

**apply this knowledge to classify a new sample.**

# Import necessary libraries

from sklearn import datasets

from sklearn import tree

from sklearn.model\_selection import train\_test\_split

# Load the iris dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Split the data into training and test sets

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=0)

# Create a Decision Tree classifier object

clf = tree.DecisionTreeClassifier(criterion='gini')

# Train the model using the training sets

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

# Predict the response for test dataset

y\_pred = clf.predict(X\_test)

# Print accuracy of the model

print("Accuracy:", clf.score(X\_test, y\_test))

# Classify a new sample

sample = [[3,5,4,2]]

prediction = clf.predict(sample)

print("Predicted class for new sample:", prediction)

In this example, we first load the iris dataset from the sklearn library and split it into training and test sets. Then we create a Decision Tree classifier object and set the criterion as 'gini' to use GINI index calculation. We fit the model using the training sets, predict the response for the test dataset, and print the accuracy of the model. Finally, we classify a new sample using the trained model and print the predicted class for the sample.

Note : this example is just a demonstration of how CART algorithm can be implemented using GINI index calculation, It is not recommended for the real-world problem, as it's not tuned for the best performance and it's not showing how to handle overfitting or underfitting.

#https://www.kaggle.com/code/faressayah/ensemble-ml-algorithms-bagging-boosting-voting

**Ensemble Learning – Bagging Boosting**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

sns.set\_style("whitegrid")

plt.style.use("fivethirtyeight")

df = pd.read\_csv("./diabetes.csv")

df.head()

df.info()

df.isnull().sum()

pd.set\_option('display.float\_format', '{:.2f}'.format)

df.describe()

categorical\_val = []

continous\_val = []

for column in df.columns:

if len(df[column].unique()) <= 10:

categorical\_val.append(column)

else:

continous\_val.append(column)

df.columns

feature\_columns = [

'Pregnancies', 'Glucose', 'BloodPressure', 'SkinThickness',

'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'

]

for column in feature\_columns:

print("============================================")

print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")

from sklearn.impute import SimpleImputer

fill\_values = SimpleImputer(missing\_values=0, strategy="mean", copy=False)

df[feature\_columns] = fill\_values.fit\_transform(df[feature\_columns])

for column in feature\_columns:

print("============================================")

print(f"{column} ==> Missing zeros : {len(df.loc[df[column] == 0])}")

from sklearn.model\_selection import train\_test\_split

X = df[feature\_columns]

y = df.Outcome

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=42)

from sklearn.metrics import confusion\_matrix, accuracy\_score, classification\_report

def evaluate(model, X\_train, X\_test, y\_train, y\_test):

y\_test\_pred = model.predict(X\_test)

y\_train\_pred = model.predict(X\_train)

print("TRAINIG RESULTS: \n===============================")

clf\_report = pd.DataFrame(classification\_report(y\_train, y\_train\_pred, output\_dict=True))

print(f"CONFUSION MATRIX:\n{confusion\_matrix(y\_train, y\_train\_pred)}")

print(f"ACCURACY SCORE:\n{accuracy\_score(y\_train, y\_train\_pred):.4f}")

print(f"CLASSIFICATION REPORT:\n{clf\_report}")

print("TESTING RESULTS: \n===============================")

clf\_report = pd.DataFrame(classification\_report(y\_test, y\_test\_pred, output\_dict=True))

print(f"CONFUSION MATRIX:\n{confusion\_matrix(y\_test, y\_test\_pred)}")

print(f"ACCURACY SCORE:\n{accuracy\_score(y\_test, y\_test\_pred):.4f}")

print(f"CLASSIFICATION REPORT:\n{clf\_report}")

#Bagged Decision Tree - Bagging Classifier

from sklearn.ensemble import BaggingClassifier

from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier()

bagging\_clf = BaggingClassifier(base\_estimator=tree, n\_estimators=1500, random\_state=42)

bagging\_clf.fit(X\_train, y\_train)

evaluate(bagging\_clf, X\_train, X\_test, y\_train, y\_test)

scores = {

'Bagging Classifier': {

'Train': accuracy\_score(y\_train, bagging\_clf.predict(X\_train)),

'Test': accuracy\_score(y\_test, bagging\_clf.predict(X\_test)),

},

}

#Random Forest

from sklearn.ensemble import RandomForestClassifier

rf\_clf = RandomForestClassifier(random\_state=42, n\_estimators=1000)

rf\_clf.fit(X\_train, y\_train)

evaluate(rf\_clf, X\_train, X\_test, y\_train, y\_test)

scores['Random Forest'] = {

'Train': accuracy\_score(y\_train, rf\_clf.predict(X\_train)),

'Test': accuracy\_score(y\_test, rf\_clf.predict(X\_test)),

}

#Extra Trees

from sklearn.ensemble import ExtraTreesClassifier

ex\_tree\_clf = ExtraTreesClassifier(n\_estimators=1000, max\_features=7, random\_state=42)

ex\_tree\_clf.fit(X\_train, y\_train)

evaluate(ex\_tree\_clf, X\_train, X\_test, y\_train, y\_test)

scores['Extra Tree'] = {

'Train': accuracy\_score(y\_train, ex\_tree\_clf.predict(X\_train)),

'Test': accuracy\_score(y\_test, ex\_tree\_clf.predict(X\_test)),

}

#Boosting - AdaBoost

from sklearn.ensemble import AdaBoostClassifier

ada\_boost\_clf = AdaBoostClassifier(n\_estimators=30)

ada\_boost\_clf.fit(X\_train, y\_train)

evaluate(ada\_boost\_clf, X\_train, X\_test, y\_train, y\_test)

scores['AdaBoost'] = {

'Train': accuracy\_score(y\_train, ada\_boost\_clf.predict(X\_train)),

'Test': accuracy\_score(y\_test, ada\_boost\_clf.predict(X\_test)),

}

#Stochastic Gradient Boosting

from sklearn.ensemble import GradientBoostingClassifier

grad\_boost\_clf = GradientBoostingClassifier(n\_estimators=100, random\_state=42)

grad\_boost\_clf.fit(X\_train, y\_train)

evaluate(grad\_boost\_clf, X\_train, X\_test, y\_train, y\_test)

scores['Gradient Boosting'] = {

'Train': accuracy\_score(y\_train, grad\_boost\_clf.predict(X\_train)),

'Test': accuracy\_score(y\_test, grad\_boost\_clf.predict(X\_test)),

}

#Voting Ensemble

from sklearn.ensemble import VotingClassifier

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

estimators = []

log\_reg = LogisticRegression(solver='liblinear')

estimators.append(('Logistic', log\_reg))

tree = DecisionTreeClassifier()

estimators.append(('Tree', tree))

svm\_clf = SVC(gamma='scale')

estimators.append(('SVM', svm\_clf))

voting = VotingClassifier(estimators=estimators)

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

evaluate(voting, X\_train, X\_test, y\_train, y\_test)

scores['Voting'] = {

'Train': accuracy\_score(y\_train, voting.predict(X\_train)),

'Test': accuracy\_score(y\_test, voting.predict(X\_test)),

}

scores\_df = pd.DataFrame(scores)

scores\_df.plot(kind='barh', figsize=(15, 8))

https://www.kaggle.com/code/abhijitbiswas040/kmeans-hierarchial-clustering/

**Clustering**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

dataset = pd.read\_csv("./dataset1.csv")

dataset.isnull().sum()

dataset['CREDIT\_LIMIT'].fillna(3000, inplace = True)

dataset.isnull().sum()

dataset.drop(['CUST\_ID'], axis= 1, inplace = True)

dataset = dataset.dropna()

X = dataset.iloc[:,:].values

X

# K - Means Clustering

from sklearn.preprocessing import StandardScaler

standardscaler= StandardScaler()

X = standardscaler.fit\_transform(X)

from sklearn.cluster import KMeans

wss= []

for i in range(1, 11):

kmeans= KMeans(n\_clusters = i, init = 'k-means++', random\_state = 0)

kmeans.fit(X)

wss.append(kmeans.inertia\_)

plt.plot(range(1,11), wss)

kmeans = KMeans(n\_clusters = 4, init= 'k-means++', random\_state = 0)

kmeans.fit(X)

Y\_pred\_K= kmeans.predict(X)

#1 Hierarchial Clustering

import scipy.cluster.hierarchy as sch

sch.dendrogram(sch.linkage(X, method= 'ward'))

plt.show()

import numpy as np

import pandas as pd

from sklearn.datasets import load\_breast\_cancer

from sklearn.preprocessing import StandardScaler

standardscaler= StandardScaler()

data = load\_breast\_cancer()

X = pd.DataFrame(data.data, columns=data.feature\_names)

y = pd.Series(data.target)

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

from sklearn.linear\_model import LogisticRegression

X\_train = standardscaler.fit\_transform(X\_train)

X\_test = standardscaler.fit\_transform(X\_test)

lr = LogisticRegression(random\_state=42)

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

y\_pred = lr.predict(X\_test)

from sklearn.metrics import accuracy\_score

accuracy\_balanced = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy on balanced dataset: {accuracy\_balanced:.2f}")

from sklearn.model\_selection import StratifiedKFold

from sklearn.metrics import precision\_score, recall\_score, f1\_score

skf = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=42)

precision\_scores = []

recall\_scores = []

f1\_scores = []

for train\_idx, test\_idx in skf.split(X, y):

X\_train, X\_test = X.iloc[train\_idx], X.iloc[test\_idx]

y\_train, y\_test = y.iloc[train\_idx], y.iloc[test\_idx]

lr = LogisticRegression(random\_state=42)

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

y\_pred = lr.predict(X\_test)

precision\_scores.append(precision\_score(y\_test, y\_pred))

recall\_scores.append(recall\_score(y\_test, y\_pred))

f1\_scores.append(f1\_score(y\_test, y\_pred))

print(f"Precision on unbalanced dataset: {np.mean(precision\_scores):.2f}")

print(f"Recall on unbalanced dataset: {np.mean(recall\_scores):.2f}")

print(f"F1 score on unbalanced dataset: {np.mean(f1\_scores):.2f}")

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

d = pd.read\_csv('Mall\_customers.csv')

mat = d.iloc[:, [3, 4]].values

import scipy.cluster.hierarchy as shc

dg = shc.dendrogram(shc.linkage(mat, method="ward"))

mtp.title("Dendrogrma Plot")

mtp.ylabel("Euclidean Distances")

mtp.xlabel("Customers")

mtp.show()

from sklearn.cluster import AgglomerativeClustering

h= AgglomerativeClustering(n\_clusters=5, affinity='euclidean', linkage='ward')

Y\_p= h.fit\_predict(mat)

mtp.scatter(mat[Y\_p == 0, 0], mat[Y\_p == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

mtp.scatter(mat[Y\_p == 1, 0], mat[Y\_p == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

mtp.scatter(mat[Y\_p== 2, 0], mat[Y\_p == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

mtp.scatter(mat[Y\_p == 3, 0], mat[Y\_p == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

mtp.scatter(mat[Y\_p == 4, 0], mat[Y\_p == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()

# importing libraries

import numpy as nm

import matplotlib.pyplot as mtp

import pandas as pd

# Importing the dataset

dataset = pd.read\_csv('Mall\_customers.csv')

x = dataset.iloc[:, [3, 4]].values

#finding optimal number of clusters using the elbow method

from sklearn.cluster import KMeans

wcss\_list= [] #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.

for i in range(1, 11):

kmeans = KMeans(n\_clusters=i, init='k-means++', random\_state= 42)

kmeans.fit(x)

wcss\_list.append(kmeans.inertia\_)

mtp.plot(range(1, 11), wcss\_list)

mtp.title('The Elobw Method Graph')

mtp.xlabel('Number of clusters(k)')

mtp.ylabel('wcss\_list')

mtp.show()

#training the K-means model on a dataset

kmeans = KMeans(n\_clusters=5, init='k-means++', random\_state= 42)

y\_predict= kmeans.fit\_predict(x)

#visulaizing the clusters

mtp.scatter(x[y\_predict == 0, 0], x[y\_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1')

#for first cluster

mtp.scatter(x[y\_predict == 1, 0], x[y\_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2')

#for second cluster

mtp.scatter(x[y\_predict== 2, 0], x[y\_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3')

#for third cluster

mtp.scatter(x[y\_predict == 3, 0], x[y\_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4')

#for fourth cluster

mtp.scatter(x[y\_predict == 4, 0], x[y\_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5')

#for fifth cluster

mtp.scatter(kmeans.cluster\_centers\_[:, 0], kmeans.cluster\_centers\_[:, 1], s = 300, c = 'yellow',

label = 'Centroid')

mtp.title('Clusters of customers')

mtp.xlabel('Annual Income (k$)')

mtp.ylabel('Spending Score (1-100)')

mtp.legend()

mtp.show()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from pandas import DataFrame

from sklearn import datasets

from sklearn.mixture import GaussianMixture

mydata = datasets.load\_iris()

a = mydata.data[:, :2]

data = pd.DataFrame(a)

plt.scatter(data[0], data[1],c='black')

gmm = GaussianMixture(n\_components = 3)

gmm.fit(data)

data\_labels = gmm.predict(data)

data['labels']= data\_labels

d0 = data[data['labels']== 0]

d1 = data[data['labels']== 1]

d2 = data[data['labels']== 2]

plt.scatter(d0[0], d0[1], c ='pink')

plt.scatter(d1[0], d1[1], c ='blue')

plt.scatter(d2[0], d2[1], c ='cyan')

print(gmm.lower\_bound\_)

print(gmm.n\_iter\_)

import numpy as np

from scipy.stats import mode

data = np.array([['A', 'B', 'C'],

['B', 'C', 'A'],

['C', 'A', 'B'],

['A', 'C', 'B'],

['A', 'A', 'B']])

k = 2

modes = [['A', 'B', 'C'],

['C', 'B', 'A']]

clusters = np.zeros(data.shape[0], dtype=int)

clusters\_prev = np.zeros(data.shape[0], dtype=int)

for i in range(10):

for j, object in enumerate(data):

distances = np.array([sum(object != mode) for mode in modes])

clusters[j] = np.argmin(distances)

for j in range(k):

modes[j] = mode(data[clusters == j]).mode[0]

if (clusters == clusters\_prev).all():

break

clusters\_prev = clusters

print("The cluster assignments for each data object: ", clusters)

print("Modes for each cluster: ", modes)

# test classification dataset

from sklearn.datasets import make\_classification

# define dataset

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=15, n\_redundant=5, random\_state=7)

# summarize the dataset

print(X.shape, y.shape)

# evaluate pca with logistic regression algorithm for classification

from numpy import mean

from numpy import std

from sklearn.datasets import make\_classification

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

# define dataset

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=15, n\_redundant=5, random\_state=7)

# define the pipeline

steps = [('pca', PCA(n\_components=10)), ('m', LogisticRegression())]

model = Pipeline(steps=steps)

# evaluate model

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

n\_scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs=-1, error\_score='raise')

# report performance

print('Accuracy: %.3f (%.3f)' % (mean(n\_scores), std(n\_scores)))

from numpy import mean

from numpy import std

from sklearn.datasets import make\_classification

from sklearn.model\_selection import cross\_val\_score

from sklearn.model\_selection import RepeatedStratifiedKFold

from sklearn.pipeline import Pipeline

from sklearn.decomposition import PCA

from sklearn.linear\_model import LogisticRegression

from matplotlib import pyplot

def get\_dataset():

X, y = make\_classification(n\_samples=1000, n\_features=20, n\_informative=15, n\_redundant=5, random\_state=7)

return X, y

def get\_models():

models = dict()

for i in range(1,21):

steps = [('pca', PCA(n\_components=i)), ('m', LogisticRegression())]

models[str(i)] = Pipeline(steps=steps)

return models

def evaluate\_model(model, X, y):

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs=-1, error\_score='raise')

return scores

X, y = get\_dataset()

models = get\_models()

results, names = list(), list()

for name, model in models.items():

scores = evaluate\_model(model, X, y)

results.append(scores)

names.append(name)

print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))

pyplot.boxplot(results, labels=names, showmeans=True)

pyplot.xticks(rotation=45)

pyplot.show()

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

sns.set()

from mlxtend.plotting import plot\_decision\_regions

import missingno as msno

from pandas.plotting import scatter\_matrix

from sklearn.preprocessing import StandardScaler

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import confusion\_matrix

from sklearn import metrics

from sklearn.metrics import classification\_report

import warnings

warnings.filterwarnings('ignore')

%matplotlib inline

diabetes\_df = pd.read\_csv('diabetes.csv')

diabetes\_df.head()

diabetes\_df.columns

diabetes\_df.info()

diabetes\_df.describe()

diabetes\_df.describe().T

diabetes\_df.isnull().head(10)

diabetes\_df.isnull().sum()

diabetes\_df\_copy = diabetes\_df.copy(deep = True)

diabetes\_df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']] = diabetes\_df\_copy[['Glucose','BloodPressure','SkinThickness','Insulin','BMI']].replace(0,np.NaN)

# Showing the Count of NANs

print(diabetes\_df\_copy.isnull().sum())

p = diabetes\_df.hist(figsize = (20,20))

diabetes\_df\_copy['Glucose'].fillna(diabetes\_df\_copy['Glucose'].mean(), inplace = True)

diabetes\_df\_copy['BloodPressure'].fillna(diabetes\_df\_copy['BloodPressure'].mean(), inplace = True)

diabetes\_df\_copy['SkinThickness'].fillna(diabetes\_df\_copy['SkinThickness'].median(), inplace = True)

diabetes\_df\_copy['Insulin'].fillna(diabetes\_df\_copy['Insulin'].median(), inplace = True)

diabetes\_df\_copy['BMI'].fillna(diabetes\_df\_copy['BMI'].median(), inplace = True)

p = diabetes\_df\_copy.hist(figsize = (20,20))

p = msno.bar(diabetes\_df)

color\_wheel = {1: "#0392cf", 2: "#7bc043"}

colors = diabetes\_df["Outcome"].map(lambda x: color\_wheel.get(x + 1))

print(diabetes\_df.Outcome.value\_counts())

p=diabetes\_df.Outcome.value\_counts().plot(kind="bar")

plt.subplot(121), sns.distplot(diabetes\_df['Insulin'])

plt.subplot(122), diabetes\_df['Insulin'].plot.box(figsize=(16,5))

plt.show()

plt.figure(figsize=(12,10))

# seaborn has an easy method to showcase heatmap

p = sns.heatmap(diabetes\_df.corr(), annot=True,cmap ='RdYlGn')

diabetes\_df\_copy.head()

sc\_X = StandardScaler()

X = pd.DataFrame(sc\_X.fit\_transform(diabetes\_df\_copy.drop(["Outcome"],axis = 1),), columns=['Pregnancies',

'Glucose', 'BloodPressure', 'SkinThickness', 'Insulin', 'BMI', 'DiabetesPedigreeFunction', 'Age'])

X.head()

X = diabetes\_df.drop('Outcome', axis=1)

y = diabetes\_df['Outcome']

from sklearn.model\_selection import train\_test\_split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X,y, test\_size=0.33,

random\_state=7)

from sklearn.model\_selection import RandomizedSearchCV

from sklearn import metrics

KNN\_model = KNeighborsClassifier()

knn\_param = {"n\_neighbors": [i for i in range(1,30,5)],

"weights": ["uniform", "distance"],

"algorithm": ["ball\_tree", "kd\_tree", "brute"],

"leaf\_size": [1, 10, 30],

"p": [1,2]}

cv = RepeatedStratifiedKFold(n\_splits=10, n\_repeats=3, random\_state=1)

RCV = RandomizedSearchCV(KNN\_model, knn\_param, n\_iter=50, scoring='roc\_auc', n\_jobs=-1, cv=5, random\_state=1)

KNN = RCV.fit(X\_train, y\_train).best\_estimator\_

pred = KNN.predict(X\_test)

pred\_prob = KNN.predict\_proba(X\_test)

Classification\_Summary(pred,pred\_prob,5)

from sklearn.model\_selection import RandomizedSearchCV

KNN\_model = KNeighborsClassifier()

KNN\_model.fit(X\_train, y\_train)

from sklearn import metrics

predictions = KNN\_model.predict(X\_test)

print("Accuracy Score =", format(metrics.accuracy\_score(y\_test,predictions)))

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, predictions))

print(classification\_report(y\_test,predictions))

from sklearn.tree import DecisionTreeClassifier

dtree = DecisionTreeClassifier()

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

from sklearn import metrics

predictions = dtree.predict(X\_test)

print("Accuracy Score =", format(metrics.accuracy\_score(y\_test,predictions)))

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, predictions))

print(classification\_report(y\_test,predictions))

from sklearn.svm import SVC

svc\_model = SVC()

svc\_model.fit(X\_train, y\_train)

svc\_pred = svc\_model.predict(X\_test)

from sklearn import metrics

print("Accuracy Score =", format(metrics.accuracy\_score(y\_test, svc\_pred)))

from sklearn.metrics import classification\_report, confusion\_matrix

print(confusion\_matrix(y\_test, svc\_pred))

print(classification\_report(y\_test,svc\_pred))