**ML Lab**

1. **Implement a logistic regression model from scratch using Python and NumPy to classify the Iris dataset into its three classes. Evaluate the model's performance using cross-validation and report the accuracy, precision, recall, and F1-score. Dataset: Iris dataset (**[**https://archive.ics.uci.edu/ml/datasets/iris**](https://archive.ics.uci.edu/ml/datasets/iris)**)**

**Code:**

import numpy as np

from sklearn import datasets

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

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

from sklearn.preprocessing import StandardScaler

from sklearn.utils import shuffle

# Load Iris dataset

iris = datasets.load\_iris()

X = iris.data

y = iris.target

# Standardize features

scaler = StandardScaler()

X = scaler.fit\_transform(X)

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

class LogisticRegression:

def \_\_init\_\_(self, lr=0.01, num\_iter=100000, fit\_intercept=True, verbose=False):

self.lr = lr

self.num\_iter = num\_iter

self.fit\_intercept = fit\_intercept

self.verbose = verbose

def \_\_add\_intercept(self, X):

intercept = np.ones((X.shape[0], 1))

return np.concatenate((intercept, X), axis=1)

def \_\_sigmoid(self, z):

return 1 / (1 + np.exp(-z))

def \_\_loss(self, h, y):

return (-y \* np.log(h) - (1 - y) \* np.log(1 - h)).mean()

def fit(self, X, y):

if self.fit\_intercept:

X = self.\_\_add\_intercept(X)

# weights initialization

self.theta = np.zeros(X.shape[1])

for i in range(self.num\_iter):

z = np.dot(X, self.theta)

h = self.\_\_sigmoid(z)

gradient = np.dot(X.T, (h - y)) / y.size

self.theta -= self.lr \* gradient

if(self.verbose == True and i % 10000 == 0):

z = np.dot(X, self.theta)

h = self.\_\_sigmoid(z)

print(f'loss: {self.\_\_loss(h, y)} \t')

def predict\_prob(self, X):

if self.fit\_intercept:

X = self.\_\_add\_intercept(X)

return self.\_\_sigmoid(np.dot(X, self.theta))

def predict(self, X, threshold):

return self.predict\_prob(X) >= threshold

model = LogisticRegression(lr=0.1, num\_iter=300000)

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

predictions = model.predict(X\_test, 0.5)

print("Accuracy:", accuracy\_score(y\_test, predictions))

print("Precision:", precision\_score(y\_test, predictions, average='macro'))

print("Recall:", recall\_score(y\_test, predictions, average='macro'))

print("F1 Score:", f1\_score(y\_test, predictions, average='macro'))

1. **Develop a decision tree classifier using scikit-learn to predict whether a patient has diabetes based on various medical features in the Pima Indians Diabetes dataset. Perform hyperparameter tuning using grid search cross-validation and report the best parameters found.   
   Dataset: Pima Indians Diabetes dataset (**[**https://www.kaggle.com/uciml/pima-indians-diabetes-database**](https://www.kaggle.com/uciml/pima-indians-diabetes-database)**)**

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

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

# Load dataset

data = pd.read\_csv('./diabetes.csv') # adjust file path as needed

# Split dataset into features and target variable

X = data.drop('Outcome', axis=1)

y = data['Outcome']

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# Create Decision Tree Classifier

clf = DecisionTreeClassifier()

# Hyperparameter tuning using GridSearchCV

param\_grid = {

'criterion': ['gini', 'entropy'],

'max\_depth': [None, 2, 3, 4, 5, 6, 7, 8, 9, 10],

'min\_samples\_split': [2, 3, 4, 5, 6, 7, 8, 9, 10],

'min\_samples\_leaf': [1, 2, 3, 4, 5, 6, 7, 8, 9, 10],

}

grid\_search = GridSearchCV(estimator=clf, param\_grid=param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Print the best parameters found

print(grid\_search.best\_params\_)

# Predict the response for test dataset

y\_pred = grid\_search.predict(X\_test)

# Model Accuracy, Precision, Recall and F1 Score

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred))

print("Recall:", recall\_score(y\_test, y\_pred))

print("F1 Score:", f1\_score(y\_test, y\_pred))

1. **Build a support vector machine (SVM) classifier using scikit-learn to classify handwritten digits in the MNIST dataset. Experiment with different kernel functions and regularization parameters and compare their performance.**

**Dataset: MNIST dataset (**[**http://yann.lecun.com/exdb/mnist/**](http://yann.lecun.com/exdb/mnist/)**)**

import pandas as pd

from sklearn import datasets, svm, metrics

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

# Load MNIST dataset

digits = datasets.load\_digits()

# Split dataset into features and target variable

n\_samples = len(digits.images)

X = digits.images.reshape((n\_samples, -1))

y = digits.target

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# Create SVM Classifier

clf = svm.SVC()

# Hyperparameter tuning using GridSearchCV

param\_grid = {

'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],

'C': [0.1, 1, 10, 100, 1000]

}

grid\_search = GridSearchCV(estimator=clf, param\_grid=param\_grid, cv=5)

grid\_search.fit(X\_train, y\_train)

# Print the best parameters found

print(grid\_search.best\_params\_)

# Predict the response for test dataset

y\_pred = grid\_search.predict(X\_test)

# Model Accuracy

print("Accuracy:", metrics.accuracy\_score(y\_test, y\_pred))

1. **Create a k-nearest neighbors (KNN) classifier using scikit-learn to classify the wine quality in the Wine Quality dataset into its various quality categories. Explore the impact of different values of k on the model's performance.**

**Dataset: Wine Quality dataset (**[**https://archive.ics.uci.edu/ml/datasets/wine+quality**](https://archive.ics.uci.edu/ml/datasets/wine+quality)**)**

import pandas as pd

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score

# Load dataset

data = pd.read\_csv('./winequality-red.csv', sep=';') # adjust file path as needed

# Split dataset into features and target variable

X = data.drop('quality', axis=1)

y = data['quality']

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# Create KNN Classifier and test different values of k

for k in range(1, 11):

knn = KNeighborsClassifier(n\_neighbors=k)

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

# Predict the response for test dataset

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

# Model Accuracy

print(f"Accuracy for k={k}: ", accuracy\_score(y\_test, y\_pred))

1. **Implement a simple neural network using TensorFlow or PyTorch to classify images in the FashionMNIST dataset into their respective clothing categories. Experiment with different architectures, activation functions, and optimization algorithms to improve model performance.**

**Dataset: FashionMNIST dataset (**[**https://github.com/zalandoresearch/fashion-mnist**](https://github.com/zalandoresearch/fashion-mnist)**)**

!pip install torch torchvision

import torch

from torch import nn, optim

import torchvision

from torchvision import datasets, transforms

# Load FashionMNIST dataset

transform = transforms.Compose([transforms.ToTensor(), transforms.Normalize((0.5,), (0.5,))])

trainset = datasets.FashionMNIST('~/.pytorch/F\_MNIST\_data/', download=True, train=True, transform=transform)

trainloader = torch.utils.data.DataLoader(trainset, batch\_size=64, shuffle=True)

testset = datasets.FashionMNIST('~/.pytorch/F\_MNIST\_data/', download=True, train=False, transform=transform)

testloader = torch.utils.data.DataLoader(testset, batch\_size=64, shuffle=True)

# Define the network architecture

model = nn.Sequential(nn.Linear(784, 256),

nn.ReLU(),

nn.Linear(256, 128),

nn.ReLU(),

nn.Linear(128, 64),

nn.ReLU(),

nn.Linear(64, 10),

nn.LogSoftmax(dim=1))

# Define the loss

criterion = nn.NLLLoss()

# Define the optimizer

optimizer = optim.Adam(model.parameters(), lr=0.003)

# Train the network

epochs = 5

for e in range(epochs):

running\_loss = 0

for images, labels in trainloader:

# Flatten FashionMNIST images into a 784 long vector

images = images.view(images.shape[0], -1)

# Training pass

optimizer.zero\_grad()

output = model(images)

loss = criterion(output, labels)

# Backpropagation

loss.backward()

# Weight update

optimizer.step()

running\_loss += loss.item()

else:

print(f"Training loss: {running\_loss/len(trainloader)}")

# Test the network

correct\_count, all\_count = 0, 0

for images,labels in testloader:

for i in range(len(labels)):

img = images[i].view(1, 784)

with torch.no\_grad():

logps = model(img)

ps = torch.exp(logps)

probab = list(ps.numpy()[0])

pred\_label = probab.index(max(probab))

true\_label = labels.numpy()[i]

if(true\_label == pred\_label):

correct\_count += 1

all\_count += 1

print("Number Of Images Tested =", all\_count)

print("\nModel Accuracy =", (correct\_count/all\_count))

1. **Develop a Gaussian mixture model (GMM) using scikit-learn to perform clustering on the Seeds dataset, aiming to identify different seed types based on various seed attributes. Evaluate clustering performance using silhouette score and visualize cluster assignments. Dataset: Seeds dataset (**[**https://archive.ics.uci.edu/ml/datasets/seeds**](https://archive.ics.uci.edu/ml/datasets/seeds)**)’**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.mixture import GaussianMixture

from sklearn.metrics import silhouette\_score

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# Load dataset

data = pd.read\_csv('seeds\_dataset.txt', sep='\t', header=None, error\_bad\_lines=False)

# adjust file path as needed

# Split dataset into features and target variable

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

y = data.iloc[:, -1]

# Standardize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create GMM

gmm = GaussianMixture(n\_components=3)

gmm.fit(X\_scaled)

# Predict the cluster for each data point

y\_cluster\_gmm = gmm.predict(X\_scaled)

# Evaluate model using silhouette score

score = silhouette\_score(X\_scaled, y\_cluster\_gmm)

print('Silhouette Score: ', score)

# Visualize cluster assignments

pca = PCA(n\_components=2)

principalComponents = pca.fit\_transform(X\_scaled)

principalDf = pd.DataFrame(data=principalComponents, columns=['principal component 1', 'principal component 2'])

finalDf = pd.concat([principalDf, pd.DataFrame(y\_cluster\_gmm, columns=['cluster'])], axis=1)

fig = plt.figure(figsize=(8, 8))

ax = fig.add\_subplot(1, 1, 1)

ax.set\_xlabel('Principal Component 1', fontsize=15)

ax.set\_ylabel('Principal Component 2', fontsize=15)

ax.set\_title('2 Component PCA', fontsize=20)

colors = ['r', 'g', 'b']

for cluster, color in zip([0, 1, 2], colors):

indicesToKeep = finalDf['cluster'] == cluster

ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1'], finalDf.loc[indicesToKeep, 'principal component 2'], c=color, s=50)

ax.legend(['Cluster 1', 'Cluster 2', 'Cluster 3'])

ax.grid()

plt.show()

1. **Construct a principal component analysis (PCA) pipeline using scikit-learn to reduce the dimensionality of the Wine dataset while preserving at least 95% of the variance. Visualize the reduced dataset and compare it with the original dataset.   
   Dataset: Wine dataset (**[**https://archive.ics.uci.edu/ml/datasets/wine**](https://archive.ics.uci.edu/ml/datasets/wine)**)**

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

from sklearn.preprocessing import StandardScaler

# Load dataset

data = pd.read\_csv('wine.data', header=None) # adjust file path as needed

# Split dataset into features and target variable

X = data.iloc[:, 1:]

y = data.iloc[:, 0]

# Standardize features

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# Create PCA

pca = PCA(n\_components=0.95)

X\_pca = pca.fit\_transform(X\_scaled)

# Print the original and reduced dimensionality

print("Original dimensionality: ", X\_scaled.shape[1])

print("Reduced dimensionality: ", X\_pca.shape[1])

# Visualize the reduced dataset

plt.figure(figsize=(8, 6))

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y, cmap='viridis')

plt.xlabel('First Principal Component')

plt.ylabel('Second Principal Component')

plt.title('PCA of Wine Dataset')

plt.show()

1. **Implement a basic sentiment analysis model using natural language processing techniques in Python to classify IMDb movie reviews into positive and negative sentiments. Utilize techniques like tokenization, stop-word removal, and TF-IDF vectorization.   
   Dataset: IMDb movie reviews dataset (**[**https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews**](https://www.kaggle.com/lakshmi25npathi/imdb-dataset-of-50k-movie-reviews)**)**

import nltk

nltk.download('punkt')

nltk.download('stopwords')

import pandas as pd

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.linear\_model import LogisticRegression

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

from nltk.corpus import stopwords

from nltk.tokenize import word\_tokenize

# Load dataset

data = pd.read\_csv('./IMDB Dataset.csv') # adjust file path as needed

# Split dataset into features and target variable

X = data['review']

y = data['sentiment']

# Tokenization and stop-word removal

stop\_words = set(stopwords.words('english'))

X = X.apply(lambda x: ' '.join([word for word in word\_tokenize(x) if word not in stop\_words]))

# Split dataset into training set and test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1)

# TF-IDF vectorization

vectorizer = TfidfVectorizer()

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

X\_test = vectorizer.transform(X\_test)

# Create Logistic Regression Classifier

clf = LogisticRegression()

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

# Predict the response for test dataset

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

# Model Accuracy, Precision, Recall and F1 Score

print("Accuracy:", accuracy\_score(y\_test, y\_pred))

print("Precision:", precision\_score(y\_test, y\_pred, pos\_label='positive'))

print("Recall:", recall\_score(y\_test, y\_pred, pos\_label='positive'))

print("F1 Score:", f1\_score(y\_test, y\_pred, pos\_label='positive'))

**Lab Questions**

**Assignment-1:**

import numpy as np

# Create an empty array

empa = np.empty((3, 3), dtype=int)

print("Empty Array")

print(empa)

# Create a full array

flla = np.full([3, 3], 55, dtype=int)

print("\n Full Array")

print(flla)

# Importing the NumPy library with an alias 'np'

import numpy as np

# Generating an array of 25 random numbers from a standard normal distribution using np.random.normal()

rand\_num = np.random.normal(0, 1, 25)

# Printing a message indicating 25 random numbers from a standard normal distribution

print("25 random numbers from a standard normal distribution:")

# Printing the array of 15 random numbers

print(rand\_num)

import numpy.matlib

import numpy as np

a = np.array([[1,2],[3,4]])

b = np.array([[11,12],[13,14]])

np.dot(a,b)

# Importing the NumPy library with an alias 'np'

import numpy as np

# Creating a NumPy array 'nums' containing values in a 3x3 matrix

nums = np.array([[3, 7, 1],

[10, 3, 2],

[5, 6, 7]])

# Printing a message indicating the original array

print("Original array:")

print(nums)

# Sorting the array 'nums' by rows in ascending order and displaying the sorted result

print("\nSort the said array by row in ascending order:")

print(np.sort(nums))

# Sorting the array 'nums' by columns in ascending order and displaying the sorted result

print("\nSort the said array by column in ascending order:")

print(np.sort(nums, axis=0))

# Python code to find mean of every numpy array in list

# Importing module

import numpy as np

# List Initialization

Input = [np.array([1, 2, 3]),

np.array([4, 5, 6]),

np.array([7, 8, 9])]

# Output list initialization

Output = []

# using np.mean()

for i in range(len(Input)):

Output.append(np.mean(Input[i]))

# Printing output

print(Output)

import numpy as np

# Original Array

array = np.array(['PHP C# Python C Java C++'], dtype=np.str)

print(array)

# Split the element of the said array with spaces

sparr = np.char.split(array)

print(sparr)

**Assignment-2: Company\_sales\_data.csv**

import pandas as pd

import matplotlib.pyplot as plt  
df = pd.read\_csv('/content/company\_sales\_data.csv')  
plt.figure(figsize=(8, 5))

plt.plot(df['month\_number'], df['total\_profit'])

plt.xlabel('Month Number')

plt.ylabel('Total profit')

plt.title('Total Profit of All Months (Line Plot)')

plt.show()  
plt.figure(figsize=(8, 5))

plt.plot(df['month\_number'], df['total\_units'], linestyle='dashed', color='red', marker='o', markersize=8, label='Sold Units')

plt.xlabel('Month Number')

plt.ylabel('Sold units number')

plt.legend(loc='lower right')

plt.title('Total Profit with Style (Line Plot)')

plt.show()  
plt.figure(figsize=(10, 6))

for product in df.columns[1:7]:

plt.plot(df['month\_number'], df[product], label=product)

plt.xlabel('Month Number')

plt.ylabel('Number of Units Sold')

plt.legend()

plt.title('Number of Units Sold per Month for Each Product (Multi-line Plot)')

plt.show()  
plt.figure(figsize=(8, 5))

plt.scatter(df['month\_number'], df['toothpaste'], marker='o')

plt.xlabel('Month Number')

plt.ylabel('Toothpaste Sales')

plt.title('Toothpaste Sales per Month (Scatter Plot)')

plt.grid(True, linestyle='--')

plt.show()  
plt.figure(figsize=(10, 6))

bar\_width = 0.35

bar\_positions = range(len(df['month\_number']))

plt.bar(bar\_positions, df['facecream'], width=bar\_width, label='Face Cream')

plt.bar([p + bar\_width for p in bar\_positions], df['facewash'], width=bar\_width, label='Facewash')

plt.xlabel('Month Number')

plt.ylabel('Number of Units Sold')

plt.title('Product Sales per Month (Bar Chart)')

plt.xticks([p + bar\_width / 2 for p in bar\_positions], df['month\_number'])

plt.legend()

plt.show()

**Assignment-3: IRIS.csv Dataset**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv('./Machine Learning Lab Assignment-3 Dataset Iris.csv') # adjust file path as needed

# a. Show size of the dataset

print("Size of the dataset: ", data.shape)

# b. Show datatype for each column

print("Datatype for each column: \n", data.dtypes)

# c. Show distribution of data

print("Distribution of data: \n", data.describe())

# d. Check if there are any null values

print("Null values in dataset: \n", data.isnull().sum())

# e. Check for duplicates

print("Number of duplicate rows: ", data.duplicated().sum())

# f. Check the number of instances for each species of flower

print("Instances for each species: \n", data['Species'].value\_counts())

# g. Compare sepal length and sepal width

sns.jointplot(x='sepal\_length', y='SepalWidthCm', data=data, hue='Species')

plt.show()

# h. Compare petal length and petal width

sns.jointplot(x='petal\_length', y='PetalWidthCm', data=data, hue='Species')

plt.show()

# i. Use pairplot to show all comparisons

sns.pairplot(data, hue='Species')

plt.show()

# j. Use histograms to compare sepal length, sepal width, petal length and petal width across the species

data.hist(by='Species', figsize=(15,15))

plt.show()

# k. Use boxplot to show distribution of data across the species

plt.figure(figsize=(15,10))

plt.subplot(2,2,1)

sns.boxplot(x = 'Species', y = 'SepalLengthCm', data = data)

plt.subplot(2,2,2)

sns.boxplot(x = 'Species', y = 'SepalWidthCm', data = data)

plt.subplot(2,2,3)

sns.boxplot(x = 'Species', y = 'PetalLengthCm', data = data)

plt.subplot(2,2,4)

sns.boxplot(x = 'Species', y = 'PetalWidthCm', data = data)

plt.show()

# l. Use violinplot to show distribution of data across the species

plt.figure(figsize=(15,10))

plt.subplot(2,2,1)

sns.violinplot(x = 'Species', y = 'SepalLengthCm', data = data)

plt.subplot(2,2,2)

sns.violinplot(x = 'Species', y = 'SepalWidthCm', data = data)

plt.subplot(2,2,3)

sns.violinplot(x = 'Species', y = 'PetalLengthCm', data = data)

plt.subplot(2,2,4)

sns.violinplot(x = 'Species', y = 'PetalWidthCm', data = data)

plt.show()

**Assignment-4:**

1. **Balance.csv Analysis**

import numpy as np

import pandas as pd

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

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.tree import plot\_tree

import matplotlib.pyplot as plt

def importdata():

balance\_data = pd.read\_csv(

'balance.csv',header=None)

print("Dataset Length: ", len(balance\_data))

print("Dataset Shape: ", balance\_data.shape)

print("Dataset: ", balance\_data.head())

return balance\_data

def splitdataset(balance\_data):

X = balance\_data.values[:, 1:10]

Y = balance\_data.values[:, 0]

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, Y, test\_size=0.3, random\_state=100)

return X, Y, X\_train, X\_test, y\_train, y\_test

def train\_using\_gini(X\_train, X\_test, y\_train):

clf\_gini = DecisionTreeClassifier(criterion="gini",

random\_state=100, max\_depth=3, min\_samples\_leaf=5)

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

return clf\_gini

def train\_using\_entropy(X\_train, X\_test, y\_train):

# Decision tree with entropy

clf\_entropy = DecisionTreeClassifier(

criterion="entropy", random\_state=100,

max\_depth=3, min\_samples\_leaf=5)

# Performing training

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

return clf\_entropy

# Function to make predictions

def prediction(X\_test, clf\_object):

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

print("Predicted values:")

print(y\_pred)

return y\_pred

# Placeholder function for cal\_accuracy

def cal\_accuracy(y\_test, y\_pred):

print("Confusion Matrix: ",

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

print("Accuracy : ",

accuracy\_score(y\_test, y\_pred)\*100)

print("Report : ",

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

# Function to plot the decision tree

def plot\_decision\_tree(clf\_object, feature\_names, class\_names):

plt.figure(figsize=(15, 10))

plot\_tree(clf\_object, filled=True, feature\_names=feature\_names, class\_names=class\_names, rounded=True)

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

data = importdata()

X, Y, X\_train, X\_test, y\_train, y\_test = splitdataset(data)

clf\_gini = train\_using\_gini(X\_train, X\_test, y\_train)

clf\_entropy = train\_using\_entropy(X\_train, X\_test, y\_train)

# Visualizing the Decision Trees

plot\_decision\_tree(clf\_gini, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])

plot\_decision\_tree(clf\_entropy, ['X1', 'X2', 'X3', 'X4'], ['L', 'B', 'R'])

# Operational Phase

print("Results Using Gini Index:")

y\_pred\_gini = prediction(X\_test, clf\_gini)

cal\_accuracy(y\_test, y\_pred\_gini)

print("Results Using Entropy:")

y\_pred\_entropy = prediction(X\_test, clf\_entropy)

cal\_accuracy(y\_test, y\_pred\_entropy)

1. **Bank Accuracy**

# Import necessary libraries

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

from sklearn.preprocessing import OneHotEncoder

from sklearn.compose import ColumnTransformer

# Load your custom dataset from CSV

file\_path = "./bank.csv"

df = pd.read\_csv(file\_path)

# Assume the target variable is in a column named 'target'

X = df.drop('balance', axis=1)

y = df['balance']

categorical\_cols = [col for col in X.columns if X[col].dtype == 'object']

# Use one-hot encoding to convert categorical variables into numerical format

preprocessor = ColumnTransformer(

transformers=[

('cat', OneHotEncoder(), categorical\_cols)

],

remainder='passthrough'

)

X = preprocessor.fit\_transform(X)

# Split the dataset into training and testing sets

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

# Create a Decision Tree Classifier

clf = DecisionTreeClassifier()

# Train the classifier on the training data

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

# Make predictions on the testing set

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

# Evaluate the model

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

print(f"Accuracy: {accuracy}")

**Assignment-5: Used Car Dataset**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.linear\_model import LinearRegression

# 1. Data Preprocessing

df = pd.read\_csv('Used Car Dataset.csv')

# Handle missing values

df.fillna(method='ffill', inplace=True)

# Handle outliers

df = df[(np.abs(df['price(in lakhs)'] - df['price(in lakhs)'].mean()) <= (3\*df['price(in lakhs)'].std()))]

# 2. Data Visualization

plt.hist(df['price(in lakhs)'])

plt.boxplot(df['price(in lakhs)'])

# 3. Exploratory Data Analysis

print(df.describe())

print(df.isnull().sum())

# 4. Feature Selection

features = ['kms\_driven', 'mileage(kmpl)', 'engine(cc)', 'max\_power(bhp)', 'torque(Nm)']

target = ['price(in lakhs)']

X = df[features]

y = df[target]

# 5. Train-Test Split

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

# 6. Model Creation

model = LinearRegression()

# 7. Model Training

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

# 8. Model Prediction

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

print(y\_pred)

**Assignment-6: Diabetes.csv**

# Importing libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix, ConfusionMatrixDisplay

# Load the dataset

diabetes\_data = pd.read\_csv('./diabetes.csv')

# Display the first few rows of the dataset

print(diabetes\_data.head())

# Preprocessing the data

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

y = diabetes\_data['Outcome']

# Splitting the data into training and testing sets

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

# Standardize features by removing the mean and scaling to unit variance

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

# Training the logistic regression model

log\_reg = LogisticRegression()

log\_reg.fit(X\_train\_scaled, y\_train)

# Making predictions on the testing set

y\_pred = log\_reg.predict(X\_test\_scaled)

# Visualizing the confusion matrix

cm = confusion\_matrix(y\_test, y\_pred)

labels = ['Not Diabetic', 'Diabetic']

disp = ConfusionMatrixDisplay(confusion\_matrix=cm, display\_labels=labels)

disp.plot(cmap=plt.cm.Blues)

plt.title('Confusion Matrix')

plt.show()

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

# Calculate accuracy

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

# Calculate precision

precision = precision\_score(y\_test, y\_pred)

# Calculate recall

recall = recall\_score(y\_test, y\_pred)

# Calculate F1 score

f1 = f1\_score(y\_test, y\_pred)

# Print the metrics

print(f"Accuracy: {accuracy}")

print(f"Precision: {precision}")

print(f"Recall: {recall}")

print(f"F1 Score: {f1}")

**Assignment-7:**

1. **KNN Algo with Wine Dataset**

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt

# Load dataset

data = pd.read\_csv('./data\_iris.csv') # adjust file path as needed

# a. Show size of the dataset

print("Size of the dataset: ", data.shape)

# b. Show datatype for each column

print("Datatype for each column: \n", data.dtypes)

# c. Show distribution of data

print("Distribution of data: \n", data.describe())

# d. Check if there are any null values

print("Null values in dataset: \n", data.isnull().sum())

# e. Check for duplicates

print("Number of duplicate rows: ", data.duplicated().sum())

# f. Check the number of instances for each species of flower

print("Instances for each species: \n", data['Species'].value\_counts())

# g. Compare sepal length and sepal width

sns.jointplot(x='SepalLengthCm', y='SepalWidthCm', data=data, hue='Species')

plt.show()

# h. Compare petal length and petal width

sns.jointplot(x='PetalLengthCm', y='PetalWidthCm', data=data, hue='Species')

plt.show()

# i. Use pairplot to show all comparisons

sns.pairplot(data, hue='Species')

plt.show()

# j. Use histograms to compare sepal length, sepal width, petal length and petal width across the species

data.hist(by='Species', figsize=(15,15))

plt.show()

# k. Use boxplot to show distribution of data across the species

plt.figure(figsize=(15,10))

plt.subplot(2,2,1)

sns.boxplot(x = 'Species', y = 'SepalLengthCm', data = data)

plt.subplot(2,2,2)

sns.boxplot(x = 'Species', y = 'SepalWidthCm', data = data)

plt.subplot(2,2,3)

sns.boxplot(x = 'Species', y = 'PetalLengthCm', data = data)

plt.subplot(2,2,4)

sns.boxplot(x = 'Species', y = 'PetalWidthCm', data = data)

plt.show()

# l. Use violinplot to show distribution of data across the species

plt.figure(figsize=(15,10))

plt.subplot(2,2,1)

sns.violinplot(x = 'Species', y = 'SepalLengthCm', data = data)

plt.subplot(2,2,2)

sns.violinplot(x = 'Species', y = 'SepalWidthCm', data = data)

plt.subplot(2,2,3)

sns.violinplot(x = 'Species', y = 'PetalLengthCm', data = data)

plt.subplot(2,2,4)

sns.violinplot(x = 'Species', y = 'PetalWidthCm', data = data)

plt.show()

1. **KNN Algo with Heart Dataset**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.neighbors import KNeighborsClassifier, KNeighborsRegressor

from sklearn.metrics import accuracy\_score, mean\_squared\_error

data = pd.read\_csv('./heart.csv')

label\_encoder = LabelEncoder()

data['status'] = label\_encoder.fit\_transform(data['status'])

X\_classification = data.drop(['target', 'status'], axis=1)

y\_classification = data['status']

X\_regression = data.drop(['target', 'status'], axis=1)

y\_regression = data['target']

X\_train\_classification, X\_test\_classification, y\_train\_classification, y\_test\_classification = train\_test\_split(

X\_classification, y\_classification, test\_size=0.2, random\_state=46)

X\_train\_regression, X\_test\_regression, y\_train\_regression, y\_test\_regression = train\_test\_split(

X\_regression, y\_regression, test\_size=0.2, random\_state=46)

scaler = StandardScaler()

X\_train\_classification\_scaled = scaler.fit\_transform(X\_train\_classification)

X\_test\_classification\_scaled = scaler.transform(X\_test\_classification)

X\_train\_regression\_scaled = scaler.fit\_transform(X\_train\_regression)

X\_test\_regression\_scaled = scaler.transform(X\_test\_regression)

accuracies\_classification = []

for k in range(1, 23):

knn\_classifier = KNeighborsClassifier(n\_neighbors=k)

knn\_classifier.fit(X\_train\_classification\_scaled, y\_train\_classification)

y\_pred\_classification = knn\_classifier.predict(X\_test\_classification\_scaled)

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

accuracies\_classification.append(accuracy)

mean\_errors\_regression = []

for k in range(1, 23):

knn\_regressor = KNeighborsRegressor(n\_neighbors=k)

knn\_regressor.fit(X\_train\_regression\_scaled, y\_train\_regression)

y\_pred\_regression = knn\_regressor.predict(X\_test\_regression\_scaled)

mean\_error = mean\_squared\_error(y\_test\_regression, y\_pred\_regression)

mean\_errors\_regression.append(mean\_error)

plt.figure(figsize=(10, 5))

plt.subplot(1, 2, 1)

plt.plot(range(1, 23), accuracies\_classification, marker='o')

plt.title('Accuracy vs. k for Classification')

plt.xlabel('k')

plt.ylabel('Accuracy')

plt.xticks(range(1, 23))

plt.grid(True)

plt.subplot(1, 2, 2)

plt.plot(range(1, 23), mean\_errors\_regression, marker='o')

plt.title('Mean Error vs. k for Regression')

plt.xlabel('k')

plt.ylabel('Mean Error')

plt.xticks(range(1, 23))

plt.grid(True)

plt.tight\_layout()

plt.show()

**Assignment-8:**

**Q1. You are given a dataset (data.csv) containing two columns - 'X' (independent variable) and 'y' (dependent variable). Implement a simple linear regression model using Python to predict 'y' based on 'X'. Load the dataset, split it into training and testing sets, and train the linear regression model. Finally, predict the values for the test set and calculate the mean squared error.**

import pandas as pd

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

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error

# Step 1: Load the dataset

data = pd.read\_csv('data.csv')

# Step 2: Split the dataset into training and testing sets

X = data[['x']] # Features (independent variables) - DataFrame is expected

y = data['y'] # Target variable

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

# Step 3: Train the linear regression model

model = LinearRegression()

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

# Step 4: Predict the 'y' values for the test set

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

# Step 5: Calculate the mean squared error

mse = mean\_squared\_error(y\_test, y\_pred)

print(f'Mean Squared Error: {mse}')

**Q2. You are provided with a dataset (data.csv) containing three columns - 'feature1', 'feature2', and 'target' (binary classification). Implement a Decision Tree classifier using Python to predict the ‘target' based on the features. Load the dataset, split it into training and testing sets, and train the Decision Tree model. Finally, predict the values for the test set and calculate the accuracy.**

import pandas as pd

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

# Step 1: Load the dataset

data = pd.read\_csv('data1.csv')

# Step 2: Prepare the data for training and testing

features = data[['feature1', 'feature2']] # Independent variables

target = data['target'] # Dependent variable (target)

# Splitting the dataset into training and testing sets

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

# Step 3: Train the Decision Tree classifier

classifier = DecisionTreeClassifier()

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

# Step 4: Predict the 'target' values for the test set

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

# Step 5: Calculate the accuracy

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

print(f'Accuracy: {accuracy}')

**Q3. You are analysing satellite images to identify different land cover types. You plan to use K-Means clustering to segment the image into regions with similar pixel intensities. Plot the diagram also.**

import numpy as np

import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

import imageio

def load\_image(image\_path):

"""Load an image into a numpy array."""

img = imageio.imread(image\_path)

return img

def apply\_kmeans\_clustering(image, n\_clusters=3):

"""Applies K-Means clustering to segment the image."""

# Reshape the image to a 2D array of pixels and 3 color values (RGB)

pixels = image.reshape(-1, 3)

# Apply K-Means clustering

kmeans = KMeans(n\_clusters=n\_clusters, random\_state=42)

kmeans.fit(pixels)

# Replace each pixel value with its corresponding cluster center value

clustered\_img = kmeans.cluster\_centers\_[kmeans.labels\_].reshape(image.shape)

return clustered\_img

def plot\_images(original\_img, clustered\_img):

"""Plots the original and clustered images side by side."""

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)

plt.imshow(original\_img)

plt.title('Original Image')

plt.subplot(1, 2, 2)

plt.imshow(clustered\_img.astype(np.uint8))

plt.title('Image Segmented with K-Means')

plt.show()

if \_\_name\_\_ == "\_\_main\_\_":

# Load the image

image\_path = 'image.png' # Change this to the path of your image

img = load\_image(image\_path)

# Apply K-Means clustering to segment the image

n\_clusters = 4 # Change this number based on your needs

clustered\_img = apply\_kmeans\_clustering(img, n\_clusters=n\_clusters)

# Plot the original and clustered images

plot\_images(img, clustered\_img)

**Q4.** **You are given a dataset (data.csv) containing three columns - 'feature1', 'feature2', and 'target' (binary classification). Implement a Random Forest classifier using Python to predict the 'target' based on the features. Load the dataset, split it into training and testing sets, and train the Random Forest model. Finally, predict the values for the test set and calculate the accuracy.**

import pandas as pd

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

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score

# Load the dataset

data = pd.read\_csv('data1.csv')

# Split the dataset into features and target variable

X = data[['feature1', 'feature2']] # Features

y = data['target'] # Target variable

# Splitting the dataset into the Training set and Test set

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

# Initialize the Random Forest Classifier

random\_forest\_classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Fit the model on the training data

random\_forest\_classifier.fit(X\_train, y\_train)

# Predict the target values for the test set

y\_pred = random\_forest\_classifier.predict(X\_test)

# Calculate and print the accuracy

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

print(f'Accuracy: {accuracy:.2f}')

**Q5. Scenario: You are building a machine learning model to classify handwritten digits from the MNIST dataset. Import libraries and data: Import necessary libraries like pandas, scikit-learn, and matplotlib. Load the MNIST dataset using sklearn datasets.**  
  
import pandas as pd

import matplotlib.pyplot as plt

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score

# Load the MNIST dataset

digits = datasets.load\_digits()

# Display the shape of data and target

print("Data Shape:", digits.data.shape)

print("Label Shape:", digits.target.shape)

# Visualize the first digit

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

plt.imshow(digits.images[0], cmap='gray')

plt.title(f'Label: {digits.target[0]}')

plt.show()

**Assignment-9:**

1. **Calculation for Wine.csv**

from sklearn import datasets

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron, LogisticRegression

from sklearn.svm import SVC

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

# Load wine dataset

wine = datasets.load\_wine()

X = wine.data

y = wine.target

# Split the dataset into a training set and a test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

# Standardize the features

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

# Define the classifiers

classifiers = [Perceptron(eta0=0.1, random\_state=1),

SVC(kernel='linear', C=1.0, random\_state=1),

LogisticRegression(C=1.0, random\_state=1)]

# Train the classifiers and test their performance

for clf in classifiers:

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

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

print('\nClassifier:', clf.\_\_class\_\_.\_\_name\_\_)

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred))

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

1. **Calculation For Data.csv**

import pandas as pd

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

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron, LogisticRegression

from sklearn.svm import SVC

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

# Load dataset from CSV file

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

X = df.iloc[:, :-1].values

y = df.iloc[:, -1].values

# Split the dataset into a training set and a test set

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1)

# Standardize the features

sc = StandardScaler()

sc.fit(X\_train)

X\_train\_std = sc.transform(X\_train)

X\_test\_std = sc.transform(X\_test)

# Define the classifiers

classifiers = [Perceptron(eta0=0.1, random\_state=1),

SVC(kernel='linear', C=1.0, random\_state=1),

LogisticRegression(C=1.0, random\_state=1)]

# Train the classifiers and test their performance

for clf in classifiers:

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

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

print('\nClassifier:', clf.\_\_class\_\_.\_\_name\_\_)

print('Accuracy: %.2f' % accuracy\_score(y\_test, y\_pred))

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

**Assignment-10: Passive Aggressive Regressor for Instagram**

!pip install chardet

import chardet

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import plotly.express as px

import os

from wordcloud import WordCloud, STOPWORDS, ImageColorGenerator

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

from sklearn.linear\_model import PassiveAggressiveRegressor

data = pd.read\_csv("./Instagram\_data.csv", encoding = 'cp1252')

print(data.head())

data.isnull().sum()

data = data.dropna()

data.info()

plt.figure(figsize=(10, 8))

plt.style.use('fivethirtyeight')

plt.title("Distribution of Impressions From Home")

sns.distplot(data['From Home'])

plt.show()

plt.figure(figsize=(10, 8))

plt.title("Distribution of Impressions From Hashtags")

sns.distplot(data['From Hashtags'])

plt.show()

plt.figure(figsize=(10, 8))

plt.title("Distribution of Impressions From Explore")

sns.distplot(data['From Explore'])

plt.show()

home = data["From Home"].sum()

hashtags = data["From Hashtags"].sum()

explore = data["From Explore"].sum()

other = data["From Other"].sum()

labels = ['From Home','From Hashtags','From Explore','Other']

values = [home, hashtags, explore, other]

fig = px.pie(data, values=values, names=labels,

title='Impressions on Instagram Posts From Various Sources', hole=0.5)

fig.show()

text = " ".join(i for i in data.Caption)

stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background\_color="white").generate(text)

plt.style.use('classic')

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

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

text = " ".join(i for i in data.Hashtags)

stopwords = set(STOPWORDS)

wordcloud = WordCloud(stopwords=stopwords, background\_color="white").generate(text)

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

plt.imshow(wordcloud, interpolation='bilinear')

plt.axis("off")

plt.show()

figure = px.scatter(data\_frame = data, x="Impressions",

y="Likes", size="Likes", trendline="ols",

title = "Relationship Between Likes and Impressions")

figure.show()

figure = px.scatter(data\_frame = data, x="Impressions",

y="Comments", size="Comments", trendline="ols",

title = "Relationship Between Comments and Total Impressions")

figure.show()

figure = px.scatter(data\_frame = data, x="Impressions",

y="Shares", size="Shares", trendline="ols",

title = "Relationship Between Shares and Total Impressions")

figure.show()

figure = px.scatter(data\_frame = data, x="Impressions",

y="Saves", size="Saves", trendline="ols",

title = "Relationship Between Post Saves and Total Impressions")

figure.show()

conversion\_rate = (data["Follows"].sum() / data["Profile Visits"].sum()) \* 100

print(conversion\_rate)

figure = px.scatter(data\_frame = data, x="Profile Visits",

y="Follows", size="Follows", trendline="ols",

title = "Relationship Between Profile Visits and Followers Gained")

figure.show()

x = np.array(data[['Likes', 'Saves', 'Comments', 'Shares',

'Profile Visits', 'Follows']])

y = np.array(data["Impressions"])

xtrain, xtest, ytrain, ytest = train\_test\_split(x, y, test\_size=0.2, random\_state=42)

model = PassiveAggressiveRegressor()

model.fit(xtrain, ytrain)

model.score(xtest, ytest)

# Features = [['Likes','Saves', 'Comments', 'Shares', 'Profile Visits', 'Follows']]

features = np.array([[282.0, 233.0, 4.0, 9.0, 165.0, 54.0]])

model.predict(features)