EXPERIMENT 1a: To solve the real-world problems using the following machine learning methods:

a) Linear regression

PROGRAM:

import pandas as pd *# used to see the data in an understandable manner*

import numpy as np *# used for the mathematical calculations on arrays*

from sklearn import datasets *# this module is used to get down the dataset which are available in sklearn*

from sklearn.model_selection import train_test_split *# this is the module where importing of train_test_split is done to get the data split into training and testing data*

 # available inside model_selection of sklearn

from sklearn.linear_model import LinearRegression *# this is the module to import the LinearRegression which will be performing the fitting of data and performing the necessary*

 # calculations

data = datasets.load_iris() *# this command is used to load the data from datasets module and the dataset is of iris dataset*

x_labels = data.data *# x_labels individually*

y_labels = data.target *# y_labels individually*

x_train,x_test,y_train,y_test = train_test_split(x_labels,y_labels, test_size = 0.25,random_state = 0)

 # this statement is used to get the work done for training and testing data and testing_size of data is 0.25% of total_data, and random_state means the series of data will be same

clf = LinearRegression() *# this is the way to create a LinearRegression obj which will work*

clf.fit(x_train,y_train) *# this is the command to train the data i.e, on training data(x and y)*

m = clf.coef_ *# this command is used to get the m value (which is the coefficient val in equation of line i.e, y = m*x + c)*

c = clf.intercept_ *# this command is used to get the c value (which is the intercept val in equation of line i.e., y = m*x + c)*

predictions = clf.predict(x_test) *# this is way to get the predictions (need not to pass the y_test, the regressor automatically generates the y_predictions)*

score = clf.score(x_test,y_test) *# to check the score or accuracy, this is the part where the testing data both x and y have to be passed and after generating the y_predictions*

 # it compares the predicted val with the y actual val and give the accuracy accordingly

print(score)

EXPERIMENT 1b: To solve the real-world problems using the following machine learning methods:

b) Logistic regression

PROGRAM:

from sklearn import datasets *# this command is used to get the dataset from sklearn's dataset module*

import pandas as pd *# importing pandas to view the dataset and understand properly*

from sklearn.model_selection import train_test_split *# this module is used to get the training and testing data from dataset*

from sklearn.linear_model import LogisticRegression *# importing the LogisticRegressor from linear model of sklearn*

data = datasets.load_iris() *# to load the dataset which is iris*

x_labels = data.data *# getting the x_labels individually*

y_labels = data.target *# getting the y_labels individually*

dataset = pd.DataFrame(x_labels) *# this is how to convert the arrays into dataframe of pandas*

dataset.columns = data.feature_names *# setting up the col names for each columns in dataframe*

dataset['target'] = y_labels *# inserting the y_labels into the dataframes (the last col as target col)*

x_train,x_test,y_train,y_test = train_test_split(x_labels,y_labels,test_size = 0.3,random_state = 1) *# this is the command to get the training and testing data*

 # with testing data size of 30 % of total_data and this is performed with random split

clf = LogisticRegression() *# same as to that of linear Regression , this is how importing of LogisticRegression is done*

clf.fit(x_train,y_train) *# training the model classifier*

predictions = clf.predict(x_test) *# these are the prediction values*

score = clf.score(x_test,y_test) *# this command is used to get the score (accuracy) by the model working on the testing data*

print(score) *# this is the score*

EXPERIMENT 2: To Implement support vector machine

PROGRAM:

```
from sklearn import datasets

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.model_selection import train_test_split

from sklearn.svm import SVC

dataset = datasets.load_iris()

frame = pd.DataFrame(dataset.data)

frame.columns = dataset.feature_names

frame["target"] = dataset.target

x_train,x_test,y_train,y_test = train_test_split(dataset.data,dataset.target,random_state = 0,test_size =
0.3)

svc = SVC()

svc.fit(x_train,y_train)

predictions = svc.predict(x_test)

svc.score(x_test,y_test)

svc1 = SVC(kernel = 'linear', C = 0.25)

svc1.fit(x_train,y_train)

predictions1 = svc.predict(x_test)

print(svc1.score(x_test,y_test))
```

EXPERIMENT 3a: To Implement K-Means clustering

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
# Generate some sample data
X = np.random.randn(100, 2) * 5
# Create an instance of the KMeans class with 3 clusters
kmeans = KMeans(n_clusters=3)
# Fit the data to the KMeans model
kmeans.fit(X)
# Get the cluster labels for each data point
labels = kmeans.predict(X)
# Get the centroids for each cluster
centroids = kmeans.cluster_centers_
# Plot the data points with different colors for each cluster
plt.scatter(X[:,0], X[:,1], c=labels)
# Plot the centroids as black circles
plt.scatter(centroids[:,0], centroids[:,1], marker='o', s=100, linewidths=2, color='black')
plt.show()
```

EXPERIMENT 3b: To Implement PCA

PROGRAM:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
from sklearn.decomposition import PCA
# Load the digits dataset
digits = load_digits()
X = digits.data
y = digits.target
# Create an instance of the PCA class with 2 components
pca = PCA(n_components=2)
# Fit the data to the PCA model
pca.fit(X)
# Transform the data to 2 dimensions
X_2d = pca.transform(X)
# Plot the transformed data
plt.scatter(X_2d[:,0], X_2d[:,1], c=y, cmap='viridis')
plt.colorbar()
plt.show()
```

EXPERIMENT 4: Implementation of map reduce

PROGRAM:

```
from mrjob.job import MRJob
import re
# Define a class that inherits from MRJob
class WordCount(MRJob):
    # Define the map step
    def mapper(self, _, line):
        # Split the line into words
        words = re.findall(r'\w+', line.lower())
        # Yield each word with a count of 1
        for word in words:
            yield word, 1
    # Define the reduce step
    def reducer(self, word, counts):
        # Sum up the counts for each word
        total = sum(counts)
        # Yield the word with its total count
        yield word, total
# Define the input file
input_file = 'large_text_file.txt'
# Create an instance of the WordCount class and run the job
job = WordCount(args=[input_file])
output = job.run()
# Print the top 10 most frequent words
word_counts = [(word, int(count)) for word, count in output]
word_counts_sorted = sorted(word_counts, key=lambda x: x[1], reverse=True)
for word, count in word_counts_sorted[:10]:
    print(word, count)
```

EXPERIMENT 5: Implementation of Naïve Bayes

PROGRAM:

```
from sklearn.datasets import fetch_20newsgroups
import string
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from sklearn.model_selection import train_test_split
import numpy as np
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report
newsgroups=fetch_20newsgroups()
stops=set(stopwords.words('english'))
punctuations=list(string.punctuation)
stops.update(punctuations)
all_documents=newsgroups.data
all_categories=newsgroups.target
#dividing into words as we have to work with words not with sentences
all_documents_modified=[word_tokenize(doc) for doc in all_documents]
#splitting into training and testing
x_train, x_test, y_train, y_test=train_test_split(all_documents_modified, all_categories, random_state=1)
all_words=[]
#this list is going to contain all the words from all our tokenized documents which we will use for counting the frequency
#and unnecessary stopswords and punctuations are removed as they dont make sense
for doc in x_train:
    for word in doc:
        # removing unncecessary words
        if (word.lower() not in stops) and len(word)!=1 and len(word)!=2 and word[0]!="'" and word!="n't" and word[0]!=".":
            #appending necessary words or features
            all_words.append(word)
# this functions returns the frequency of all the words from all_words which we will use as features for our classifier
```

```
def freq_dict(all_words):
    dic=dict()
    #it iterates through all the elements in the list and increases the frequency by one if it encounters the same element.
    for word in all_words:
        if word in dic.keys():
            dic[word]+=1
        else:
            dic[word]=1
    return dic
dic=freq_dict(all_words)
#diving the freq and words into two lists and sorting them into deccreasing order of frequency
#to get the maximum frequency words.
freq=np.array([i for i in dic.values()])
words=np.array([i for i in dic.keys()])
words=words[np.argsort(freq)][::-1]
for i in range(50):
    print(words[i])
# taking only the releavent words as features since the most common are useless so that is why we are taking from 20 onwards
# upto 10,000 top words
features=words[20:10000]
# It takes the patameters x_train or x_test and the list of all features and converts it into a 2-D array which contains the frequency of that feature
# in that particular document, where rows are the documents and columns are the features.
def data_modifier(x_data, features):
    modified_data=np.zeros((len(x_data), len(features)))
    #creating the empty 2d array
    for i in range(len(x_data)):
        #looping over each and every row in the x_data
        current_doc=x_data[i]
        #current_doc contains the current document on which we are iterating.(As the name suggests obviously)
        d=dict()
        #this dictionary contains the frequency of all the elements in our current_doc.
```

```
        for word in current_doc:
            if word in d.keys():
                d[word]+=1
            else:
                d[word]=1
        #dictionary created
        for j in range(len(features)):
            #now for each feature in features we will insert the value of the dictionary for the corresponding. that is,
            #the frequency of each feature in that current document.
            if features[j] in d.keys():
                modified_data[i][j]=d[features[j]]
            else:
                continue
    #finally I have returned the modified array.
    return modified_data
x_train_modified = data_modifier(x_train, features)
x_test_modified= data_modifier(x_test, features)
#first trying out the inbuilt Multinomial naive bayes classifier.
clf=MultinomialNB()
clf.fit(x_train_modified, y_train)
print(clf.score(x_test_modified, y_test)*100, "%")
#this function takes our xtrain and ytrain and combine them into a dictionary with features and the count of
words present
#in them and then returns a dictionary
def fit(x_train , y_train):
    d = {}
    #defining a dictionary
    for i in range(20):
        docs = x_train[y_train == i]
        #taking the classes one by one from x_train
        d[i] = {}
        #making a dictionary on the ith class to save the features and their total values
        d[i]['total'] = 0
```

```
        #this holds the value of the total words present in the class to be used in the probability function
        for j in range(len(features)):
            d[i][features[j]] = docs[:, j].sum()
            #how many times jth feature is coming corresponding to class i
            d[i]['total']+=d[i][features[j]]
            #stores the sum of all the values of ith key
    return d
# finding probabilty of each word in document for the current class
def probability(dictionary , x , current_class):
    prob_word = []
    #it will save all the probabs
    for i in range(len(x)):
        if x[i]!=0:
            #we dont want to consider words which are not present
            num = dictionary[current_class][features[i]]
            #finding numerator
            denom = dictionary[current_class]['total']
            #finding denominator
            prob = np.log((num + 1)/(denom + len(x)))
            #finding probability with laplace correction
            prob_word.append(prob)
            # appending in the list
    return sum(prob_word)
#finding the best class using the above function by comparing all the probabilities
def predictSinglePoint(dictionary, x):
    classes = dictionary.keys()
    # finding all classes
    bestp = -20
    #taking best probability negative
    bestc = -20
    #taking the best class negative
    firstrun = True
    #firstrun is created to update with the first probability no matter the case so negative probab can be removed
```

```
    for clas in classes:
        #iterating through each class
        prob_class = probability(dictionary, x, clas)
        #finding the probab of current class using the probabilty function as given above
        if(firstrun == True or bestp < prob_class):
            #updating the values in our variables to get the maximum probab class
            bestp = prob_class
            bestc = clas
        firstrun = False
        #making firstrun as false as we dont want to use it anymore
    return bestc
    #this function return the predicted classes by using the above functions
def predict(x_test, dictionary):
    y_pred = []
    #creating the empty list for predicted values
    for doc in x_test:
        #iterating through every doc and predicting values and appending to the predict list
        y_pred.append(predictSinglePoint(dictionary ,doc))
    return y_pred
#dictionary created through fit function contains classes and their features list
dictionary=fit(x_train_modified, y_train)
#predicted values from the predict function
y_predicted=predict(x_test_modified, dictionary)
#comparing our predcited values with the y_test
print(classification_report(y_true=y_test, y_pred=y_predicted))
```

EXPERIMENT 6: Exploratory Data Analysis for Classification using Pandas and Matplotlib

PROGRAM:

```
import pandas as pd
import matplotlib.pyplot as plt
# Load the Iris dataset
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data"
names = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width', 'class']
dataset = pd.read_csv(url, names=names)
# Print the first 5 rows of the dataset
print(dataset.head())
# Print the summary statistics of the dataset
print(dataset.describe())
# Print the class distribution
print(dataset['class'].value_counts())
# Box plots
dataset.plot(kind='box', subplots=True, layout=(2,2), sharex=False, sharey=False)
plt.show()
# Histograms
dataset.hist()
plt.show()
# Scatter plot matrix
pd.plotting.scatter_matrix(dataset)
plt.show()
# Create a scatter plot of the petal length and petal width features
plt.scatter(dataset['petal-length'], dataset['petal-width'])
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
plt.show()
```