**EXPERIMENT NO : 5**

**k-NN Algorithm**

**AIM : Program to implement k-NN classification using any standard dataset available in the public domain and find the accuracy of the algorithm**

from sklearn.datasets import load\_iris

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

from sklearn.neighbors import KNeighborsClassifier

from sklearn import metrics

iris = load\_iris()

x = iris.data

y = iris.target

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

c\_knn = KNeighborsClassifier(n\_neighbors=3)

c\_knn.fit(x\_train,y\_train)

y\_pred = c\_knn.predict(x\_test)

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

sample = [[2,2,2,2]]

pred = c\_knn.predict(sample)

pred\_v = [iris.target\_names[p] for p in pred]

print(pred\_v)

**EXPERIMENT NO : 6**

**Naive Bayes algorithm**

**AIM :Program to implement Naive Bayes Algorithm using any standard dataset available in the public domain and find the accuracy of the algorithm**

import pandas as pd

from sklearn.preprocessing import LabelEncoder

from sklearn.preprocessing import StandardScaler

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

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

from sklearn.naive\_bayes import GaussianNB

dataset = pd.read\_csv('Iris.csv')

print(dataset.describe())

print(dataset.head())

X = dataset.iloc[:, [1, 2, 3]].values

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

le = LabelEncoder()

X[:,0] = le.fit\_transform(X[:,0])

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size = 0.20, random\_state =0)

sc = StandardScaler()

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

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

from sklearn.naive\_bayes import GaussianNB

classifier = GaussianNB()

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

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

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

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

print(cm)

print(“Accuracy :”,ac)

**EXPERIMENT 7**

**LINEAR REGRESSION**

**AIM :Program to implement linear regression technique using any standard dataset available in the public domain and evaluate its performance.**

Predict canada's per capita income in year 2020. There is an exercise folder here on github at same level as this notebook, download that and you will find canada\_per\_capita\_income.csv file. Using this build a regression model and predict the per capita income fo canadian citizens in year 2020

"""

import numpy as np

import matplotlib.pyplot as plt

import pandas as pd

from sklearn.linear\_model import LinearRegression

# Importing the dataset

dataset = pd.read\_csv('salary\_data.csv')

X = dataset.iloc[:, :-1].values #get a copy of dataset exclude last column

y = dataset.iloc[:, 1].values #get array of dataset in column 1st

print(dataset.head())

# Splitting the dataset into the Training set and Test set

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

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=1/3, random\_state=0)

#

regressor = LinearRegression()

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

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

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

y\_pred

# Visualizing the Training set results

viz\_train = plt

viz\_train.scatter(X\_train, y\_train, color='red')

viz\_train.plot(X\_train, regressor.predict(X\_train), color='blue')

viz\_train.title('Salary VS Experience (Training set)')

viz\_train.xlabel('Year of Experience')

viz\_train.ylabel('Salary')

viz\_train.show()

# Visualizing the Test set results

viz\_test = plt

viz\_test.scatter(X\_test, y\_test, color='red')

viz\_test.plot(X\_train, regressor.predict(X\_train), color='blue')

viz\_test.title('Salary VS Experience (Test set)')

viz\_test.xlabel('Year of Experience')

viz\_test.ylabel('Salary')

viz\_test.show()

# Predicting the result of 5 Years Experience

y\_pred = regressor.predict([[5]])

print(y\_pred)

**EXPERIMENT NO : 8**

**AIM :Program to implement multiple linear regression technique using any standard dataset available in the public domain and evaluate its performance.**

import pandas as pd

import numpy as np

"""step 2:Import Dataset"""

data\_df=pd.read\_csv('CCCP - Sheet1.csv')

data\_df.head()

"""step 3: Define x and y """

x=data\_df.drop(['PE'],axis=1).values

y=data\_df['PE'].values

print(x)

print(y)

"""Step 4 :Split the dataset in training and test set"""

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=0)

"""step 5: Train the model on the training set"""

#Fitting Simple Linear Regression to the Training set

from sklearn.linear\_model import LinearRegression

regressor = LinearRegression()

regressor.fit(x\_train, y\_train)

"""step 6 : Predict the test results"""

y\_pred = regressor.predict(x\_test)

print(y\_pred)

#regressor.predict([[14.96,41.76,1024.07,73.17]])

"""Step 7 : Evaluate the model"""

from sklearn.metrics import r2\_score

r2\_score(y\_test,y\_pred)

"""Step 8 : Plot the results"""

import matplotlib.pyplot as plt

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

plt.scatter(y\_test,y\_pred)

plt.xlabel('Actual')

plt.ylabel('predicted')

plt.title('ACTUAL VS pREDICTED')

**EXPERIMENT NO :9**

**AIM : Program to implement text classification using Support vector machine.**

import numpy as np

from sklearn import datasets

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

from sklearn.feature\_extraction.text import TfidfVectorizer

from sklearn.svm import SVC

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

# Step 1: Load the dataset

# Using the 20 Newsgroups dataset from scikit-learn

newsgroups = datasets.fetch\_20newsgroups(subset='all')

# Step 2: Preprocess the text data

# Split the data into training and testing sets

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

# Step 3: Convert the text data into numerical features using TfidfVectorizer

vectorizer = TfidfVectorizer(stop\_words='english', max\_features=5000)

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

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

# Step 4: Initialize the SVM classifier

svm = SVC(kernel='linear', random\_state=42)

# Step 5: Train the model

svm.fit(X\_train\_tfidf, y\_train)

# Step 6: Predict on the test set

y\_pred = svm.predict(X\_test\_tfidf)

# Step 7: Evaluate the model

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

print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred, target\_names=newsgroups.target\_names))

**EXPERIMENT NO :10**

**DECISION TREE**

**AIM : Program to implement decision trees using any standard dataset available in the public domain and find the accuracy of the algorithm.**

from sklearn.datasets import load\_iris

from sklearn import metrics

from sklearn import tree

import matplotlib.pyplot as plt

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

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score

iris = load\_iris()

x,y = iris.data,iris.target

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

clf = DecisionTreeClassifier(criterion = 'entropy')

clf = clf.fit(x\_train,y\_train)

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

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

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

tree.plot\_tree(clf,

fontsize = 10,

filled = True,

rounded = True,

class\_names = iris.target\_names,

feature\_names = iris.feature\_names)

plt.show()

**EXPERIMENT NO : 11**

**AIM :Program to implement k-means clustering technique using any standard dataset available in the public domain**

import matplotlib.pyplot as plt

from sklearn import datasets

from sklearn.cluster import KMeans

#

# Load Sklearn IRIS dataset

#

iris = datasets.load\_iris()

X = iris.data

y = iris.target

#

# Do the scatter plot and see that clusters are evident

#

plt.scatter(X[:,1], X[:,3],

             color='white', marker='o', edgecolor='red', s=50)

plt.grid()

plt.tight\_layout()

plt.show()

# Create an instance of K-Means

#

kmc = KMeans(n\_clusters=3, init='random', n\_init=10, max\_iter=300,tol=1e-04, random\_state=0)

#

# Fit and make predictions

#

y\_kmc = kmc.fit\_predict(X)

#

# Create the K-means cluster plot

#

plt.scatter(X[y\_kmc == 0, 1], X[y\_kmc == 0, 3], s=50,

            c='lightgreen', marker='s', edgecolor='black', label='Cluster 1')

plt.scatter(X[y\_kmc == 1, 1], X[y\_kmc == 1, 3],

             s=50, c='orange', marker='o', edgecolor='black', label='Cluster 2')

plt.scatter(X[y\_kmc == 2, 1], X[y\_kmc == 2, 3], s=50,

            c='blue', marker='P', edgecolor='black', label='Cluster 3')

plt.scatter(kmc.cluster\_centers\_[:, 1], kmc.cluster\_centers\_[:, 3],

            s=250, marker='\*', c='red', edgecolor='black', label='Centroids')

plt.legend(scatterpoints=1)

plt.grid()

plt.tight\_layout()

plt.show()

**EXPERIMENT NO :12**

**AIM : Implement a program to scrap the web page of any popular website**

**(scrapes the titles of articles from a website)**

import requests

from bs4 import BeautifulSoup

# Step 1: Send a request to the webpage

url = 'https://www.geeksforgeeks.org/mac

response = requests.get(url)

# Step 2: Parse the HTML content

soup = BeautifulSoup(response.content, 'html.parser')

# Step 3: Find all the article titles (assuming they are in <h2> tags)

titles = soup.find\_all('h2')

# Step 4: Print the titles

for title in titles:

    print(title.get\_text())

**EXPERIMENT NO :11**

**AIM : Demonstrates basic NLP tasks-Tokenization, POS tagging, Chunking**

import nltk

# Download necessary NLTK resources

nltk.download('punkt')

nltk.download('averaged\_perceptron\_tagger')

# Sample text

text = """Machines can learn from past data and automatically improve their performance.

Given a dataset, ML can detect various patterns in the data.

For large organizations, branding is crucial, and targeting a relatable customer base becomes easier.

It is similar to data mining, as both deal with substantial amounts of data.."""

# Tokenize the text into words

tokens = nltk.word\_tokenize(text)

# Perform Part-of-Speech tagging

pos\_tags = nltk.pos\_tag(tokens)

# Define chunking grammar

grammar = r"""

    NP: {<DT>?<JJ>\*<NN>}   # Noun Phrase: Optional Determiner, Adjectives, and Noun

"""

# Create a chunk parser

chunk\_parser = nltk.RegexpParser(grammar)

# Parse the tagged words into chunks

chunked = chunk\_parser.parse(pos\_tags)

# Display the chunked structure

print("Chunked Output:")

print(chunked)