Machine Learning

Program1 – Web Scraping

```
import requests
from bs4 import BeautifulSoup
import pandas as pd
response = requests.get('https://books.toscrape.com/')
html_content = response.text
#print('Successfully fetched data from {response}')
#print(html_content)

if html_content:
    soup = BeautifulSoup(html_content,'html.parser')
    #print(soup)
    books = soup.find_all('article',class_='product_pod')
data=[]
for book in books:
    title = book.find('h3').find('a')['title']
    price = book.find('p',class_='price_color').text
    #data.append([title,price])
    #print(title,':',price)

df= pd.DataFrame(data,columns=['Title','Price'])
print(df)

df.csv('books_data.csv',index=False)
print("Data saved to books_data.csv")
```

Program2 – Data pre-processing

```
import pandas as pd
import numpy as np
#load the data contained file
dataset = pd.read_csv("student.csv")
print(dataset.describe)
print(dataset.info())

#printing the first 5 values used head
h1=dataset.head()
print(h1)

#printing the last  3 values using tail
t1=dataset.tail(3)
print(t1)

#creating independent dataset
x=dataset.iloc[:,:-1].values
print(x)

#creating independent dataset
y=dataset.iloc[:,-1].values
print(y)

#handlimg missing data
missingcount = dataset.isnull().sum()
print(missingcount)

data1 = dataset.dropna(axis=0,how="all")

data1 = dataset.dropna(axis=0,how="any")
```

```
print(data1.isnull().sum())

#calculate average values
m1 = dataset['CGPA'].mean
print(m1)

dataset['CGPA']=dataset['CGPA'].fillna(m1)
print(dataset['CGPA'])

print(dataset.isnull().sum())

#fill null values
dataset['Age']=dataset['Age'].fillna("0")
print(dataset.isnull().sum())
```

Program3 – Linear Regression

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
data=pd.read_csv("C:/Users/BMSIT/Downloads/program3.csv")
print(data)

x=data['YearsExperience']
#x.shape
x=np.array(data.iloc[:,0])
x=np.array(data.iloc[:][["YearsExperience"]])

y=data['Salary']
y=np.array(data.iloc[:,1])
```

```
from sklearn.model_selection import train_test_split
xtrain, xtest, ytrain, ytest=train_test_split(x,y,train_size=0.80,random_state=1)

from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(xtrain,ytrain)
ypred=model.predict(xtest)

from sklearn.metrics import r2_score
r2=r2_score(ytest, ypred)
plt.figure(figsize=(8,5))
plt.scatter(xtrain,ytrain,color='blue',s=100,label="Actual Point")
plt.scatter(xtrain, model.predict(xtrain), color='red',s=100,label="Predicted point")
plt.show()

plt.plot(xtrain,model.predict(xtrain),linestyle="dotted",color='orange',label="line of regression")
plt.show()

plt.figure(figsize=(8,5))
plt.scatter(xtrain,ytrain,color='blue',s=100,label="Actual Point")
plt.scatter(xtrain, model.predict(xtrain), color='red',s=100,label="Predicted point")
plt.plot(xtrain,model.predict(xtrain),linestyle="dotted",color='orange',label="line of regression")
plt.show()

plt.figure(figsize=(8,5))
plt.scatter(xtest, ytest, color='blue', s=100, label="Actual Point")
plt.scatter(xtest, model.predict(xtest), color='red', s=100, label=" Predicted point")
plt.plot(xtest, model.predict(xtest), linestyle='dotted', color='orange', label="Line of regression")
plt.show()

scores=[]
for i in range(5000):
```

```
    xtrain1, xtest1, ytrain1, ytest1 = train_test_split(x,y,train_size =0.80, random_state=i)
    model1=LinearRegression()
    model1.fit(xtrain1, ytrain1)
    ypred1=model1.predict(xtest1)
    scores.append(r2_score(ytest1, ypred1))

# command mode print(scores)
np.max(scores) # typre dosmode
np.argmax(scores) # dosmode

xtrain, xtest, ytrain, ytest = train_test_split(x,y,train_size=.80,random_state=4697)
from sklearn.linear_model import LinearRegression
model=LinearRegression()
model.fit(xtrain, ytrain)
ypred = model.predict(xtest)
from sklearn.metrics import r2_score
r2=r2_score(ytest, ypred)  #type r2

plt.figure(figsize=(8,5))
plt.scatter(xtrain, ytrain, color='blue', s=100, label="Actual Point ")
plt.scatter(xtrain, model.predict(xtrain), color='red', s=100, label ="Predicted point")
plt.show()

plt.plot(xtrain, model.predict(xtrain), linestyle='dotted', color='orange', label="Line of regression")
plt.show()

plt.figure(figsize=(8,5))
plt.scatter(xtest, ytest, color='blue', s=100, label="Actual Point")
plt.scatter(xtest, model.predict(xtest), color='red', s=100, label="Predicted point")
plt.plot(xtest, model.predict(xtest), linestyle='dotted', color='orange', label="Line of regression")
plt.show()
```

Program4 – Find S Algorithm

```
import pandas as pd
import numpy as np
data = pd.read_csv("program4_data.csv")
print(data)
concepts = np.array(data)[:,:-1]
print('The target column')
target = np.array(data)[:,-1]
print(target)

def train(con,tar):
    for i,val in enumerate(tar):
        if val == 'yes':
            specific_h = con[i].copy()
            break
    for i,val in enumerate((con)):
        if tar[i] == 'yes':
            for x in range((len(specific_h))):
                if val[x] != specific_h[x]:
                    specific_h[x] = '?'
                else :
                    pass
    return specific_h
print('The final output')
print(train(concepts,target))
```

Program5 – KNN Algorithm

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.metrics import accuracy_score
data = pd.read_csv("C:/Users/BMSIT/Iris.csv")
print(data)
data.head()
x=data.drop('species',axis=1)
y=data['species']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=0.2)
scaler = StandardScaler()
x_train_scaled = scaler.fit_transform(x_train)
x_test_scaled = scaler.transform(x_test)

#Initialize the KNN classifier with k=5
classifier = KNeighborsClassifier(n_neighbors=5)

#Train the KNN classifier
classifier.fit(x_train_scaled,y_train)

y_pred = classifier.predict(x_test_scaled)

accuracy = accuracy_score(y_test,y_pred)
print(f"Accuracy of the KNN model:{accuracy * 100: .2f}%")

print(f"Predicted labels: {y_pred}")
print(f"Actual labels:{y_test.values}")
```

Program 6 – SVM Algorithm

```
import numpy as np
from sklearn import datasets
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score,classification_report
```

```
data=datasets.load_iris()
print(data)

x=data.data
y=data.target

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=42)
svm_classifier=SVC(kernel='linear')

svm_classifier.fit(x_train,y_train)

#make prediction of the test set
y_pred=svm_classifier.predict(x_test)

#evaluate the model
accuracy=accuracy_score(y_test, y_pred)
print(f'Accuracy:{accuracy*100:.2f}%')

print('Classification report')
print(classification_report(y_test, y_pred))
```

Program 7 - Naive-Bayes Classifier

```
#Naive-Bayes Classifier
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score,classification_report
from sklearn.preprocessing import StandardScaler

data = pd.read_csv('program7.csv')
```

```
x = data.drop('Outcome',axis=1)
y = data['Outcome']
y = np.array(data.iloc[:,-1])

x_train,x_test,y_train,y_test = train_test_split(x,y,test_size=30,random_state=42)

scaler = StandardScaler()
x_train = scaler.fit_transform(x_train)
x_test = scaler.transform(x_test)

nb_classifier = GaussianNB()
nb_classifier.fit(x_train,y_train)

y_pred = nb_classifier.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy*100:.2f}% ")
#print(accuracy*100)

print("\n Classification Report")
print(classification_report(y_test, y_pred))
```

Progarm 8 – Kmeans Clustering

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
from sklearn.datasets import load_iris
from sklearn.preprocessing import StandardScaler

#step1 load dataset
iris = load_iris()
```

```
x=iris.data
y=iris.target

#step2 pre proceswsing-scalling the data
scaler  = StandardScaler()
x_scaled = scaler.fit_transform(x)

kmeans = KMeans(n_clusters=3)
kmeans.fit(x_scaled)

#step4 get  cluster center and labels
centroid = kmeans.cluster_centers_
labels=kmeans.labels_

#step5 visualize the results
plt.figure(figsize=(8,6))
plt.scatter(x_scaled[:,0],x_scaled[:,1],c=labels,cmap='viridis')

#plot the centroids
plt.scatter(centroid[:,0],centroid[:,1],c='red',s=300,marker='X',label="Centroids")

#Adding the labels
plt.title('kMeans clustering algorithm ')
plt.xlabel('Scaled sepal length')
plt.ylabel('Scaled sepal width')
plt.legend()
plt.show()
```