MSc(IT) Sem-III

Machine Learning Slips Solutions

Execute the following programs in **Jupyter.**

1. Write a python program to splitting the dataset into training and testing set.

(

// numpy for mathematical operations

// pandas to use .csv or .xl file, or to import column from dataset

// Scikit-Learn, also known as sklearn is a python library to implement machine learning models and statistical modelling. Through scikit-learn, we can implement various machine learning models for regression, classification, clustering, and statistical tools for analyzing these models.

// The encode() function in Python is responsible for returning the encoded form of any given string

// The fit\_transform () method is used to fit the data into a model and transform it into a form that is more suitable for the model in a single step.

//: means all row, : -1 means excluding last column

)

**Solution:**

import numpy as np

import pandas as pd

dataset = pd.read\_csv("play\_tennis.csv")

dataset

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

dataset['outlook'] = le.fit\_transform(dataset.outlook)

dataset['temp'] = le.fit\_transform(dataset.temp)

dataset['humidity'] = le.fit\_transform(dataset.humidity)

dataset['wind'] = le.fit\_transform(dataset.wind)

dataset['play'] = le.fit\_transform(dataset.play)

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

print(x)

y=dataset.iloc[:,4].values

print(y)

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)

print(x\_train)

print(x\_test)

1. Consider following dataset

weather=['Sunny','Sunny','Overcast','Rainy','Rainy','Rainy','Overcast','S  
 unny','Sunny','Rainy','Sunny','Overcast','Overcast','Rainy']  
 temp=['Hot','Hot','Hot','Mild','Cool','Cool','Cool','Mild','Cool','Mild','Mi  
 ld','Mild','Hot','Mild']  
 play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Y  
 es','No'].

Use Naïve Bayes algorithm to predict [ 0:Overcast, 2:Mild]   
 tuple belongs to which class whether to play the sports or not.

**Solution:**

weather=['sunny','sunny','overcast','rainy','rainy','rainy','overcast','sunny','sunny','rainy','sunny','overcast','overcast','rainy']

temp=['hot','hot','hot','mild','cool','cool','cool','mild','cool','mild','mild','mild','hot','mild']

play=['No','No','Yes','Yes','Yes','No','Yes','No','Yes','Yes','Yes','Yes','Yes','No']

from sklearn import preprocessing

le = preprocessing.LabelEncoder()

wheather\_encoded = le.fit\_transform(weather)

print(wheather\_encoded)

temp\_encoded = le.fit\_transform(temp)

label = le.fit\_transform(play)

print("Temp:",temp\_encoded)

print("Play:",label)

features = list(zip(wheather\_encoded,temp\_encoded))

print(features)

from sklearn.naive\_bayes import GaussianNB

model = GaussianNB()

model.fit(features,label)

predicted = model.predict([[0,2]])

print("Predicted Value:",predicted)

1. Write a python program the Categorical values in numeric format for a given dataset.

Solution :

import pandas as pd

from sklearn.preprocessing import LabelEncoder

# Sample dataset

data = {

'Category': ['A', 'B', 'A', 'C', 'B', 'A']

}

# Creating a DataFrame

df = pd.DataFrame(data)

# Initialize the LabelEncoder

label\_encoder = LabelEncoder()

# Apply label encoding to the 'Category' column

df ['Category\_encoded'] = label\_encoder.fit\_transform(df ['Category'])

print(df)

1. Write a python program to find all null values in a given data set and remove them.  
   ((Use diabetes\_null\_values.csv)

Solution :

import pandas as pd

# Load the dataset

df =pd.read\_csv('diabetes\_null\_values.csv')

print(df)

# Display the number of null values in each column

null\_counts = df.isnull().sum()

print("Null value counts:\n", null\_counts)

# Remove rows with any null values

df\_cleaned = df.dropna()

# Display the cleaned dataset

print("\nCleaned dataset:\n", df\_cleaned)

1. **write a python program to perform k-Nearest Neighbour classification using in-built iris dataset.**

# Importing necessary libraries

from sklearn.neighbors import KNeighborsClassifier

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features

y = iris.target # Target variable

# Splitting the dataset into training and testing sets (if not split already)

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

# Creating KNN classifier

knn\_classifier = KNeighborsClassifier(n\_neighbors=5) # You can adjust the number of neighbors as needed

# Training the classifier

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

# Making predictions on the testing set

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

# Calculating the accuracy

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

print("Accuracy:", accuracy)

1. write a python program to perform support vector machine classification using in-built iris dataset.

# Importing necessary libraries

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

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features

y = iris.target # Target variable

# Splitting 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)

# Creating SVM classifier

svm\_classifier = SVC(kernel='linear') # Linear kernel

# Other options for the kernel parameter are 'poly', 'rbf', 'sigmoid', etc.

# Training the classifier

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

# Making predictions on the testing set

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

# Calculating the accuracy

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

print("Accuracy:", accuracy)

1. write a python program to perform Decision Tree classification using in-built iris dataset.

# Importing necessary libraries

from sklearn.tree import DecisionTreeClassifier

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features

y = iris.target # Target variable

# Splitting the dataset into training and testing sets (if not split already)

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

# Creating Decision Tree classifier

decision\_tree\_classifier = DecisionTreeClassifier()

# Training the classifier

decision\_tree\_classifier.fit(X\_train, y\_train)

# Making predictions on the testing set

y\_pred = decision\_tree\_classifier.predict(X\_test)

# Calculating the accuracy

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

print("Accuracy:", accuracy)

1. write a python program to perform Naive Bayes classification using in-built iris dataset,.

# Importing necessary libraries

from sklearn.naive\_bayes import GaussianNB

# Load the Iris dataset

iris = datasets.load\_iris()

X = iris.data # Features

y = iris.target # Target variable

# Splitting the dataset into training and testing sets (if not split already)

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

# Creating Naive Bayes classifier

naive\_bayes\_classifier = GaussianNB()

# Training the classifier

naive\_bayes\_classifier.fit(X\_train, y\_train)

# Making predictions on the testing set

y\_pred = naive\_bayes\_classifier.predict(X\_test)

# Calculating the accuracy

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

print("Accuracy:", accuracy)

# Write a Program to implement simple linear regression using Sample Data - Salary and Years of Experience

# from sklearn.linear\_model import LinearRegression

import numpy as np

# Sample data - Salary and Years of Experience

x = np.array([1, 2, 3, 4, 5]).reshape(-1, 1) # Years of experience

y = np.array([30000, 35000, 40000, 45000, 50000]) # Salary

# Create and fit the model

model = LinearRegression().fit(x, y)

# Get the slope and intercept

slope = model.coef\_[0]

intercept = model.intercept\_

# Predict salary for new value of years of experience

new\_x = np.array([[6]])

predicted\_salary = model.predict(new\_x)

print("Slope (m):", slope)

print("Intercept (b):", intercept)

print("Predicted salary for {} years of experience: ${}".format(new\_x[0][0], predicted\_salary[0]))

**Output:**

Slope (m): 5000.000000000001

Intercept (b): 24999.999999999996

Predicted salary for 6 years of experience: $55000.0

# Write a Program to implement multiple linear regression using Sample Data - Salary , Years of Experience, Age and Performance of employee.

from sklearn.linear\_model import LinearRegression

import numpy as np

# Sample data - Salary, Years of Experience, Performance

x = np.array([[1, 25, 8], [2, 30, 7], [3, 35, 9], [4, 40, 6], [5, 45, 8]]) # Years of experience, Age, Performance

y = np.array([30000, 35000, 40000, 45000, 50000]) # Salary

# Create and fit the model

model = LinearRegression().fit(x, y)

# Get the coefficients and intercept

coefficients = model.coef\_

intercept = model.intercept\_

print("Coefficients:", coefficients)

print("Intercept:", intercept)

# Create array for new employee data

new\_employee\_data = np.array([[6, 45, 8]])

# Predict salary for new employee

predicted\_salary = model.predict(new\_employee\_data)

print("Predicted salary for the new employee: $", predicted\_salary[0])

**Output:**

Coefficients: [1.92307692e+02 9.61538462e+02 3.15309372e-14]

Intercept: 5769.230769230751

Predicted salary for the new employee: $ 29999.999999999996

# Write a Program to implement non- linear regression using Sample Data - Score and Study hours of students.

import numpy as np

import matplotlib.pyplot as plt

from sklearn.preprocessing import PolynomialFeatures

from sklearn.linear\_model import LinearRegression

# Student dataset

Student = {

'Hours': [2.5, 5.1, 3.2, 8.5, 3.5, 1.5, 9.2, 5.5, 8.3, 2.7, 7.7, 5.9, 4.5, 3.3, 1.1, 8.9, 2.5, 1.9, 6.1, 7.4, 2.7, 4.8],

'Scores': [21, 47, 27, 75, 30, 20, 88, 60, 81, 25, 85, 62, 41, 42, 17, 95, 30, 24, 67, 69, 30, 54]

}

# Convert to numpy arrays

hours = np.array(Student['Hours']).reshape(-1, 1)

scores = np.array(Student['Scores'])

# Define polynomial degree

degree = 2

# Create polynomial features

poly = PolynomialFeatures(degree=degree)

hours\_poly = poly.fit\_transform(hours)

# Create and fit the model

model = LinearRegression().fit(hours\_poly, scores)

# Predict score for 9 hours of study

hours\_to\_predict = np.array([[9]])

hours\_to\_predict\_poly = poly.transform(hours\_to\_predict)

predicted\_score = model.predict(hours\_to\_predict\_poly)

print("Predicted score for 9 hours of study using polynomial regression (degree={}): {}".format(degree, predicted\_score[0]))

**Output:**

Predicted score for 9 hours of study using polynomial regression (degree=2): 98.95199850969358

12.

