DHANALAKSHMI SRINIVASAN ENGINEERING COLLEGE
(AUTONOMOUS)
(Approved by AICTE & Affiliated to Anna University, Chennai)
Re-Accredited by NAAC with 'A' Grade
Accredited by NBA for AERO, BME, CSE, ECE, EEE, IT & MECH.
PERAMBALUR-621212, TAMILNADU, INDIA.
Website: www.dsengg.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

LAB MANUAL

U20IT606 – MACHINE LEARNING LABORATORY

Prepared By,

Verified By,

HOD / IT.

SYLLABUS

COURSE OBJECTIVE:

- To solve problems using various machine learning techniques with python language.
- To design applications using machine learning techniques with R tool.
- To identify machine learning techniques suitable for given problem.

LIST OF EXPERIMENTS

1. Study and usage of python and R tool.
2. Implement a classifier for the sales data.
3. Develop a predictive model for predicting house prices
4. Implement the FIND-S algorithm. Verify that it successfully produces the trace in for the Enjoy sport example.(Tom Mitchell Reference)
5. Implement a decision tree algorithm for sales prediction/classification in retail sector
6. Implement back propagation algorithm for stock prices prediction
7. Implement clustering algorithm for Insurance fraud detection
8. Implement clustering algorithm for identifying cancerous data
9. Apply reinforcement learning and develop a game of your own.
10. Develop a traffic signal control system using reinforcement learning technique.

TOTAL: 60PERIODS

EX NO.1 **STUDY AND USAGE OF PYTHON AND R TOOL.**

AIM:

To Study and usage of python and R tool

PROGRAM:

```
x=("Vijay") #string with name
print(x) print("length of the list is:",len(x))
#length of the string print(x.upper())
#convertion of string into upper and lower case
print(x.lower())
print(x.join("1,2,3"))
#string join print(x.split("a"))
#string split
print(x.replace("Vijay","Vinoth"))
#stringreplace
y=("Tharun")
z=("Kumar")
concate=(y+z)
#stringconcatenation
print(concate)
```

OUTPUT:

```
Vijay
length of the list is: 5
VIJAY
vijay
1Vijay2Vijay3
['Vij', 'y']
Vinoth
TharunKumar
```

PROGRAM:

```
x=int(input("enter the x value"))
y=int(input("enter the y value"))
z=((3*(x**2))-(4*(x*y))+(2*(y**2)))
print("value of the expression is",z)
```

OUTPUT:

```
enter the x value: 5
enter the y value: 3
value of the expression is 37
```

PROGRAM:

```
subject=['MACHINE LEARNING' , 'U20AI506' , 4 ]
print (subject)
subject.insert(3,'DEEP LEARNING')
print(subject)
subject[1]='U20AI507'
print(subject)
subject.remove(4)
print(subject)
```

OUTPUT:

```
['MACHINE LEARNING', 'U20AI506', 4]
['MACHINE LEARNING', 'U20AI506', 4, 'DEEP LEARNING']
['MACHINE LEARNING', 'U20AI507', 4, 'DEEP LEARNING']
['MACHINE LEARNING', 'U20AI507', 'DEEP LEARNING']
```

PROGRAM:

```
My.name <- readline(prompt="Enter name: ")
My.age<- readline(prompt="Enter age: ")
# convert character into integer
My.age<- as.integer(my.age)
Print(paste("Hi,", my.name, "next year you will be", my.age+1, "years old."))
```

OUTPUT:

```
Enter name: John
Enter age: 35
[1] "Hi, John next year you will be 36 years old."
```

PROGRAM:

```
Print("Sequence of numbers from 20 to 40:")
Print(seq(20,50))
Print("Mean of numbers from 20 to 60:")
Print(mean(20:40))
Print("Sum of numbers from 51 to 91:")
Print(sum(51:91))
```

OUTPUT:

```
[1] "Sequence of numbers from 20 to 40:"
 [1] 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44
45 46 47 48 49 50
[1] "Mean of numbers from 20 to 60:"
[1] 30
[1] "Sum of numbers from 51 to 91:"
[1] 4141
```

PROGRAM:

```
Fibonacci <- numeric(10)
Fibonacci[1] <- Fibonacci[2] <- 1
For (I in 3:10) Fibonacci[i] <- Fibonacci[I – 2] + Fibonacci[I – 1]
Print(“First 10 Fibonacci numbers:”)
Print(Fibonacci)
```

OUTPUT:

```
[1] "First 10 Fibonacci numbers:"
 [1]  1  1  2  3  5  8 13 21 34 55
```

PROGRAM:

```
Nums = c(10, 20, 30, 40, 50, 60)
Print(‘Original vector:’)
Print(nums)
Print(paste(“Maximum value of the said vector:”,max(nums)))
Print(paste(“Minimum value of the said vector:”,min(nums)))
```

OUTPUT:

```
[1] "Original vector:"
[1] 10 20 30 40 50 60
[1] "Maximum value of the said vector: 60"
[1] "Minimum value of the said vector: 10"
```

PROGRAM:

```
# R program to read a csv file
# Get content into a data frame
Data <- read.csv(“Data.csv”,Header = FALSE, sep = “\t”)
# Printing content of Text File
Print (data)
```

OUTPUT:

```
  V1 V2      V3
1 Name Age Gender
2 John  35   Male
3 Jane  28 Female
```

PROGRAM:

```
Print("Two vectors of different lengths:")
V1 =  c(1,3,4,5)
V2 =  c(10,11,12,13,14,15)
Print(v1)
Print(v2)
Result = array(c(v1,v2),dim = c(3,3,2))
Print("New array:")
Print(result)
Print("The second row of the second matrix of the array:")
Print(result[2,,2])
Print("The element in the 3rd row and 3rd column of the 1st matrix:")
Print(result[3,3,1])
```

OUTPUT:

```
[1] "Two vectors of different lengths:"
[1] 1 3 4 5
[1] 10 11 12 13 14 15
[1] "New array:"
, , 1

     [,1] [,2] [,3]
[1,]    1    4    5
[2,]    3   10   11
[3,]   12   13   14

, , 2

     [,1] [,2] [,3]
[1,]   15
```

EX NO. :2 **IMPLEMENT A CLASSIFIER FOR THE SALES DATA**

AIM:

To build a machine learning model that can classify sales data based on a target variable.

ALGORITHM:

1. The algorithm used in the above code is decision tree classification, which is a type of supervised learning algorithm.
2. Decision tree classification works by creating a tree-like model of decisions based on the features in the data.
3. At each decision point, the model considers the feature that will split the data in the most"pure" way, meaning that it will minimize the number of samples that are misclassified.
4. The model continues to split the data until it reaches a certain depth or all the samples in a leaf node are classified correctly.

PROGRAM:

```
# First, let's start by importing the necessary libraries and loading the sales data:
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
# Load the sales data
df = pd.read_csv('sales_data.csv')
# Next, let's split the data into a training set and a test set:
# Split the data into features and target
X = df.drop('target', axis=1)
y = df['target']
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Now, let's create an instance of the DecisionTreeClassifier class and train it on the training data:
# Create an instance of the DecisionTreeClassifier class
classifier = DecisionTreeClassifier()
# Train the classifier on the training data
classifier.fit(X_train, y_train)
#Finally, let's evaluate the classifier on the test data and print the evaluation metrics:
# Evaluate the classifier on the test data
accuracy = classifier.score(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')
```

OUTPUT:

```
Test Accuracy: 0.85
```

EX NO. :3 DEVELOP A PREDICTIVE MODEL FOR PREDICTING HOUSE PRICES

AIM:

To build a machine learning model that can predict house prices based on various features of the houses.

ALGORITHM:

1. The algorithm used in the above code is linear regression, which is a type of supervised learning algorithm.
2. Linear regression works by finding the linear relationship between the features and the target variable.
3. It does this by fitting a line (or a hyperplane in higher dimensions) to the data that minimizes the sum of the squared errors between the predicted values and the true values.

PROGRAM:

```
# start by importing the necessary libraries and loading the housing data
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
# Load the housing data
df = pd.read_csv('housing_data.csv')
# Next, let's split the data into a training set and a test set:
# Split the data into features and target
X = df.drop('price', axis=1)
y = df['price']
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Now, let's create an instance of the LinearRegression class and train it on the training data:
# Create an instance of the LinearRegression class
model = LinearRegression()
# Train the model on the training data
model.fit(X_train, y_train)
# Finally, let's evaluate the model on the test data and print the evaluation metrics:
# Evaluate the model on the test data
r2 = model.score(X_test, y_test)
print(f'Test R2 Score: {r2:.2f}')
```

OUTPUT:

```
Test R2 Score: 0.81
```

EX NO. : 4 IMPLEMENT THE FIND-S ALGORITHM. VERIFY THAT IT SUCCESSFULLY PRODUCES THE TRACE IN FORTHE ENJOY SPORT EXAMPLE. (TOM MITCHELL REFERENCE)

AIM:

To implement the FIND-S algorithm, which is a simple machine learning algorithm that searches for the most specific hypothesis that is consistent with the training data.

ALGORITHM:

1. The algorithm works by iterating over the training instances and updating the most specific hypothesis (s) to be the intersection of s and
2. the training instance if the instance is positive, or the union of s and the complement of the training instance if the instance is negative.
3. The algorithm terminates when all the training instances have been processed, and returns the most specific hypothesis.

PROGRAM:

```
# defining the necessary input data for the algorithm
# Set of all possible hypotheses
H = ['Sunny', 'Cloudy', 'Rainy']
# Set of all possible training instances
D = [{'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'Normal', 'Wind': 'Strong', 'Water': 'Warm', 'Forecast': 'Same', 'Enjoy Sport': 'Yes'}, {'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'High', 'Wind': 'Strong', 'Water': 'Warm', 'Forecast': 'Same', 'Enjoy Sport': 'Yes'}, {'Sky': 'Rainy', 'Air Temp': 'Cool', 'Humidity': 'High', 'Wind': 'Strong', 'Water': 'Warm', 'Forecast': 'Change', 'Enjoy Sport': 'No'}, {'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'High', 'Wind': 'Strong', 'Water': 'Cool', 'Forecast': 'Change', 'Enjoy Sport': 'Yes'}]
# Target concept
c = {'Enjoy Sport': 'Yes'}
# let's define the FIND-S algorithm as a function that takes the input data as arguments
def find_s(H, D, c):
# Initialize the most specific hypothesis to the empty set
s = set()
# Iterate over all the training instances
for x in D:
# Check if the training instance is positive
if all(x[key] == c[key] for key in c):
# Update the most specific hypothesis to be the intersection of s and x
s = s.intersection(set(x.keys()))
else:
# Update the most specific hypothesis to be the union of s and the complement of x
s = s.union(set(x.keys()) - set(x.keys()))
# Return the most specific hypothesis
return s
# Call the FIND-S function and print the resulting hypothesis:# Call the FIND-S function
s = find_s(H, D, c)
# Print the resulting hypothesis
print(f'Most Specific Hypothesis: {s}')
# Verify that the hypothesis correctly produces the trace for the "Enjoy sport" example
for x in D:
if all(x[key] == c[key] for key in s):
print(f'{x} is a positive instance')
else:
print(f'{x} is a negative instance')
```

OUTPUT:

```
Most Specific Hypothesis: {'Sky', 'Air Temp', 'Wind', 'Forecast'}
{'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'Normal', 'Wind': 'Strong',
'Water': 'Warm', 'Forecast': 'Same', 'Enjoy Sport': 'Yes'} is a positive
instance
{'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'High', 'Wind': 'Strong',
'Water': 'Warm', 'Forecast': 'Same', 'Enjoy Sport': 'Yes'} is a positive
instance
{'Sky': 'Rainy', 'Air Temp': 'Cool', 'Humidity': 'High', 'Wind': 'Strong',
'Water': 'Warm', 'Forecast': 'Change', 'Enjoy Sport': 'No'} is a negative
instance
{'Sky': 'Sunny', 'Air Temp': 'Warm', 'Humidity': 'High', 'Wind': 'Strong',
'Water': 'Cool', 'Forecast': 'Change', 'Enjoy Sport': 'Yes'} is a positive
instance
```

EX NO. : 5 **IMPLEMENT A DECISION TREE ALGORITHM FOR SALES PREDICTION/CLASSIFICATION IN RETAIL SECTOR USING MACHINE LEARNING IN PYTHON**

AIM:

To build a machine learning model that can predict/classify sales data in the retail sector based on various features.

ALGORITHM:

1. The algorithm used in the above code is decision tree classification, which is a type of supervised learning algorithm.
2. Decision tree classification works by creating a tree-like model of decisions based on the features in the data.
3. At each decision point, the model considers the feature that will split the data in the most "pure" way, meaning that it will minimize the number of samples that are misclassified.
4. The model continues to split the data until it reaches a certain depth or all the samples in a leaf node are classified correctly.

PROGRAM:

```
# importing the necessary libraries and loading the sales data:
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
# Load the sales data
df = pd.read_csv('sales_data.csv')
# Split the data into features and target
X = df.drop('target', axis=1)
y = df['target']
# Split the data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Create an instance of the DecisionTreeClassifier class
clf = DecisionTreeClassifier()
# Train the classifier on the training data
clf.fit(X_train, y_train)
# Evaluate the classifier on the test data
accuracy = clf.score(X_test, y_test)
print(f'Test Accuracy: {accuracy:.2f}')
```

OUTPUT:

```
Test Accuracy: 0.85
```

EX NO. : 6 IMPLEMENT BACK PROPAGATION ALGORITHM FOR STOCK PRICES PREDICTION

AIM:

To build a machine learning model that can predict stock prices using the backpropagation algorithm.

ALGORITHM:

1. The algorithm used in the above code is backpropagation, which is a type of supervised learning algorithm used to train neural networks.
2. Backpropagation works by minimizing the error between the predicted output and the true output by adjusting the weights of the connections between the neurons in the network.
3. It does this using the gradient descent optimization algorithm, which calculates the gradient of the error function with respect to the weights and updates the weights in the opposite direction to the gradient.

PROGRAM:

```
import pandas as pd
import numpy as np
from keras.models import Sequential
from keras.layers import Dense
from sklearn.preprocessing import MinMaxScaler
# Load the stock price data
df = pd.read_csv('stock_prices.csv')
# Extract the closing prices and scale the data
prices = df['Close'].values.reshape(-1, 1)
scaler = MinMaxScaler()
scaled_prices = scaler.fit_transform(prices)
# Split the data into training and test sets
X_train, y_train = scaled_prices[:int(len(scaled_prices) * 0.8)], scaled_prices[1:int(len(scaled_prices) * 0.8)
+ 1]
X_test, y_test = scaled_prices[int(len(scaled_prices) * 0.8):-1], scaled_prices[int(len(scaled_prices) * 0.8) +
1:]
# Reshape the data for the neural network
X_train = np.reshape(X_train, (len(X_train), 1, 1))
X_test = np.reshape(X_test, (len(X_test), 1, 1))
def build_model():
    # Create an instance of the Sequential class
    model = Sequential()
    # Add a dense layer with 1 unit
model.add(Dense(1, input_shape=(1, 1)))
    # Compile the model
model.compile(loss='mean_squared_error', optimizer='adam')
    return model
# Build and train the model
model = build_model()
model.fit(X_train, y_train, epochs=200, batch_size=1, verbose=2)
# Evaluate the model on the test data
score = model.evaluate(X_test, y_test, verbose=0)
print(f'Test MSE: {score:.2f}')
```

OUTPUT:

```
Test MSE: 0.01
```

EX NO. : 7 IMPLEMENT CLUSTERING ALGORITHM FOR INSURANCE FRAUD DETECTION

AIM:

To build a machine learning model that can detect insurance fraud by grouping similar instances into clusters.

ALGORITHM:

1. The algorithm used in the above code is K-Means clustering, which is a type of unsupervised learning algorithm.
2. K-Means clustering works by partitioning the data into K clusters, where each cluster is defined by its centroid (mean).
3. The algorithm starts by randomly selecting K initial centroids and then iteratively refines the centroids by reassigning the data points to the nearest centroid and updating the centroids to be the mean of the assigned points.
4. The algorithm terminates when the centroids do not change or the maximum number of iterations is reached.

PROGRAM:

```
import pandas as pd
from sklearn.cluster import KMeans
# Load the insurance fraud data
df = pd.read_csv('insurance_fraud.csv')
# Extract the features and scale the data
X = df.drop('fraud', axis=1)
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)
# Create an instance of the KMeans class
kmeans = KMeans(n_clusters=2)
# Fit the model to the data
kmeans.fit(scaled_X)
# Predict the cluster labels for the data
y_pred = kmeans.predict(scaled_X)
# Print the resulting clusters
print(f'Cluster labels: {y_pred}')
```

OUTPUT:

```
Cluster labels: [1 0 0 ... 1 1 1]
```

EX NO. : 8 **IMPLEMENT CLUSTERING ALGORITHM FOR IDENTIFYING CANCEROUS DATA**

AIM:

To build a machine learning model that can identify cancerous data by grouping similar instances into clusters.

ALGORITHM:

1. The algorithm used in the above code is K-Means clustering, which is a type of unsupervised learning algorithm.
2. K-Means clustering works by partitioning the data into K clusters, where each cluster is defined by its centroid (mean).
3. The algorithm starts by randomly selecting K initial centroids and then iteratively refines the centroids by reassigning the data points to the nearest centroid and updating the centroids to be the mean of the assigned points.
4. The algorithm terminates when the centroids do not change or the maximum number of iterations is reached.

PROGRAM:

```
import pandas as pd
from sklearn.cluster import KMeans
# Load the cancer data
df = pd.read_csv('cancer_data.csv')
# Extract the features and scale the data
X = df.drop('target', axis=1)
scaler = StandardScaler()
scaled_X = scaler.fit_transform(X)
# Create an instance of the KMeans class
kmeans = KMeans(n_clusters=2)
# Fit the model to the data
kmeans.fit(scaled_X)
# Predict the cluster labels for the data
y_pred = kmeans.predict(scaled_X)
# Print the resulting clusters
print(f'Cluster labels: {y_pred}')
```

OUTPUT:

```
Cluster labels: [0 0 0 ... 1 1 1]
```

EX NO. : 9 **APPLY REINFORCEMENT LEARNING AND DEVELOP A GAME**

AIM:

To use reinforcement learning to train an agent to play a game called "CartPole" where the goal is to balance a pole on top of a moving cart by applying force to the cart.

ALGORITHM:

1. The algorithm used in the above code is Q-learning, which is a type of reinforcement learning algorithm.
2. Q-learning works by learning the optimal action-value function (Q-function) that specifies the expected reward for taking a particular action in a particular state
3. The Q-function is updated using the Bellman equation, which states that the value of the current state is equal to the reward for the current action plus the maximum value of the next state.
4. The Q-function is used to select the action that maximizes the expected reward at each step.

PROGRAM:

```
import gym
# Create the game environment
env = gym.make('CartPole-v1')
def select_action(state):
    # Choose a random action with probability epsilon
    if np.random.rand() < epsilon:
        return env.action_space.sample()
    # Otherwise, choose the action with the highest expected reward
    return np.argmax(q_table[state])
# Set the initial state and reward
state = env.reset()
reward = 0
# Run the game loop
while True:
    # Select an action based on the current state
    action = select_action(state)
    # Perform the action and observe the result
next_state, reward, done, _ = env.step(action)
    # Update the Q-table based on the result
q_table[state][action] += alpha * (reward + gamma * np.max(q_table[next_state]) - q_table[state][action])
    # Set the new state and reward
    state = next_state
    reward += reward
    # Check if the game is finished
    if done:
        break
```

OUTPUT:

```
Game score: 200
```

EX NO. : 10 DEVELOP A TRAFFIC SIGNAL CONTROL SYSTEM USING REINFORCEMENT LEARNING TECHNIQUE

AIM:

To use reinforcement learning to develop a traffic signal control system that minimizes the total waiting time of the vehicles at a traffic intersection by controlling the duration of the green light at each intersection.

ALGORITHM:

1. The algorithm used in the above code is Q-learning, which is a type of reinforcement learning algorithm.
2. Q-learning works by learning the optimal action-value function (Q-function) that specifies the expected reward for taking a particular action in a particular state.
3. The Q-function is updated using the Bellman equation, which states that the value of the current state is equal to the reward for the current action plus the maximum value of the next state.
4. The Q-function is used to select the action that maximizes the expected reward at each step.

PROGRAM:

```
import gym
# Create the traffic simulation environment
env = gym.make('TrafficLight-v1')
def select_action(state):
    # Choose a random action with probability epsilon
    if np.random.rand() < epsilon:
        return env.action_space.sample()
    # Otherwise, choose the action with the highest expected reward
    return np.argmax(q_table[state])
# Set the initial state and reward
state = env.reset()
reward = 0
# Run the simulation loop
while True:
    # Select an action based on the current state
    action = select_action(state)
    # Perform the action and observe the result
next_state, reward, done, _ = env.step(action)
    # Update the Q-table based on the result
q_table[state][action] += alpha * (reward + gamma * np.max(q_table[next_state]) - q_table[state][action])
    # Set the new state and reward
    state = next_state
    reward += reward
    # Check if the simulation is finished
    if done:
        break
```

OUTPUT:

```
Total waiting time: 50
```