INDEX

1) Implement FIND-S algorithm for finding the most specific hypothesis.
a. Read the Iris training data from a .CSV file.
b. Output best hypothesis that better separate the data.
Objectives
Enable the students:

- • To understand how to learn specific boundaries from the given hypothesis space.
- • To analyze how to learn concepts from positive targets.
- • To Develop a classification model using Find-s algorithm on a training set.
- • To evaluate the model using test set.

Description:
The a. csv can be any dataset that that reflects classification problem. Classification task predict the future over a learned model. The given data is divided in to training and test set folds . Here, a model is learned using **FIND-S** algorithm over train data. The performance of the learned model is measured on test data. For classification task the data format is like (set of concepts, Target) i.e each instance is associated with group information (class label).
Procedure:
Step1: Load Data set
Step2: Initialize Specific Hypothesis with set of Ø.
Step3: For each training example
Step4: If example is positive example
if attribute_value == hypothesis_value:
Do nothing
else:
replace attribute value with '?' (Basically generalizing it)

Source Code:

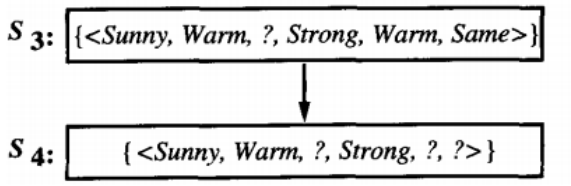
```
import numpy as np
import pandas as pd
data=pd.read_csv("enjoy.csv")
print(data)
train=np.array(data)[:,:-1]
print(train)
target=np.array(data)[:,-1]
print(target)
h=train[0]print(h)
for i,val in enumerate(target):
        if val=='yes':
                temp_h=train[i]
                for j in range(len(h)):
                        if(h[j])!=temp_h[j]:
                                h[j]="?"
                        else:
                                pass

print(h)
```

INPUT:
Data set, enjoysport.csv, which reflects a classification task of whether a student enjoyed the sport (yes) or not (no). Here the concepts are (sky, air_temp, humidity, wind, water, forecast) the target is enjoy_sport.

Sky	air_temp	humidity	wind	Water	forecast	enjoy_sport
Sunny	Warm	normal	strong	Warm	Same	Yes
Sunny	Warm	high	strong	Warm	Same	Yes
Rainy	Cold	high	strong	Warm	Change	No
Sunny	Warm	high	strong	Cool	Same	Yes

OUTPUT:
Specific Hypothesis



2) **Implement linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs**

a. Read the training data from a .CSV file.

b. Output the RMSE

Objectives

Enable the students:

- • To understand the for regression principle.
- • To analyze the computation of regression coefficient.
- • To implement a program for regress model on a training set using coefficients.
- • To evaluate the model using test set.

Description:

This is an implementation for Linear regression model , where the regression coefficients that are computed using Least Squares method. A the end the fitted regression model on the forecast datasets used to predict the real value of the test data.

Procedure:

1. Load the Dataset
2. Calculate Mean and Variance.
3. Calculate Covariance.
4. Estimate Coefficients.
5. Make Predictions.
6. Predict for New Independent variables

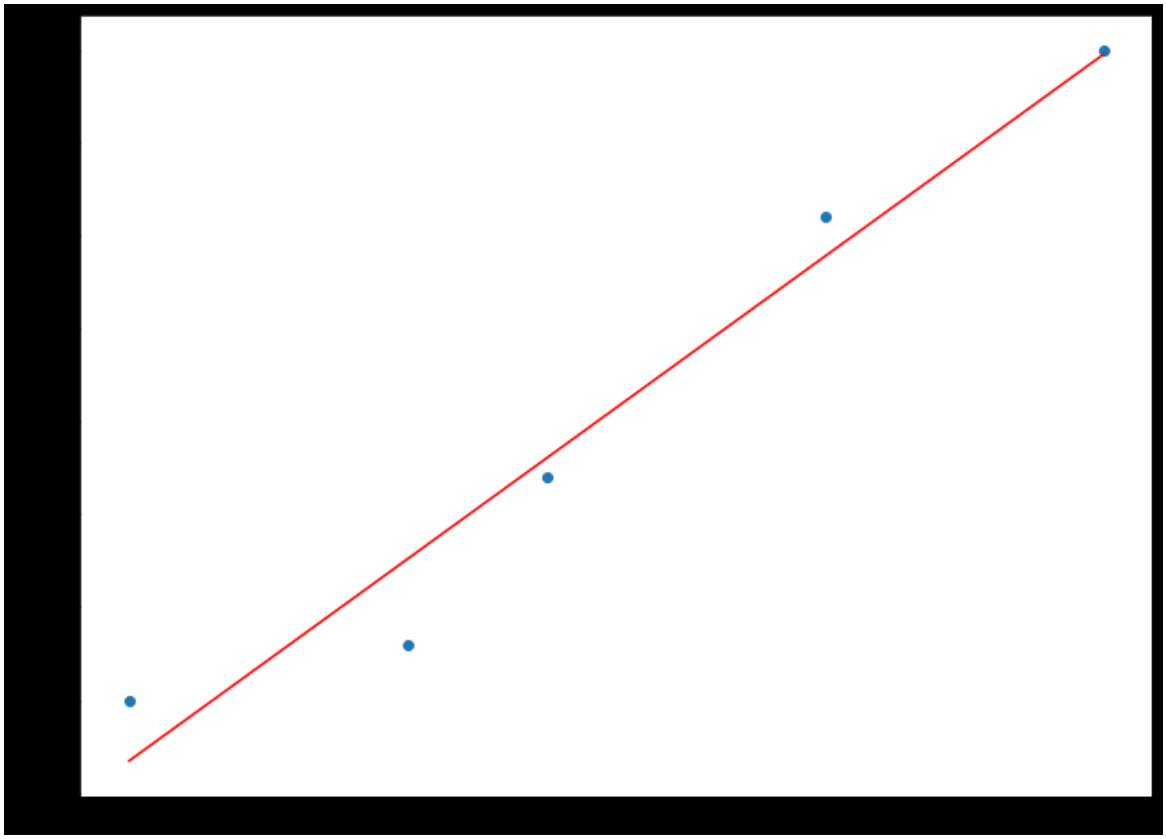
SourceCode:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['figure.figsize'] = (12.0, 9.0)
# Preprocessing Input data
data = pd.read_csv('home.csv')
X = data.iloc[:, 0]
Y = data.iloc[:, 1]
plt.scatter(X, Y)
plt.show()
X_mean = np.mean(X)
print(X_mean)
Y_mean = np.mean(Y)
num = 0
den = 0
for i in range(len(X)):
        num += (X[i] - X_mean)*(Y[i] - Y_mean)
        den += (X[i] - X_mean)**2
m = num / den
c = Y_mean - m*X_mean
print (m, c)
# Y_pred = m*X + c
plt.scatter(X, Y) # actual
# plt.scatter(X, Y_pred, color='red')
plt.plot([min(X), max(X)], [min(Y_pred), max(Y_pred)], color='red') # predicted
plt.show() Making predictions
```

INPUT:

area	**price**
2600	550000
3000	565000
3200	610000
3600	680000
4000	725000

OUTPUT:
Estimated coefficients:
M= 135.7876712328767
C= 180616.43835616444



3) Write a program to demonstrate the working of the decision tree based ID3 algorithm

a. Read the Iris training data from a .CSV file.

b. Output a description of the set of all the decision rules.

c. Output the accuracy of the Iris data

Objectives

Enable the students:

- • To understand the ID3 algorithm.
- • To analyze how to learn from multiple classes.
- • To Develop a classification model by fitting using **ID3** algorithm on a given training set.
- • To evaluate the model using test set.

Description:

The a.csv can be any dataset that that reflects classification problem. Classification task predict the future over a learned model. The given data is divided in to training and test set folds . Here, a model is learned by fitting ID3 (which is a decision tree induction algorithm for categorical data) algorithm over train data. The performance of the learned model is measured on test data. For classification task the data format is like (set of concepts, Target) i.e each instance is associated with group information (class label).

ID3 Algorithm

■ Basic algorithm (a greedy algorithm) ■ Tree is constructed in a top-down recursive divide-and-conquer manner

■ At start, all the training examples are at the root

■ Attributes are categorical (if continuous-valued, they are discredited in advance)

■ Examples are partitioned recursively based on selected attributes

■ Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

■ Conditions for stopping partitioning ■ All samples for a given node belong to the same class

■ There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf

■ There are no samples left

Source Code:

```
import numpy as np
import pandas as pd
eps = np.finfo(float).eps
data=pd.read_csv("play_tennis.csv")
print(data)
# Entropy of the target attribute values
def find_entropy(df):
    target = df.keys()[-1]  # The last dataframe column is the target attribute (playGolf)

    entropy = 0
    values = df[target].unique()

    # for each value in the target playGolf attribute values
    for value in values:
        # ratio of values occurring and entropy
        fraction = df[target].value_counts()[value] / len(df[target])
        entropy += -fraction * np.log2(fraction)
    return entropy
def bestClassifier(df):
    # Entropy_att = []
    # information gain array for all attributes
    IG = []
    # for all attributes excluding target
    for key in df.keys()[:-1]:
        # Entropy_att.append(find_entropy_attribute(df,key))
```

```
        # calculate and record information gain value
        IG.append(find_entropy(df) - find_entropy_attribute(df, key)) #0.940 -0.693= 0.247
    return df.keys()[:-1][np.argmax(IG)]  # IG[0.247, 0.029, 0.152, 0.048 ]
def get_subtable(df, node, value):
    return df[df[node] == value].reset_index(drop=True)
def ID3split(df, tree=None):
    target = df.keys()[-1]



    # Here we build our decision tree


    # Get attribute with maximum information gain
    node = bestClassifier(df) # 0.247


    # Get distinct value of that attribute e.g Salary is node and Low,Med and High are values
    attributeValues = np.unique(df[node])


    # Create an empty dictionary to create tree (recursive-friendly definition)
    if tree is None:              #  Outlook ->root node attribute
        tree = {}
        tree[node] = {}


    # following loop recursively calls ID3split to create and add to the tree
    # it runs till the tree is pure (leaf (result) node branches are added to the tree)
    for value in attributeValues:
        # get the subtable from current node based on the value
```

```
        subtable = get_subtable(df, node, value)
        # get the most common target value in the subtable
        targetValues, counts = np.unique(subtable[target], return_counts=True)

        # if the subtable is empty, assign the leaf node to the most common target value
        if len(counts) == 1:
            tree[node][value] = targetValues[0]
        else:
            # recursively call ID3 to create subtrees
            tree[node][value] = ID3split(subtable)  # Calling the function recursively

    return tree
print("*********************************************************************************")
decisionTree = ID3split(data)
print(decisionTree)
```

INPUT:

day	outlook	temp	humidity	wind	play
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

OUTPUT:

{'Outlook': {'Overcast': 'Yes', 'Rain': {'Wind': {'Strong': 'No', 'Weak': 'Yes'}}, 'Sunny': {'Humidity': {'High': 'No', 'Normal': 'Yes'}}}}