GURU JAMESHWAR UNIVERSITY OF SCIENCE AND TECHNOLOGY

(Hisar-Haryana)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Practical file

Machine Learning

(PCC-CSEAI301-P)

Submitted to :

Dr. Narender

Dept. of CSE

Submitted by :

Sarita

220010150027

B.Tech CSE- AI & ML

INDEX

1. Assignment demonstrating Linear Regression:

a) Implementing linear regression on placement dataset and predicting the dependent variable.

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
df=pd.read_csv("placement_data.csv")
```

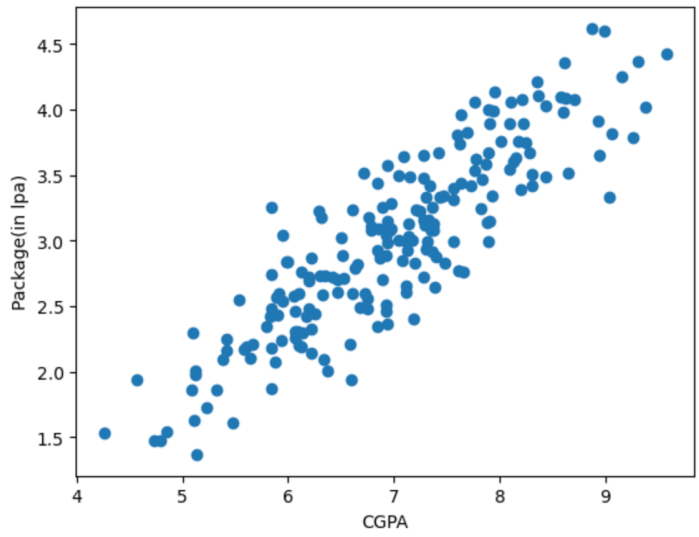
[2]:
```
df.head()
```

[2]:

	cgpa	package
0	6.89	3.26
1	5.12	1.98
2	7.82	3.25
3	7.42	3.67
4	6.94	3.57

[3]:
```
plt.scatter(df['cgpa'],df['package'])
plt.xlabel('CGPA')
plt.ylabel('Package(in lpa)')
```

[3]:
```
Text(0, 0.5, 'Package(in lpa)')
```



[4]:
```
X=df.iloc[:,0:1]
Y=df.iloc[:,-1]
```

[5]:
```
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,random_state=123)
```

[6]:
```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
lr.fit(X_train,Y_train)
```

[6]:
```
LinearRegression
LinearRegression()
```

[7]:
```
X_test[:5]
```

[7]:

	cgpa
50	9.58
127	6.78
37	5.90
149	8.28
19	7.48

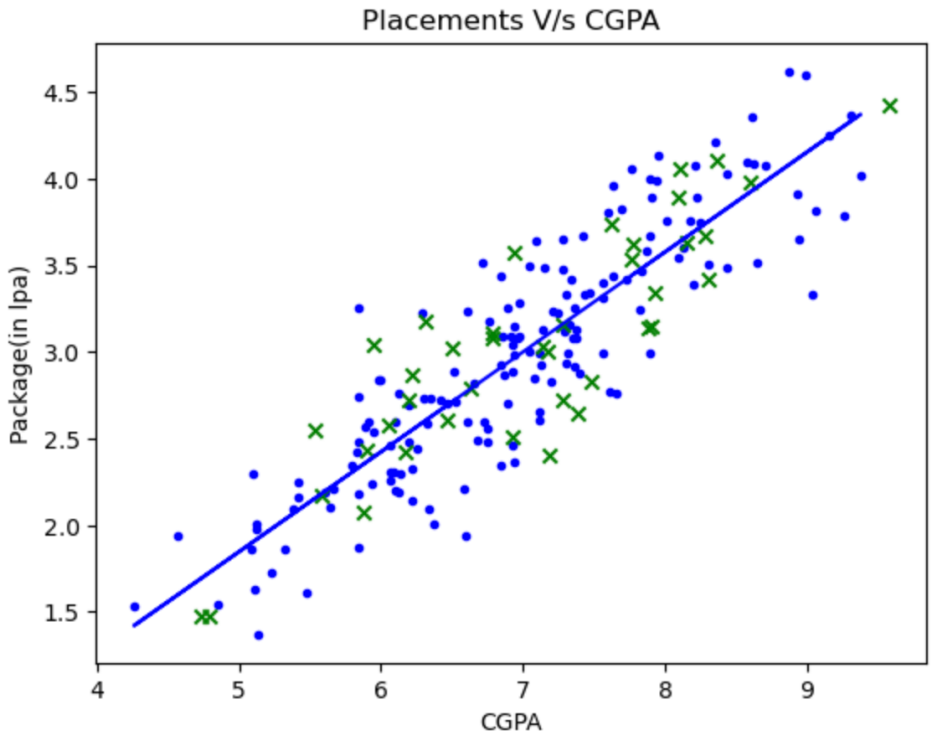
[8]:
```
Y_test[:5]
```

[8]:
```
50     4.43
127    3.11
37     2.43
149    3.67
19     2.83
Name: package, dtype: float64
```

[9]:
```
Y_predicted = lr.predict(X_train)
```

[10]:
```
# Plot the linear fit
plt.scatter(X_train, Y_train, marker='.', c='b')
plt.plot(X_train, Y_predicted, c = "b")
plt.scatter(X_test, Y_test, marker='x', c='g')
plt.title("Placements V/s CGPA")
plt.ylabel('Package(in lpa)')
plt.xlabel('CGPA')
```

[10]:
```
Text(0.5, 0, 'CGPA')
```



b) Implementing linear regression on randomly generated dataset and evaluation of the regression model using R2 score.

[1]:
```
# importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error,r2_score
```

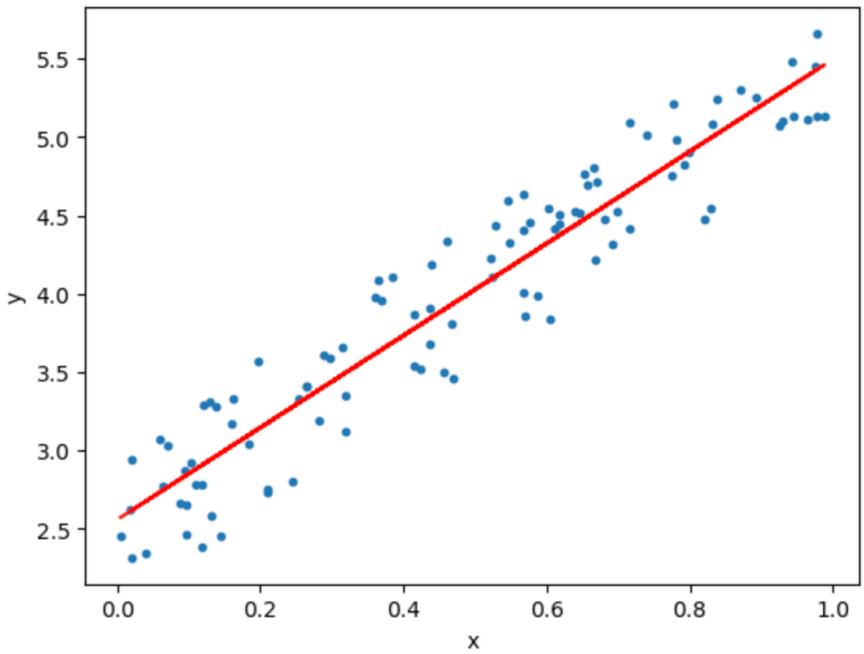
[2]:
```
#generating  a random dataset
np.random.seed(0)
x=np.random.rand(100,1)
y=2+3*x+np.random.rand(100,1)

#sckit_learn implementation

#model_initialisation
r_model=LinearRegression()
#fit the data(Train the model)
r_model.fit(x,y)
#predict
y_predicted=r_model.predict(x)
```

[3]:
```
#model evaluation
rmse=mean_squared_error(y,y_predicted)
r2=r2_score(y,y_predicted)
```

[4]:
```
#printing values
print('slope:',r_model.coef_)
print('intercept:',r_model.intercept_)
print("root mean squared error:",rmse)
print("R2 score:",r2)
```

```
slope: [[2.93655106]]
intercept: [2.55808002]
root mean squared error: 0.07623324582875009
R2 score: 0.9038655568672764
```

[5]:
```
#plotting values
#data points
plt.scatter(x,y,s=10)
plt.xlabel('x')
plt.ylabel('y')

#predicted values
plt.plot(x,y_predicted, c='r')
plt.show()
```



2. Implementing and demonstrating the Find-S algorithm for finding most specific hypothesis using :

a) Cat - non cat dataset.

[1]:
```
#Initialize the hypothesis with the most specific hypothesis
def initialize_hypothesis(attributes):
    hypothesis = {}
    for attribute in attributes:
        hypothesis[attribute] = "null"
    return hypothesis
```

[2]:
```
# Update the hypothesis based on a positive example
def update_hypothesis(hypothesis, example):
    for attribute, value in example.items():
        if hypothesis[attribute] == "null":
            hypothesis[attribute] = value
        elif hypothesis[attribute] != value:
            hypothesis [attribute]="?"
    return hypothesis
```

[3]:
```
#Find-S algorithm
def find_s(training_data):
    attributes= list(training_data[0].keys())
    hypothesis= initialize_hypothesis(attributes)
    for example in training_data:
        if example['target'] == 'cat':
            hypothesis = update_hypothesis(hypothesis, example)
    return hypothesis
```

[4]:
```
#Example training data
training_data = [
     {'color': 'brown', 'size': 'small', 'tail': 'long', 'target': 'cat'},
     {'color': 'gray', 'size': 'medium', 'tail': 'short', 'target': 'cat'},
     {'color': 'black', 'size': 'large', 'tail': 'long', 'target': 'not_cat'},
     {'color': 'white', 'size': 'small', 'tail': 'short', 'target': 'not_cat'}
]
```

[5]:
```
#Apply Find-S algorithm
learned_hypothesis = find_s(training_data)
print("Learned Hypothesis:", learned_hypothesis)
```

```
Learned Hypothesis: {'color': '?', 'size': '?', 'tail': '?', 'target': 'cat'}
```

b) EnjoySport dataset.

[1]:
```
import pandas as pd
import numpy as np
d = pd.read_csv("enjoysport.csv")
print(d)
```

```
     sky air_temp humidity    wind water forecast enjoy_sport
0  sunny     warm   normal  strong  warm     same         yes
1  sunny     warm     high  strong  warm     same         yes
2  rainy     cold     high  strong  warm   change          no
3  sunny     warm     high  strong  cool   change         yes
```

[2]:
```
a = np.array(d)[:,:-1]
print("The attributes are : ", a)
```

```
The attributes are :  [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]
```

[3]:
```
t = np.array(d)[:,-1]
print('The target is : ', t)
```

```
The target is :  ['yes' 'yes' 'no' 'yes']
```

[4]:
```
def train(c, t):
    for i, val in enumerate(t):
        if val == "yes":
            specific_hypothesis = c[i].copy()
            break
    for i, val in enumerate(c):
        if t[i] == "yes":
            for x in range(len(specific_hypothesis)):
                if val[x] != specific_hypothesis[x]:
                    specific_hypothesis[x] = '?'
    return specific_hypothesis
```

[5]:
```
print(" The final hypothesis is:", train(a,t))
```

```
The final hypothesis is: ['sunny' 'warm' '?' 'strong' '?' '?']
```

3. Implementing Candidate Elimination algorithm and finding specific and general boundary sets of hypotheses consistent with EnjoySport dataset via.

a) Program 1

[1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n", concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are: ",target)
```

```
 Instances are:
[['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' 'high' 'strong' 'warm' 'same']
['rainy' 'cold' 'high' 'strong' 'warm' 'change']
['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

Target Values are:  ['yes' 'yes' 'no' 'yes']
```

[2]:
```
def learn(concepts, target):
    specific_h = concepts[0].copy()
    print("\n Initlalization of specific_h and genearal_h")
    print("\n Specific Boundary: ", specific_h)
    general_h = [["?" for i in range(len(specific_h))] for i in range(len(specific_h))]
    print("\nGeneric Boundary: ", general_h)

    for i,h in enumerate(concepts):
        print("\nInstance ", i+1, " is ", h)
        if target[i] == "yes":
            print("Instance is Positive")
            for x in range(len(specific_h)):
                if h[x] != specific_h[x]:
                    specific_h[x] = "?"
                    general_h[x][x] = '?'

        if target[i] == "no":
            print("Instance is Negative")
            for x in range(len(specific_h)):
                if h[x]!= specific_h[x]:
                    general_h[x][x] = specific_h[x]
                else:
                    general_h[x][x] = '?'
        print("Specific Bundary after ", i+1, "Instance is ", specific_h)
        print("Generic Boundary after ", i+1, "Instance is ", general_h)
        print("\n")

    indices = [i for i, val in enumerate(general_h) if val == ['?', '?', '?', '?', '?', '?']]
    for i in indices:
        general_h.remove(['?', '?', '?', '?', '?', '?'])

    return specific_h, general_h
```

[3]:
```
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General h: ", g_final, sep="\n")
```

```
 Initlalization of specific_h and genearal_h

 Specific Boundary:  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']

Generic Boundary:  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

Instance  1  is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  1 Instance is  ['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
Generic Boundary after  1 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance  2  is  ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
Instance is Positive
Specific Bundary after  2 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  2 Instance is  [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]



Instance  3  is  ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
Instance is Negative
Specific Bundary after  3 Instance is  ['sunny' 'warm' '?' 'strong' 'warm' 'same']
Generic Boundary after  3 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'same']]



Instance  4  is  ['sunny' 'warm' 'high' 'strong' 'cool' 'change']
Instance is Positive
Specific Bundary after  4 Instance is  ['sunny' 'warm' '?' 'strong' '?' '?']
Generic Boundary after  4 Instance is  [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]


Final Specific_h:
['sunny' 'warm' '?' 'strong' '?' '?']
Final General h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

b) Program 2

[1]:
```
import numpy as np
import pandas as pd
data = pd.read_csv('enjoysport.csv')
concepts = np.array(data.iloc[:,0:-1])
print("\n Instances are:\n", concepts)
target = np.array(data.iloc[:,-1])
print("\n Target Values are: ",target)
```

```
 Instances are:
 [['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
 ['sunny' 'warm' 'high' 'strong' 'warm' 'same']
 ['rainy' 'cold' 'high' 'strong' 'warm' 'change']
 ['sunny' 'warm' 'high' 'strong' 'cool' 'change']]

 Target Values are:  ['yes' 'yes' 'no' 'yes']
```

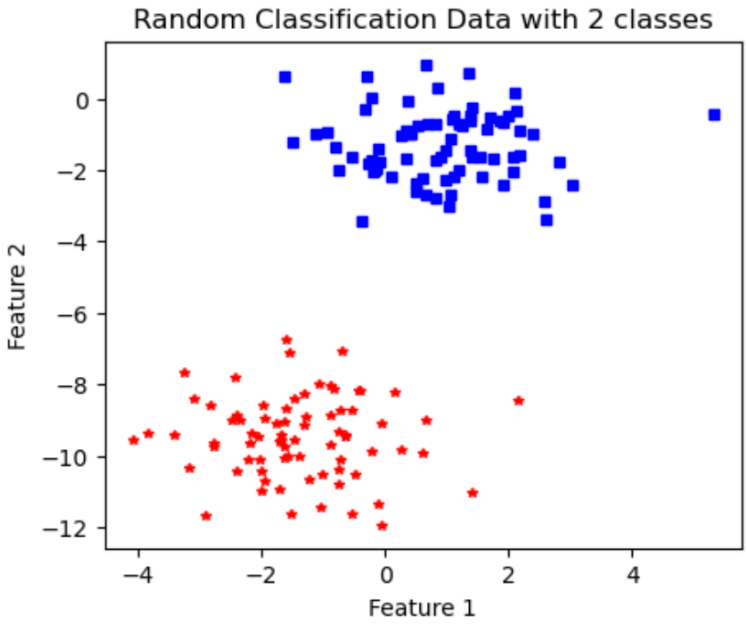
[2]:
```
def candidate_elimination(concepts, target):
    S = concepts[0].copy()
    G = [["?" for _ in range(len(S))]]

    for i,h in enumerate(concepts):
        if target[i] == "yes":
            for x in range(len(S)):
                if h[x] != S[x]:
                    S[x] = "?"
            G = [g for g in G if all(g[x] == "?" or g[x] == h[x] for x in range(len(g)))]

        elif target[i] == "no":
            G_prev = G.copy()
            for g in G_prev:
                for x in range(len(g)):
                    if g[x] == "?":
                        for val in set(concepts[:, x]):
                            if val != h[x]:
                                new_g = g.copy()
                                new_g[x] = val
                                if any(new_g[j] != S[j] and S[j] != "?" for j in range(len(S))):
                                    G.append(new_g)
                G.remove(g)
            G = [g for g in G if any(all(g[x] == "?" or g[x] == S[x] for x in range(len(g))) for S in [S])]

    return S, G
# Run the Candidate-Elimination algorithm
S_final, G_final = candidate_elimination(concepts, target)
S_final, G_final
```

[2]:
```
(array(['sunny', 'warm', '?', 'strong', '?', '?'], dtype=object),
 [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']])
```

4. Implementing Perceptron learning from scratch and showing decision boundary.

[1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets
X, y = datasets.make_blobs(n_samples=150, n_features=2, centers=2,
                           cluster_std=1.05, random_state=2)
#Plotting
X.shape
fig = plt.figure(figsize=(5,4))
plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "r*", markersize=4)
plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs', markersize=4)
plt.xlabel("Feature 1")
plt.ylabel("Feature 2")
plt.title('Random Classification Data with 2 classes')
```

[1]:
```
Text(0.5, 1.0, 'Random Classification Data with 2 classes')
```



[2]:
```
def perceptron(X, y, lr, epochs):
    m, n = X.shape
    theta = np.zeros((n+1,1))
    n_miss_list = []
    for epoch in range(epochs):
        n_miss = 0
        for idx, x_i in enumerate(X):
            x_i = np.insert(x_i, 0, 1).reshape(-1,1)
            y_hat = step_func(np.dot(x_i.T, theta))
            if (np.squeeze(y_hat) - y[idx]) != 0:
                theta += lr*((y[idx] - y_hat)*x_i)
                n_miss += 1
        n_miss_list.append(n_miss)
    return theta, n_miss_list
```

[3]:
```
def step_func(z):
    return 1.0 if (z>0) else 0.0
```

[4]:
```
def plot_decision_boundary(X, theta):
    x1 = [min(X[:,0]), max(X[:,0])]
    m = -theta[1]/theta[2]
    c = -theta[0]/theta[2]
    x2 = m*x1 + c
    fig = plt.figure(figsize=(5,4))
    plt.plot(X[:, 0][y == 0], X[:, 1][y == 0], "r*", markersize=4)
    plt.plot(X[:, 0][y == 1], X[:, 1][y == 1], 'bs', markersize=4)
    plt.xlabel("Feature 1")
    plt.ylabel("Feature 2")
    plt.title('Perceptron Algorithm')
    plt.plot(x1, x2,"y-")
```

[5]:
```
theta, miss_1 = perceptron(X, y , 0.5, 100)
plot_decision_boundary(X, theta)
```