GURU JAMBHESHWAR 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")
```

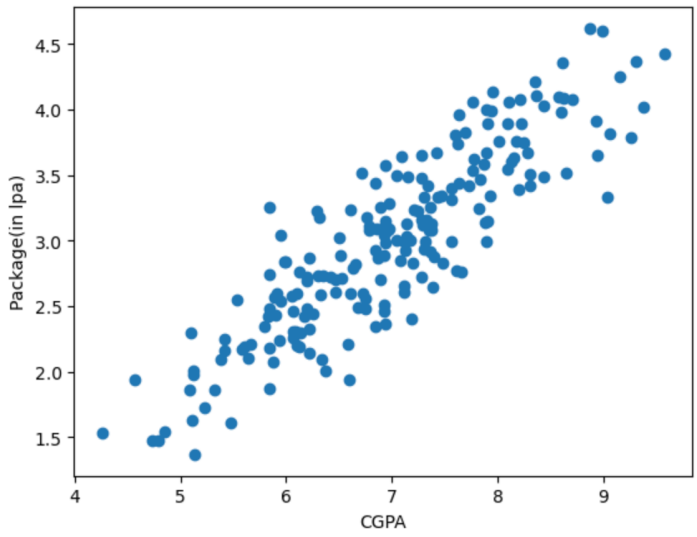
[2]:
```
df.head()
```

[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

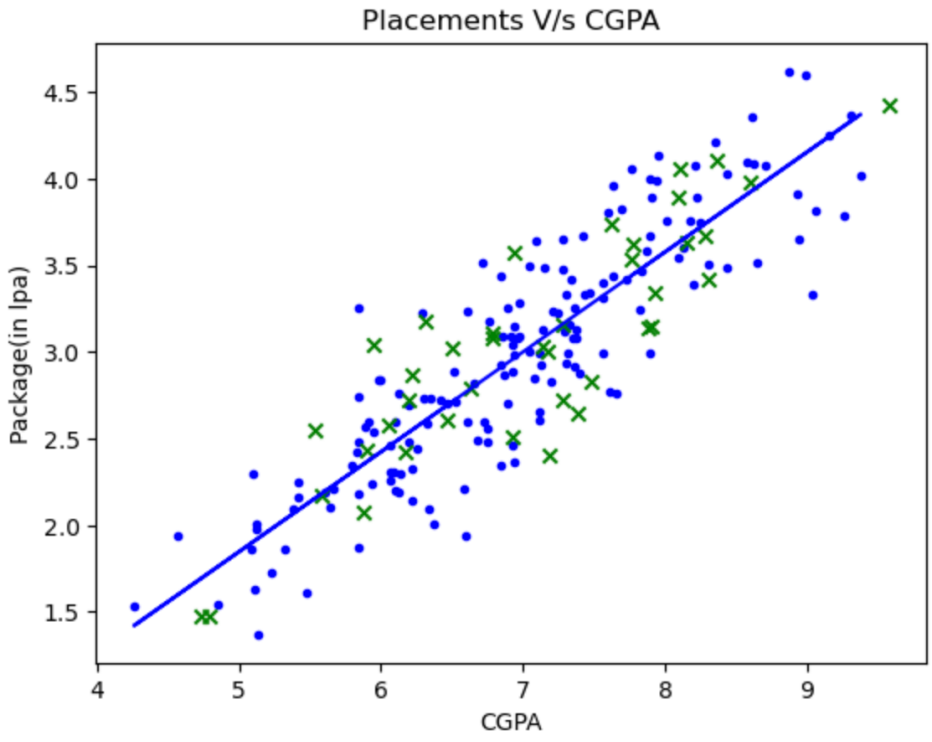
[8]:
```
Y_test[:5]
```

[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
```

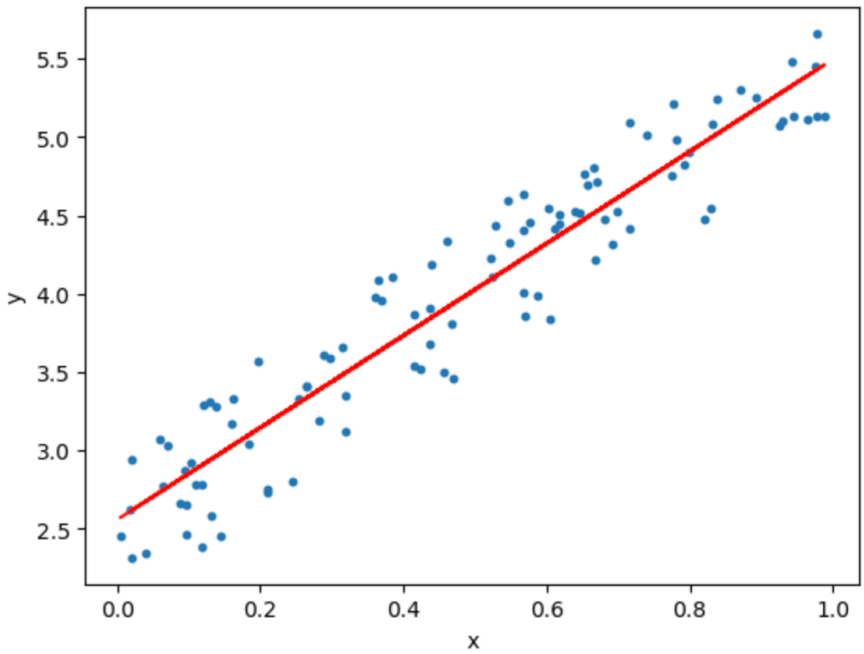
[2]:
```
#generating  a random dataset
np.random.seed(0)
x=np.random.rand(100,1)
y=2+3*x+np.random.rand(100,1)

#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 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'}
```

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

[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
```

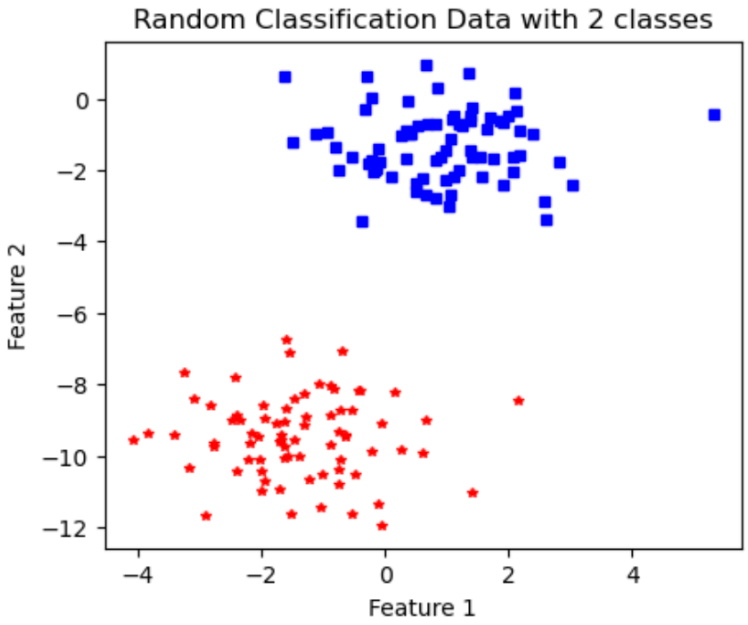
[3]:
```
s_final, g_final = learn(concepts, target)
print("Final Specific_h: ", s_final, sep="\n")
print("Final General h: ", g_final, sep="\n")
```

```
 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', '?', '?', '?', '?']]
```

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')
```

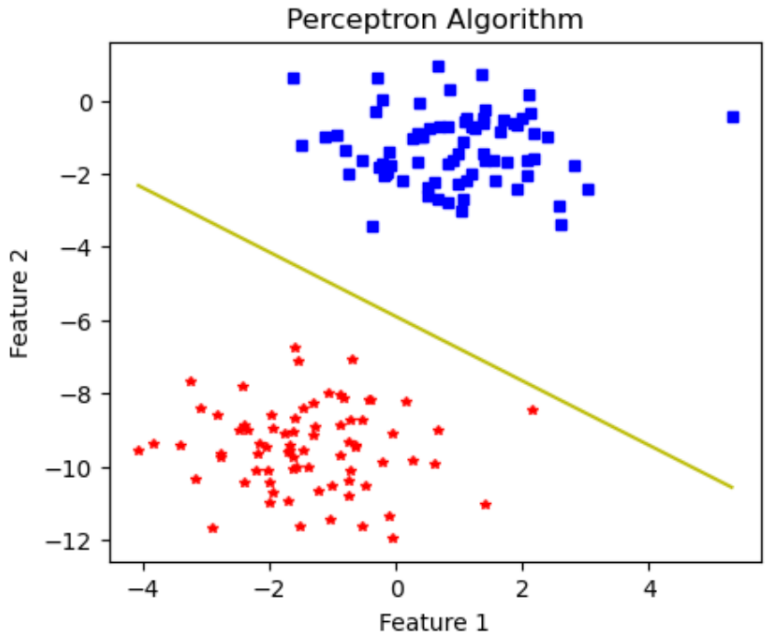


[2]:
```
def perceptron(X, y, lr, epochs):
    m, n = X.shape
    theta = np.zeros((n+1,1))
    n_miss_list = []
    for epoch in range(epochs):
        n_miss = 0
        for idx, x_i in enumerate(X):
            x_i = np.insert(x_i, 0, 1).reshape(-1,1)
            y_hat = step_func(np.dot(x_i.T, theta))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



5. Implementing classification using SVM :

a) For iris dataset using SVC default settings.

[1]:
```
from sklearn.datasets import load_iris
iris = load_iris()
dir(iris)
iris.data
iris.target
iris.target_names
iris.feature_names
```

[1]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

[2]:
```
import pandas as pd
df = pd.DataFrame(iris.data, columns=iris.feature_names)
df.head()
df['target'] = iris.target
df.head()
```

[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

[3]:
```
import matplotlib.pyplot as plt
import seaborn as sns
```

[4]:
```
sns.pairplot(df, hue='target', palette = 'brg')
plt.show()
```

[5]:
```
x = df.drop(['target'], axis = 'columns')
y = df.target
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)
```

[6]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

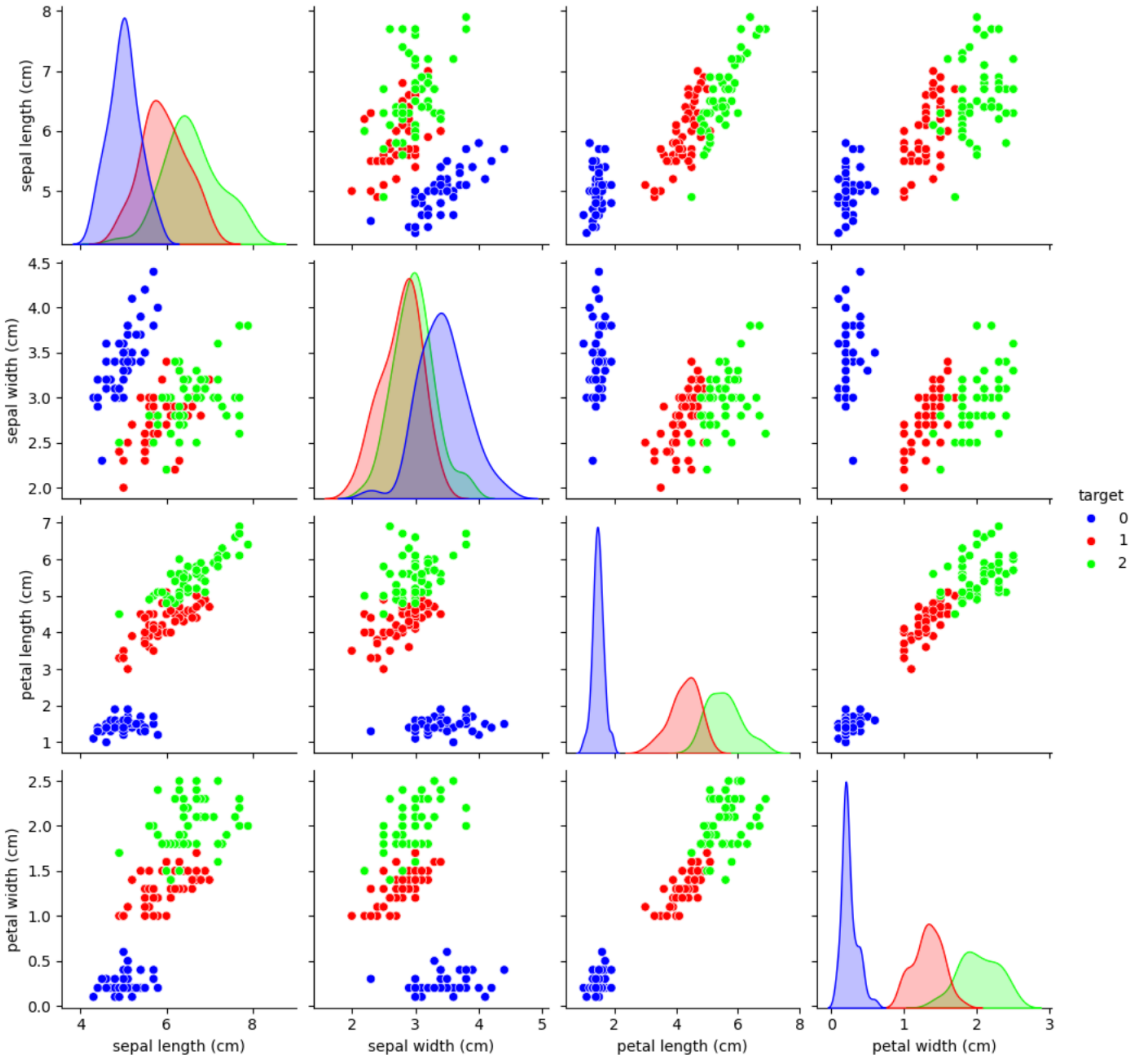
[6]:
```
0.9777777777777777
```

[7]:
```
model.predict(pd.DataFrame([iris.data[50]],columns=iris.feature_names))
```

[7]:
```
array([1])
```

[8]:
```
model.predict(pd.DataFrame([[6.5, 3.0, 5.2, 2.0]],columns=iris.feature_names))
```

[8]:
```
array([2])
```


sepal length (cm)
sepal width (cm)
petal length (cm)
petal width (cm)
target
0
1
2

b) For recognition of handwritten digits.

[1]:
```
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits
digits = load_digits()
dir(digits)
```

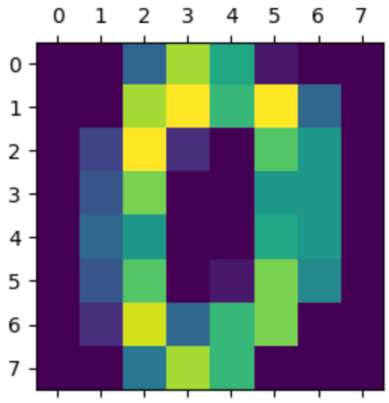
[1]:
```
['DESCR', 'data', 'feature_names', 'frame', 'images', 'target', 'target_names']
```

[2]:
```
digits.data[0]
```

[2]:
```
array([ 0.,  0.,  5., 13.,  9.,  1.,  0.,  0.,  0.,  0., 13., 15., 10.,
       15.,  5.,  0.,  0.,  3., 15.,  2.,  0., 11.,  8.,  0.,  0.,  4.,
       12.,  0.,  0.,  8.,  8.,  0.,  0.,  5.,  8.,  0.,  0.,  9.,  8.,
        0.,  0.,  4., 11.,  0.,  1., 12.,  7.,  0.,  0.,  2., 14.,  5.,
       10., 12.,  0.,  0.,  0.,  0.,  6., 13., 10.,  0.,  0.,  0.])
```

[3]:
```
plt.figure(figsize=(3,3))
plt.matshow(digits.images[0],fignum=1)
digits.target [0]
```

[3]:
```
0
```



[4]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(digits.data, digits.target, test_size=0.2)
x_train
```

[4]:
```
array([[ 0.,  0.,  7., ..., 16.,  9.,  0.],
       [ 0.,  0., 15., ...,  0.,  0.,  0.],
       [ 0.,  1.,  8., ...,  0.,  0.,  0.],
       ...,
       [ 0.,  0.,  0., ...,  0.,  0.,  0.],
       [ 0.,  0.,  0., ..., 14.,  5.,  0.],
       [ 0.,  0.,  0., ..., 15.,  8.,  0.]])
```

[5]:
```
from sklearn.svm import SVC
model = SVC()
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[5]:
```
0.9888888888888889
```

If we want to use different kernels and find the accuracy levels, we can use the following code :

[7]:
```
# C=1.0, kernel='linear', gamna='scale'
model = SVC(kernel='linear')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[7]:
```
0.9944444444444445
```

[8]:
```
# C=1.0, kernel='sigmoid', gamma='scale'
model = SVC(kernel='sigmoid')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[8]: 0.9055555555555556

[9]:
```
# C=1.0, kernel='poly', gamna='scale'
model = SVC(kernel='poly')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

[9]: 0.9944444444444445

[10]:
```
# C=1.0, kernel='poly', gamma='auto'
model = SVC(kernel='poly', gamma='auto')
model.fit(x_train, y_train)
model.score(x_test, y_test)
```

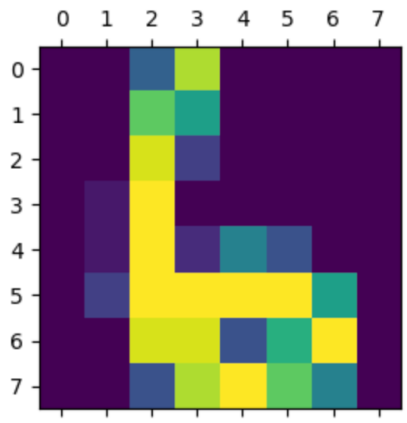
[10]: 0.9944444444444445

[11]:
```
x_test[100]
print(f'Predicted Digit: {model.predict([x_test[100]])} \nActual Digit : {y_test[100]}')
```

```
Predicted Digit: [6]
Actual Digit : 6
```

[12]:
```
plt.figure(figsize=(3,3))
plt.matshow(digits.images[67],fignum=1)
model.predict([digits.data[67]])
digits.target[67]
```

[12]: 6

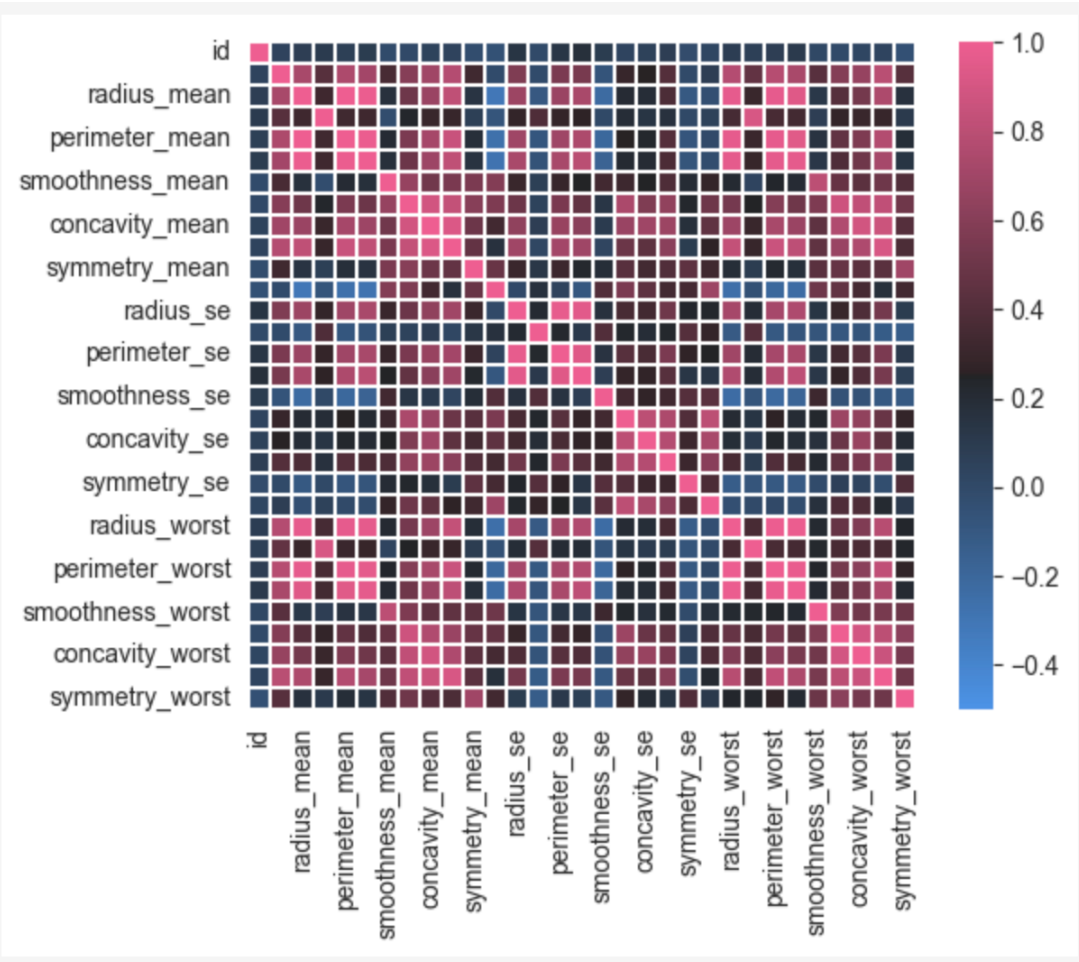


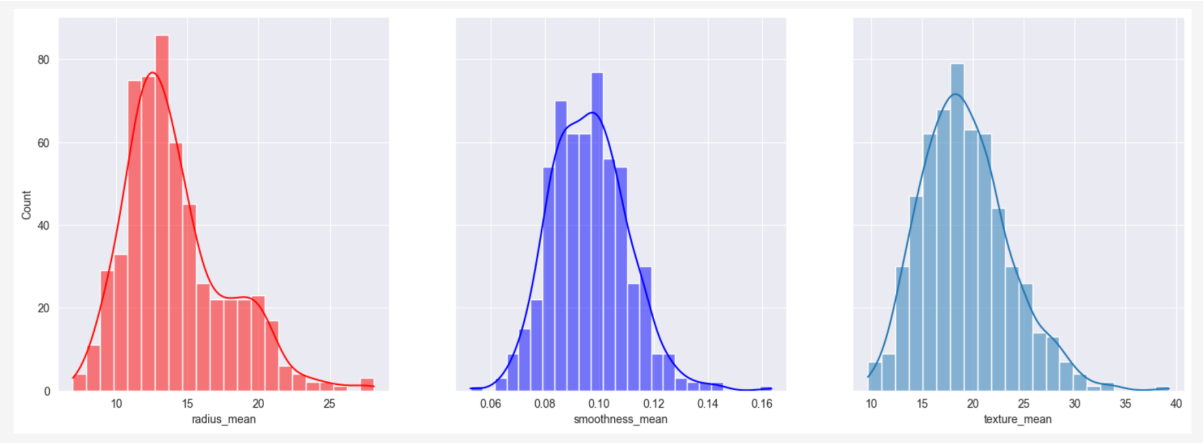
6. Implement Naïve Bayes Classifier

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
sns.set_style("darkgrid")
data = pd.read_csv("breast-cancer.csv")
data.head(10)

# Convert diagnosis column to numerical values
data["diagnosis"] = data["diagnosis"].map({'M': 1, 'B': 0})

data["diagnosis"].hist()
corr = data.iloc[:,:-1].corr(method="pearson")
cmap = sns.diverging_palette(250, 354, 80, 60, center = "dark", as_cmap=True)
sns.heatmap(corr, vmax=1, vmin=-.5, cmap=cmap, square=True, linewidths=.2)
data = data[["radius_mean", "texture_mean", "smoothness_mean", "diagnosis"]]
data.head(10)
fig, axes = plt.subplots(1, 3, figsize=(18, 6), sharey=True)
sns.histplot(data, ax=axes[0], x="radius_mean", kde=True, color='r')
sns.histplot(data, ax=axes[1], x="smoothness_mean", kde=True, color='b')
sns.histplot(data, ax=axes[2], x="texture_mean", kde=True)
```

[1] ✓ 0.7s

```
<Axes: xlabel='texture_mean', ylabel='Count'>
```




Count
80
60
40
20
0
10
15
20
25
radius_mean
0.06
0.08
0.10
0.12
0.14
0.16
smoothness_mean
10
15
20
25
30
35
40
texture_mean

7. Implementing PCA on Iris Dataset.

```
import numpy as np
import pandas as pd
import seaborn as sns
from sklearn.datasets import load_iris

# Load the iris dataset
iris = load_iris()

# Convert to DataFrame
df = pd.DataFrame(data=iris.data, columns=iris.feature_names)
df['target'] = iris.target

# Display the first few rows
df.head()
```

[1] ✓ 0.7s

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

```
# Display the last few rows
data = np.array(df)
instance_space = data[:,0:data.shape[1]-1]
target_class = data[:,-1]
print(instance_space[1:5])
print(target_class[1:5])
```

[2] ✓ 0.0s

```
[[4.9 3.  1.4 0.2]
 [4.7 3.2 1.3 0.2]
 [4.6 3.1 1.5 0.2]
 [5.  3.6 1.4 0.2]]
[0. 0. 0. 0.]
```

```
# z score normalization
z1 = (instance_space[:,0]-np.mean(instance_space[:,0]))/np.std(instance_space[:,0])
z2 = (instance_space[:,1]-np.mean(instance_space[:,1]))/np.std(instance_space[:,1])
z3 = (instance_space[:,2]-np.mean(instance_space[:,2]))/np.std(instance_space[:,2])
z4 = (instance_space[:,3]-np.mean(instance_space[:,3]))/np.std(instance_space[:,3])
new_array = np.array([z1,z2,z3,z4]).T
print(new_array[0:4:,0:4])
```

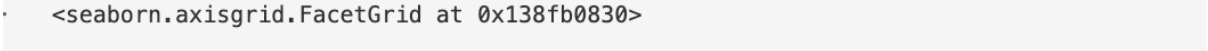
[3] ✓ 0.0s

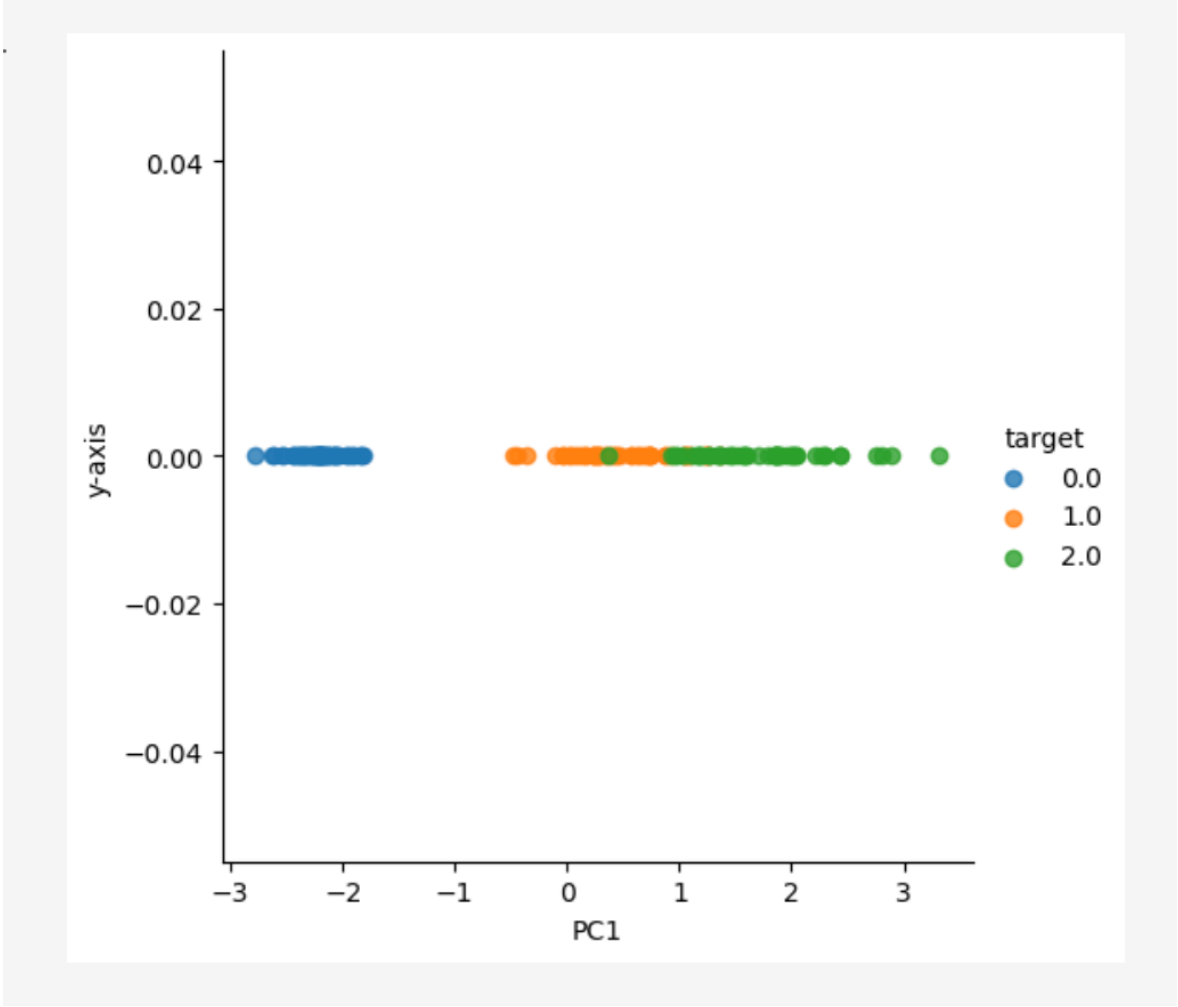
```
[[-0.90068117  1.01900435 -1.34022653 -1.3154443 ]
 [-1.14301691 -0.13197948 -1.34022653 -1.3154443 ]
 [-1.38535265  0.32841405 -1.39706395 -1.3154443 ]
 [-1.50652052  0.09821729 -1.2833891  -1.3154443 ]]
```

```
#creating covariance matrix
transposed_new_array=new_array.T
print(transposed_new_array.shape)
a=np.array(transposed_new_array.dot(new_array)/150,dtype=float)
print(a.dtype)
eigen_values,eigen_vectors=np.linalg.eig(a)
print(eigen_values,eigen_vectors)
for i in range(len(eigen_values)):
    print(eigen_values[i]/sum(eigen_values))
selected_eigen_vector=eigen_vectors[:,0].reshape((4,1))
projected_x=new_array.dot(selected_eigen_vector)
print(projected_x)
new_df=pd.DataFrame(projected_x,columns=['PC1'])
new_df['y-axis']=0.0
new_df['target']=target_class
print(new_df)
sns.lmplot(x='PC1',y='y-axis',data=new_df,fit_reg=False,hue='target')
```

[4] ✓ 0.1s

```
[2.91849782 0.91403047 0.14675688 0.02071484] [[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]
0.7296244541329989
0.2285076178670176
0.03668921889282877
0.005178709107154799
[[-2.26470281]
 [-2.08096115]
 [-2.36422905]
 [-2.29938422]
 [-2.38984217]
 [-2.07563095]
 [-2.44402884]
 [-2.23284716]
 [-2.33464048]
 [-2.18432817]
 [-2.1663101 ]
 [-2.32613087]
 [ 1.37278779]
 [ 0.96065603]]
          PC1  y-axis  target
0   -2.264703     0.0     0.0
1   -2.080961     0.0     0.0
2   -2.364229     0.0     0.0
3   -2.299384     0.0     0.0
4   -2.389842     0.0     0.0
..        ...     ...     ...
145  1.870503     0.0     2.0
146  1.564580     0.0     2.0
147  1.521170     0.0     2.0
148  1.372788     0.0     2.0
149  0.960656     0.0     2.0

[150 rows x 3 columns]
```




0.04
0.02
0.00
−0.02
−0.04
y-axis
−3
−2
−1
0
1
2
3
PC1
target
0.0
1.0
2.0

8. Implementing Gradient Descent Algorithm

```
[1]  ✓ 0.2s
```

```python
import numpy as np
import matplotlib.pyplot as plt
```

```
[2]  ✓ 0.0s
```

```python
def gradient_descent(initial_x, learning_rate, num_iterations):
    x_values = []
    y_values = []
    x = initial_x
    for i in range(num_iterations):
        # Calculate the gradient of the function at the current point
        gradient = 2 * x
        # Update the value of x using the gradient and learning rate
        x = x - learning_rate * gradient
        # Store the current values for plotting
        x_values.append(x)
        y_values.append(x**2 + 5)
    return x_values, y_values
```

```
[3]  ✓ 0.0s
```

```python
# Set initial parameters
initial_x = 3.0
learning_rate = 0.1
num_iterations = 20
# Run gradient descent
x_values, y_values = gradient_descent(initial_x, learning_rate, num_iterations)
# Plotting
plt.plot(x_values, y_values, marker='o', linestyle='-')
plt.title('Gradient Descent Optimization')
plt.xlabel('x')
plt.ylabel('f(x)')
plt.show()
```