Madhukesh Bhat M
1BM18CS051
6B, CSE

ML Lab Report

Implement and demonstrate the FIND-S algorithm for finding the most specific hypothesis based on a given set of training data samples.

In [ ]:
```
import pandas as pd
```

In [3]:
```
weatherInfo = pd.read_csv('Data.csv')
weatherInfo
```

Out[3]:

	Weather	AirTemperature	Humidity	WaterTemperature	Wind	Goes out
0	Sunny	Moderate	Normal	Warm	Strong	Yes
1	Sunny	High	Normal	Too Warm	Strong	No
2	Rainy	Low	High	Cold	Breezy	No
3	Rainy	High	High	Warm	Breezy	Yes
4	Snowy	Moderate	Normal	Cold	Strong	Yes

In [16]:
```
import numpy as np
data = np.array(weatherInfo)[:,:-1]
data
```

Out[16]:
```
array([['Sunny', 'Moderate', 'Normal', 'Warm', 'Strong'],
       ['Sunny', 'High', 'Normal', 'Too Warm ', 'Strong'],
       ['Rainy', 'Low', 'High', 'Cold', 'Breezy'],
       ['Rainy', 'High', 'High', 'Warm', 'Breezy'],
       ['Snowy', 'Moderate', 'Normal', 'Cold', 'Strong']], dtype=object)
```

In [18]:
```
values = np.array(weatherInfo)[:,-1]
values
```

Out[18]:
```
array(['Yes', 'No', 'No', 'Yes', 'Yes'], dtype=object)
```

In [34]:
```
hypothesis = ['NULL'] * len(data[0])
print('Initial hypothesis:', hypothesis)
for j in range(0, len(values)):
    if values[j] == 'Yes':
        for i in range(0, len(data[0])):
            if hypothesis[i] == 'NULL' or hypothesis[i] == data[j][i]:
                hypothesis[i] = data[j][i]
            else:
                hypothesis[i] = '?'
    print('After', j, 'iteration in dataset, the hypothesis is:', hypothesis)
print('Final hypothesis:', hypothesis)
```

```
Initial hypothesis: ['NULL', 'NULL', 'NULL', 'NULL', 'NULL']
After 0 iteration in dataset, the hypothesis is: ['Sunny', 'Moderate', 'Normal', 'Warm', 'Strong']
After 1 iteration in dataset, the hypothesis is: ['Sunny', 'Moderate', 'Normal', 'Warm', 'Strong']
After 2 iteration in dataset, the hypothesis is: ['Sunny', 'Moderate', 'Normal', 'Warm', 'Strong']
After 3 iteration in dataset, the hypothesis is: ['?', '?', '?', 'Warm', '?']
After 4 iteration in dataset, the hypothesis is: ['?', '?', '?', '?', '?']
Final hypothesis: ['?', '?', '?', '?', '?']
```

For a given set of training data examples stored in a .CSV file, implement and demonstrate the Candidate-Elimination algorithm to output a description of the set of all hypotheses consistent with the training examples.

In [56]:
```python
import pandas as pd
```

In [57]:
```python
goesOut = pd.read_csv("Data.csv")
goesOut
```

Out[57]:

	Weather	AirTemperature	Humidity	WaterTemperature	Wind	Goes out
0	Sunny	Moderate	Normal	Warm	Strong	Yes
1	Sunny	High	Normal	Cold	Strong	No
2	Rainy	Moderate	High	Cold	Strong	No
3	Sunny	High	High	Warm	Strong	Yes
4	Sunny	Moderate	Normal	Cold	Strong	Yes

In [58]:
```python
import numpy as np
data = np.array(goesOut)[:,:-1]
data
```

Out[58]:
```
array([['Sunny', 'Moderate', 'Normal', 'Warm', 'Strong'],
       ['Sunny', 'High', 'Normal', 'Cold', 'Strong'],
       ['Rainy', 'Moderate', 'High', 'Cold', 'Strong'],
       ['Sunny', 'High', 'High', 'Warm', 'Strong'],
       ['Sunny', 'Moderate', 'Normal', 'Cold', 'Strong']], dtype=object)
```

In [59]:
```python
values = np.array(goesOut)[:,-1]
values
```

Out[59]:
```
array(['Yes', 'No', 'No', 'Yes', 'Yes'], dtype=object)
```

In [62]:
```python
specHypothesis = data[0].copy()
genHypothesis = [['?' for i in range(len(specHypothesis))] for i in range(len(specHypothesis))]
print(f"Initial Specific Hypothesis: {specHypothesis}")
print(f"Initial General Hypothesis: {genHypothesis}")
```

```
Initial Specific Hypothesis: ['Sunny' 'Moderate' 'Normal' 'Warm' 'Strong']
Initial General Hypothesis: [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
```

In [63]:
```python
for i, h in enumerate(data):
    print(f"Iteration {i}:")
    print(f"Instance {i}: {h}")
    if values[i] == "Yes":
        print("This is a POSITIVE instance")
        for j in range(len(specHypothesis)):
            if h[j] != specHypothesis[j]:
                genHypothesis[j][j] = '?'
                specHypothesis[j] = '?'
    else:
        print("This is a NEGATIVE instance")
        for j in range(len(specHypothesis)):
            if h[j] != specHypothesis[j]:
                genHypothesis[j][j] = specHypothesis[j]
            else:
                genHypothesis[j][j] = '?'
    print(f"Specific Hypothesis after this iteration: {specHypothesis}")
    print(f"General Hypothesis after this iteration: {genHypothesis}")
    print()
```

```
Iteration 0:
Instance 0: ['Sunny' 'Moderate' 'Normal' 'Warm' 'Strong']
This is a POSITIVE instance
Specific Hypothesis after this iteration: ['Sunny' 'Moderate' 'Normal' 'Warm' 'Strong']
General Hypothesis after this iteration: [['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]

Iteration 1:
Instance 1: ['Sunny' 'High' 'Normal' 'Cold' 'Strong']
This is a NEGATIVE instance
Specific Hypothesis after this iteration: ['Sunny' 'Moderate' 'Normal' 'Warm' 'Strong']
General Hypothesis after this iteration: [['?', '?', '?', '?', '?'], ['?', 'Moderate', '?', '?', '?'], ['?',
'?', '?', '?', '?'], ['?', '?', '?', 'Warm', '?'], ['?', '?', '?', '?', '?']]

Iteration 2:
Instance 2: ['Rainy' 'Moderate' 'High' 'Cold' 'Strong']
This is a NEGATIVE instance
Specific Hypothesis after this iteration: ['Sunny' 'Moderate' 'Normal' 'Warm' 'Strong']
General Hypothesis after this iteration: [['Sunny', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'Normal', '?', '?'], ['?', '?', '?', 'Warm', '?'], ['?', '?', '?', '?', '?']]

Iteration 3:
Instance 3: ['Sunny' 'High' 'High' 'Warm' 'Strong']
This is a POSITIVE instance
Specific Hypothesis after this iteration: ['Sunny' '?' '?' 'Warm' 'Strong']
General Hypothesis after this iteration: [['Sunny', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?'], ['?', '?', '?', 'Warm', '?'], ['?', '?', '?', '?', '?']]

Iteration 4:
Instance 4: ['Sunny' 'Moderate' 'Normal' 'Cold' 'Strong']
This is a POSITIVE instance
Specific Hypothesis after this iteration: ['Sunny' '?' '?' '?' 'Strong']
General Hypothesis after this iteration: [['Sunny', '?', '?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?']]
```

In [64]:
```python
indices = [i for i, val in enumerate(genHypothesis) if val == ['?','?','?','?','?']]
for i in indices:
    genHypothesis.remove(['?','?','?','?','?'])
print(f"Final Specific Hypothesis: {specHypothesis}")
print(f"Final General Hypothesis: {genHypothesis}")
```

```
Final Specific Hypothesis: ['Sunny' '?' '?' '?' 'Strong']
Final General Hypothesis: [['Sunny', '?', '?', '?', '?']]
```

Write a program to demonstrate the working of the decision tree based ID3 algorithm. Use an appropriate data set for building the decision tree and apply this knowledge to classify a new sample.

In [32]:
```python
import pandas as pd
import numpy as np
import math
```

In [33]:
```python
df = pd.read_csv('PlayTennis.csv')
df.head()
```

Out[33]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [34]:
```python
data = df.iloc[:,1:]
data.head()
```

Out[34]:

	Outlook	Temperature	Humidity	Wind
0	Sunny	Hot	High	Weak
1	Sunny	Hot	High	Strong
2	Overcast	Hot	High	Weak
3	Rain	Mild	High	Weak
4	Rain	Cool	Normal	Weak

In [35]:
```python
target = df.iloc[:,:1]
target.head()
```

Out[35]:

	PlayTennis
0	No
1	No
2	Yes
3	Yes
4	Yes

In [36]:
```python
def get_attributes(df):
    attributes = df.columns.tolist()[1:]
    return attributes
get_attributes(df)
```

Out[36]:
```
['Outlook', 'Temperature', 'Humidity', 'Wind']
```

In [37]:
```python
def get_attributes_values(attributes, df):
    values = {}
    for i in attributes:
        values[i] = df[i].unique().tolist()
    return values
get_attributes_values(get_attributes(df), df)
```

Out[37]:
```
{'Outlook': ['Sunny', 'Overcast', 'Rain'],
 'Temperature': ['Hot', 'Mild', 'Cool'],
 'Humidity': ['High', 'Normal'],
 'Wind': ['Weak', 'Strong']}
```

In [38]:
```python
def calculate_entropy(df):
    positive = len(df[df['PlayTennis'] == 'Yes'])
    negative = len(df[df['PlayTennis'] == 'No'])
    if negative == positive:
        return 1, positive, negative
    elif negative == 0 or positive == 0:
        return 0, positive, negative
    else:
        entropy = (((-1) * positive * math.log(positive / (positive + negative), 2) / (positive + negative)) +
                   ((-1) * negative * math.log(negative / (positive + negative), 2) / (positive + negative)))
        return float(entropy), positive, negative
#calculate_entropy(df)
```

In [39]:
```python
def calculate_information_gain(df, attribute, attribute_values):
    information_gain = 0
    for value in attribute_values:
        pos_values = len(df.where((df[attribute] == value) & (df['PlayTennis'] == 'Yes')).dropna())
        neg_values = len(df.where((df[attribute] == value) & (df['PlayTennis'] == 'No')).dropna())
        entropy, positives, negatives = calculate_entropy(df[df[attribute] == value])
        information_gain += ((pos_values + neg_values) / (positives + negatives)) * entropy
    return information_gain
```

In [40]:
```python
class Node:
    def __init__(self, attribute):
        self.attribute = attribute
        self.children = []
        self.answer = {}
```

In [72]:
```python
def compute_tree(df):
    if len(set(df[df.columns.tolist()[0]])) == 1:
        return set(df[df.columns.tolist()[0]])
    entropy, p, n = calculate_entropy(df)
    attributes = get_attributes(df)
    attribute_values = get_attributes_values(attributes, df)
    gain_of_attributes = {}
    gain = 0
    best_attribute = attributes[0]
    for attribute in attributes:
        gain_of_attributes[attribute] = entropy - calculate_information_gain(df, attribute, attribute_values[at
        gain, best_attribute = (gain_of_attributes[attribute], attribute) if gain_of_attributes[attribute] > ga
    node = Node(best_attribute)
    node.children = attribute_values[best_attribute]
    for child in node.children:
        node.answer[child] = compute_tree(df[df[best_attribute] == child])
    return node

compute_tree(df)
```

Out[72]:
```
<__main__.Node at 0x7f0e6cf57940>
```

In [74]:
```python
def print_tree(node, label):
    if node == {'Yes'} or node == {'No'} :
        return
    print(f"\t\t\t\t {str(node.attribute).upper()}({label})")
    print(f"Branches of {node.attribute}:")
    print()
    for child in node.children:
        print(child)
        if node.answer[child] != {'Yes'} and node.answer[child] != {'No'}:
            print()
        else:
            print(f'[{node.answer[child]}]')
            print()
    for child in node.children:
        if node.answer[child] != {'Yes'} and node.answer[child] != {'No'}:
            print_tree(node.answer[child], f'{str(node.attribute).upper()} --> {child}')

print_tree(compute_tree(df), "Root")
```

```
                                 OUTLOOK(Root)
Branches of Outlook:

Sunny

Overcast
[{'Yes'}]

Rain

                                 HUMIDITY(OUTLOOK --> Sunny)
Branches of Humidity:

High
[{'No'}]

Normal
[{'Yes'}]

                                 WIND(OUTLOOK --> Rain)
Branches of Wind:

Weak
[{'Yes'}]

Strong
[{'No'}]
```

Write a program to implement the naïve Bayesian classifier for a sample training data set stored as a .CSV file. Compute the accuracy of the classifier, considering few test data sets

In [16]:
```
import pandas as pd

data = pd.read_csv('PlayTennis.csv')
data.head()
```

Out[16]:

	PlayTennis	Outlook	Temperature	Humidity	Wind
0	No	Sunny	Hot	High	Weak
1	No	Sunny	Hot	High	Strong
2	Yes	Overcast	Hot	High	Weak
3	Yes	Rain	Mild	High	Weak
4	Yes	Rain	Cool	Normal	Weak

In [2]:
```
y = list(data['PlayTennis'].values)
X = data.iloc[:,1:].values

print(f'Target Values: {y}')
print(f'Features: \n{X}')
```

```
Target Values: ['No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No']
Features:
[['Sunny' 'Hot' 'High' 'Weak']
 ['Sunny' 'Hot' 'High' 'Strong']
 ['Overcast' 'Hot' 'High' 'Weak']
 ['Rain' 'Mild' 'High' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Cool' 'Normal' 'Strong']
 ['Overcast' 'Cool' 'Normal' 'Strong']
 ['Sunny' 'Mild' 'High' 'Weak']
 ['Sunny' 'Cool' 'Normal' 'Weak']
 ['Rain' 'Mild' 'Normal' 'Weak']
 ['Sunny' 'Mild' 'Normal' 'Strong']
 ['Overcast' 'Mild' 'High' 'Strong']
 ['Overcast' 'Hot' 'Normal' 'Weak']
 ['Rain' 'Mild' 'High' 'Strong']]
```

In [9]:
```
y_train = y[:11]
y_val = y[11:]

X_train = X[:11]
X_val = X[11:]

print(f"Number of instances in training set: {len(X_train)}")
print(f"Number of instances in testing set: {len(X_val)}")
```

```
Number of instances in training set: 11
Number of instances in testing set: 3
```

In [10]:
```
class NaiveBayesClassifier:


    def __init__(self, X, y):

        self.X, self.y = X, y

        self.N = len(self.X)

        self.dim = len(self.X[0])

        self.attrs = [[] for _ in range(self.dim)]

        self.output_dom = {}

        self.data = []

        for i in range(len(self.X)):
            for j in range(self.dim):
                if not self.X[i][j] in self.attrs[j]:
                    self.attrs[j].append(self.X[i][j])

            if not self.y[i] in self.output_dom.keys():
                self.output_dom[self.y[i]] = 1

            else:
                self.output_dom[self.y[i]] += 1

            self.data.append([self.X[i], self.y[i]])


    def classify(self, entry):

        solve = None
        max_arg = -1

        for y in self.output_dom.keys():

            prob = self.output_dom[y]/self.N

            for i in range(self.dim):
                cases = [x for x in self.data if x[0][i] == entry[i] and x[1] == y]
                n = len(cases)
                prob *= n/self.N

            if prob > max_arg:
                max_arg = prob
                solve = y

        return solve



n = NaiveBayesClassifier(X_train, y_train)
n.output_dom
```

Out[10]:
```
{'No': 4, 'Yes': 7}
```

In [11]:
```
nbc = NaiveBayesClassifier(X_train, y_train)

total_cases = len(y_val)

good = 0
bad = 0
predictions = []

for i in range(total_cases):
    predict = nbc.classify(X_val[i])
    predictions.append(predict)

    if y_val[i] == predict:
        good += 1
    else:
        bad += 1

print('Predicted values:', predictions)
print('Actual values:', y_val)
print()
print('Total number of testing instances in the dataset:', total_cases)
print('Number of correct predictions:', good)
print('Number of wrong predictions:', bad)
print()
print('Accuracy of Bayes Classifier:', good/total_cases)
```

```
Predicted values: ['Yes', 'Yes', 'Yes']
Actual values: ['Yes', 'Yes', 'No']

Total number of testing instances in the dataset: 3
Number of correct predictions: 2
Number of wrong predictions: 1

Accuracy of Bayes Classifier: 0.6666666666666666
```

In [ ]:
```
```

Write a program to construct a Bayesian network considering training data. Use this model to make predictions.

In [2]:
```
import numpy as np
import pandas as pd
import csv
from pgmpy.estimators import MaximumLikelihoodEstimator
from pgmpy.models import BayesianModel
from pgmpy.inference import VariableElimination
```

In [13]:
```
heartDisease = pd.read_csv('heart.csv')
heartDisease.head(10)
```

Out[13]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
0	63	1	1	145	233	1	2	150	0	2.3	3	0	6	0
1	67	1	4	160	286	0	2	108	1	1.5	2	3	3	2
2	67	1	4	120	229	0	2	129	1	2.6	2	2	7	1
3	37	1	3	130	250	0	0	187	0	3.5	3	0	3	0
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
5	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
6	62	0	4	140	268	0	2	160	0	3.6	3	2	3	3
7	57	0	4	120	354	0	0	163	1	0.6	1	0	3	0
8	63	1	4	130	254	0	2	147	0	1.4	2	1	7	2
9	53	1	4	140	203	1	2	155	1	3.1	3	0	7	1

In [5]:
```
print('Attributes and datatypes')
heartDisease.dtypes
```

```
Attributes and datatypes
```

Out[5]:
```
age               int64
sex               int64
cp                int64
trestbps          int64
chol              int64
fbs               int64
restecg           int64
thalach           int64
exang             int64
oldpeak         float64
slope             int64
ca                int64
thal              int64
heartdisease      int64
dtype: object
```

In [6]:
```
heartDisease.describe()
```

Out[6]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope
count	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000	303.000000
mean	54.438944	0.679868	3.158416	131.689769	246.693069	0.148515	0.990099	149.607261	0.326733	1.039604	1.600660
std	9.038662	0.467299	0.960126	17.599748	51.776918	0.356198	0.994971	22.875003	0.469794	1.161075	0.616226
min	29.000000	0.000000	1.000000	94.000000	126.000000	0.000000	0.000000	71.000000	0.000000	0.000000	1.000000
25%	48.000000	0.000000	3.000000	120.000000	211.000000	0.000000	0.000000	133.500000	0.000000	0.000000	1.000000
50%	56.000000	1.000000	3.000000	130.000000	241.000000	0.000000	1.000000	153.000000	0.000000	0.800000	2.000000
75%	61.000000	1.000000	4.000000	140.000000	275.000000	0.000000	2.000000	166.000000	1.000000	1.600000	2.000000
max	77.000000	1.000000	4.000000	200.000000	564.000000	1.000000	2.000000	202.000000	1.000000	6.200000	3.000000

In [7]:
```
model = BayesianModel([('age','heartdisease'),('sex','heartdisease'),('exang','heartdisease'),('cp','heartdisea
print('Learning CPD using Maximum likelihood estimators')
```

```
Learning CPD using Maximum likelihood estimators
```

In [8]:
```
model.fit(heartDisease, estimator=MaximumLikelihoodEstimator)
```

In [9]:
```
print('Inferencing with Bayesian Network:')
HeartDiseasetest_infer = VariableElimination(model)
```

```
Inferencing with Bayesian Network:
```

In [24]:
```
print('1. Probability of HeartDisease given evidence = restecg : 1')
q1 = HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'restecg':1})
print(q1)
```

```
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: cp:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex: 100%|██████████| 5/5 [00:00<00:00, 155.84it/s]
1. Probability of HeartDisease given evidence = restecg : 1
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.1016 |
+-----------------+---------------------+
| heartdisease(1) |              0.0000 |
+-----------------+---------------------+
| heartdisease(2) |              0.2361 |
+-----------------+---------------------+
| heartdisease(3) |              0.2017 |
+-----------------+---------------------+
| heartdisease(4) |              0.4605 |
+-----------------+---------------------+
```

In [18]:
```
print('Tuples with \'restecg = 1\' in the database are:')
heartDisease[heartDisease['restecg'] == 1]
```

```
Tuples with 'restecg = 1' in the database are:
```

Out[18]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
231	55	0	4	180	327	0	1	117	1	3.4	2	0	3	2
257	76	0	3	140	197	0	1	116	0	1.1	2	0	3	0
282	55	0	4	128	205	0	1	130	1	2.0	2	1	7	3
285	58	1	4	114	318	0	1	140	0	4.4	3	3	6	4

In [20]:
```
print('2. Probability of HeartDisease given evidence = cp : 2')
q2=HeartDiseasetest_infer.query(variables=['heartdisease'],evidence={'cp':2})
print(q2)
```

```
Finding Elimination Order: : 100%|██████████| 5/5 [00:37<00:00,  7.56s/it]
Finding Elimination Order: :   0%|          | 0/5 [00:00<?, ?it/s]
  0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: chol:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: restecg:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: age:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: exang:   0%|          | 0/5 [00:00<?, ?it/s]
Eliminating: sex: 100%|██████████| 5/5 [00:00<00:00, 142.78it/s]
2. Probability of HeartDisease given evidence = cp : 2
+-----------------+---------------------+
| heartdisease    |   phi(heartdisease) |
+=================+=====================+
| heartdisease(0) |              0.3742 |
+-----------------+---------------------+
| heartdisease(1) |              0.2018 |
+-----------------+---------------------+
| heartdisease(2) |              0.1375 |
+-----------------+---------------------+
| heartdisease(3) |              0.1541 |
+-----------------+---------------------+
| heartdisease(4) |              0.1323 |
+-----------------+---------------------+
```

In [23]:
```
print('Tuples with \'cp = 2\' in the database are:')
heartDisease[heartDisease['cp'] == 2].head(10)
```

```
Tuples with 'cp = 2' in the database are:
```

Out[23]:

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heartdisease
4	41	0	2	130	204	0	2	172	0	1.4	1	0	3	0
5	56	1	2	120	236	0	0	178	0	0.8	1	0	3	0
11	56	0	2	140	294	0	2	153	0	1.3	2	0	3	0
13	44	1	2	120	263	0	0	173	0	0.0	1	0	7	0
16	48	1	2	110	229	0	0	168	0	1.0	3	0	7	1
19	49	1	2	130	266	0	0	171	0	0.6	1	0	3	0
22	58	1	2	120	284	0	2	160	0	1.8	2	0	3	1
42	71	0	2	160	302	0	0	162	0	0.4	1	2	3	0
50	41	0	2	105	198	0	0	168	0	0.0	1	1	3	0
53	44	1	2	130	219	0	2	188	0	0.0	1	0	3	0

In [2]:
```python
from pgmpy.models import BayesianModel
from pgmpy.factors.discrete import TabularCPD
from pgmpy.inference import VariableElimination
```

In [3]:
```python
cancer_model = BayesianModel([('Pollution','Cancer'),
                              ('Smoker','Cancer'),
                              ('Cancer','Xray'),
                              ('Cancer','Dyspnoea')])

print('Bayesian network nodes are:')
print("\t",cancer_model.nodes())
print()
print('Bayesian network edges are:')
print("\t",cancer_model.edges())

cpd_poll = TabularCPD(variable='Pollution',variable_card=2,values=[[0.9],[0.1]])
cpd_smoke = TabularCPD(variable='Smoker',variable_card=2,values=[[0.3],[0.7]])
cpd_cancer = TabularCPD(variable='Cancer',variable_card=2,values=[[0.03,0.05,0.001,0.02], [0.97,0.95,0.999,0.98
                       evidence=['Smoker','Pollution'],evidence_card=[2,2])
cpd_xray = TabularCPD(variable='Xray',variable_card=2,values=[[0.9,0.2],[0.1,0.8]],
                     evidence=['Cancer'],evidence_card=[2])
cpd_dysp = TabularCPD(variable='Dyspnoea',variable_card=2,values=[[0.65,0.3],[0.35,0.7]],
                     evidence=['Cancer'],evidence_card=[2])
```

```
Bayesian network nodes are:
	 ['Pollution', 'Cancer', 'Smoker', 'Xray', 'Dyspnoea']

Bayesian network edges are:
	 [('Pollution', 'Cancer'), ('Cancer', 'Xray'), ('Cancer', 'Dyspnoea'), ('Smoker', 'Cancer')]
```

In [4]:
```python
cancer_model.add_cpds(cpd_poll,cpd_smoke,cpd_cancer,cpd_xray,cpd_dysp)
print('Model generated by adding cpts(cpds)')
print('Checking correctness of model:',end=' ')
print(cancer_model.check_model())
```

```
Model generated by adding cpts(cpds)
Checking correctness of model: True
```

In [5]:
```python
print('All local depencies are as follows: ')
cancer_model.get_independencies()
```

```
All local depencies are as follows:
```

Out[5]:
```
(Pollution ⊥ Smoker)
(Pollution ⊥ Dyspnoea, Xray | Cancer)
(Pollution ⊥ Xray | Dyspnoea, Cancer)
(Pollution ⊥ Dyspnoea | Xray, Cancer)
(Pollution ⊥ Dyspnoea, Xray | Smoker, Cancer)
(Pollution ⊥ Xray | Dyspnoea, Smoker, Cancer)
(Pollution ⊥ Dyspnoea | Xray, Smoker, Cancer)
(Smoker ⊥ Pollution)
(Smoker ⊥ Dyspnoea, Xray | Cancer)
(Smoker ⊥ Xray | Dyspnoea, Cancer)
(Smoker ⊥ Dyspnoea | Xray, Cancer)
(Smoker ⊥ Dyspnoea, Xray | Pollution, Cancer)
(Smoker ⊥ Xray | Dyspnoea, Pollution, Cancer)
(Smoker ⊥ Dyspnoea | Xray, Pollution, Cancer)
(Xray ⊥ Dyspnoea, Smoker, Pollution | Cancer)
(Xray ⊥ Smoker, Pollution | Dyspnoea, Cancer)
(Xray ⊥ Dyspnoea, Pollution | Smoker, Cancer)
(Xray ⊥ Dyspnoea, Smoker | Pollution, Cancer)
(Xray ⊥ Pollution | Dyspnoea, Smoker, Cancer)
(Xray ⊥ Smoker | Dyspnoea, Pollution, Cancer)
(Xray ⊥ Dyspnoea | Smoker, Pollution, Cancer)
(Dyspnoea ⊥ Xray, Smoker, Pollution | Cancer)
(Dyspnoea ⊥ Smoker, Pollution | Xray, Cancer)
(Dyspnoea ⊥ Xray, Pollution | Smoker, Cancer)
(Dyspnoea ⊥ Xray, Smoker | Pollution, Cancer)
(Dyspnoea ⊥ Pollution | Xray, Smoker, Cancer)
(Dyspnoea ⊥ Smoker | Xray, Pollution, Cancer)
(Dyspnoea ⊥ Xray | Smoker, Pollution, Cancer)
```

In [6]:
```python
print('Displaying  CPDs')
print()
print(cancer_model.get_cpds('Pollution'))
print()
print(cancer_model.get_cpds('Smoker'))
print()
print(cancer_model.get_cpds('Cancer'))
print()
print(cancer_model.get_cpds('Xray'))
print()
print(cancer_model.get_cpds('Dyspnoea'))
```

```
Displaying  CPDs

+--------------+-----+
| Pollution(0) | 0.9 |
+--------------+-----+
| Pollution(1) | 0.1 |
+--------------+-----+

+-----------+-----+
| Smoker(0) | 0.3 |
+-----------+-----+
| Smoker(1) | 0.7 |
+-----------+-----+

+-----------+--------------+--------------+--------------+--------------+
| Smoker    | Smoker(0)    | Smoker(0)    | Smoker(1)    | Smoker(1)    |
+-----------+--------------+--------------+--------------+--------------+
| Pollution | Pollution(0) | Pollution(1) | Pollution(0) | Pollution(1) |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(0) | 0.03         | 0.05         | 0.001        | 0.02         |
+-----------+--------------+--------------+--------------+--------------+
| Cancer(1) | 0.97         | 0.95         | 0.999        | 0.98         |
+-----------+--------------+--------------+--------------+--------------+

+---------+-----------+-----------+
| Cancer  | Cancer(0) | Cancer(1) |
+---------+-----------+-----------+
| Xray(0) | 0.9       | 0.2       |
+---------+-----------+-----------+
| Xray(1) | 0.1       | 0.8       |
+---------+-----------+-----------+

+-------------+-----------+-----------+
| Cancer      | Cancer(0) | Cancer(1) |
+-------------+-----------+-----------+
| Dyspnoea(0) | 0.65      | 0.3       |
+-------------+-----------+-----------+
| Dyspnoea(1) | 0.35      | 0.7       |
+-------------+-----------+-----------+
```

In [7]:
```python
cancer_infer=VariableElimination(cancer_model)
print('\nInferencing with bayesian network')
print("\n\nProbability of Cancer given smoker")
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1})
print(q)

print("\nProbability of Cancer given smoker and pollution")
q=cancer_infer.query(variables=['Cancer'],evidence={'Smoker':1,'Pollution':1})
print(q)
```

```
Finding Elimination Order: :   0%|          | 0/3 [00:00<?, ?it/s]
  0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Dyspnoea:   0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Xray:   0%|          | 0/3 [00:00<?, ?it/s]
Eliminating: Pollution: 100%|██████████| 3/3 [00:00<00:00, 123.35it/s]

  0%|          | 0/2 [00:00<?, ?it/s]
Finding Elimination Order: :   0%|          | 0/2 [00:00<?, ?it/s]

  0%|          | 0/2 [00:00<?, ?it/s]

Eliminating: Dyspnoea:   0%|          | 0/2 [00:00<?, ?it/s]

Eliminating: Xray: 100%|██████████| 2/2 [00:00<00:00, 212.53it/s]A
```

```
Inferencing with bayesian network


Probability of Cancer given smoker
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0029 |
+-----------+---------------+
| Cancer(1) |        0.9971 |
+-----------+---------------+

Probability of Cancer given smoker and pollution
+-----------+---------------+
| Cancer    |   phi(Cancer) |
+===========+===============+
| Cancer(0) |        0.0200 |
+-----------+---------------+
| Cancer(1) |        0.9800 |
+-----------+---------------+
```

Apply k-Means algorithm to cluster a set of data stored in a .CSV file.

In [16]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler
from matplotlib import pyplot as plt
%matplotlib inline
```

In [17]:
```
df = pd.read_csv('income.csv')
df.head()
```

Out[17]:

	Name	Age	Income($)
0	Rob	27	70000
1	Michael	29	90000
2	Mohan	29	61000
3	Ismail	28	60000
4	Kory	42	150000

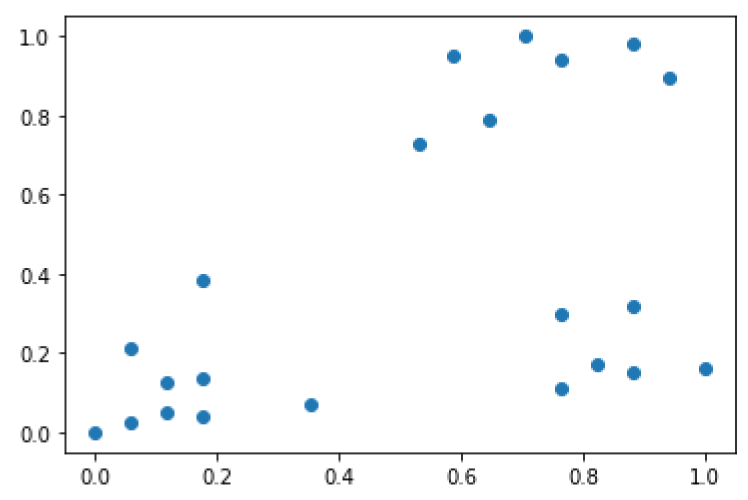
In [18]:
```
scaler = MinMaxScaler()
scaler.fit(df[['Age']])
df[['Age']] = scaler.transform(df[['Age']])

scaler.fit(df[['Income($)']])
df[['Income($)']] = scaler.transform(df[['Income($)']])
df.head(5)
```

Out[18]:

	Name	Age	Income($)
0	Rob	0.058824	0.213675
1	Michael	0.176471	0.384615
2	Mohan	0.176471	0.136752
3	Ismail	0.117647	0.128205
4	Kory	0.941176	0.897436

In [19]:
```
plt.scatter(df['Age'], df['Income($)'])
```

Out[19]:
```
<matplotlib.collections.PathCollection at 0x7f370544ef10>
```



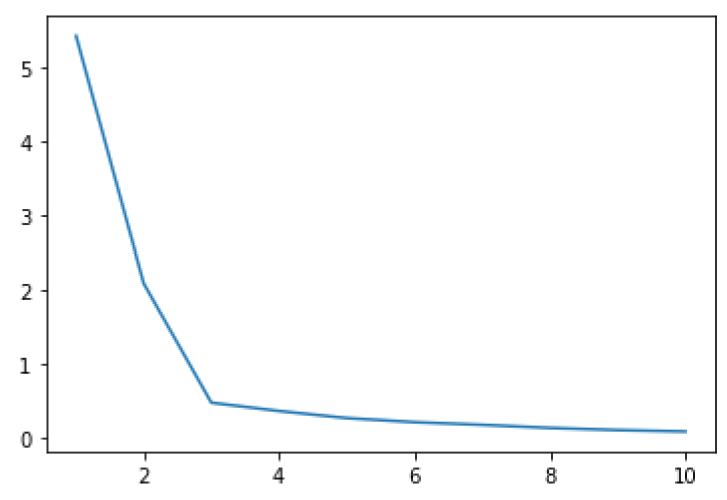
Finding Elbow Point

In [20]:
```
k_range = range(1, 11)
sse = []
for k in k_range:
    kmc = KMeans(n_clusters=k)
    kmc.fit(df[['Age', 'Income($)']])
    sse.append(kmc.inertia_)
sse
```

Out[20]:
```
[5.434011511988179,
 2.091136388699078,
 0.4750783498553097,
 0.3625079900797329,
 0.2664030124668416,
 0.21066678488010523,
 0.17462386586687895,
 0.13265419827245162,
 0.10383752586603562,
 0.08467306738090172]
```

In [21]:
```
plt.xlabel = 'Number of Clusters'
plt.ylabel = 'Sum of Squared Errors'
plt.plot(k_range, sse)
```

Out[21]:
```
[<matplotlib.lines.Line2D at 0x7f370542f580>]
```



Therefore, the elbow point is 3

In [22]:
```
km = KMeans(n_clusters=3)
km
```

Out[22]:
```
KMeans(n_clusters=3)
```

In [23]:
```
y_predict = km.fit_predict(df[['Age', 'Income($)']])
y_predict
```

Out[23]:
```
array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2],
      dtype=int32)
```

In [24]:
```
df['cluster'] = y_predict
df.head()
```

Out[24]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
4	Kory	0.941176	0.897436	1

In [25]:
```
df0 = df[df.cluster == 0]
df0
```

Out[25]:

	Name	Age	Income($)	cluster
0	Rob	0.058824	0.213675	0
1	Michael	0.176471	0.384615	0
2	Mohan	0.176471	0.136752	0
3	Ismail	0.117647	0.128205	0
11	Tom	0.000000	0.000000	0
12	Arnold	0.058824	0.025641	0
13	Jared	0.117647	0.051282	0
14	Stark	0.176471	0.038462	0
15	Ranbir	0.352941	0.068376	0

In [26]:
```
df1 = df[df.cluster == 1]
df1
```

Out[26]:

	Name	Age	Income($)	cluster
4	Kory	0.941176	0.897436	1
5	Gautam	0.764706	0.940171	1
6	David	0.882353	0.982906	1
7	Andrea	0.705882	1.000000	1
8	Brad	0.588235	0.948718	1
9	Angelina	0.529412	0.726496	1
10	Donald	0.647059	0.786325	1

In [27]:
```
df2 = df[df.cluster == 2]
df2
```

Out[27]:

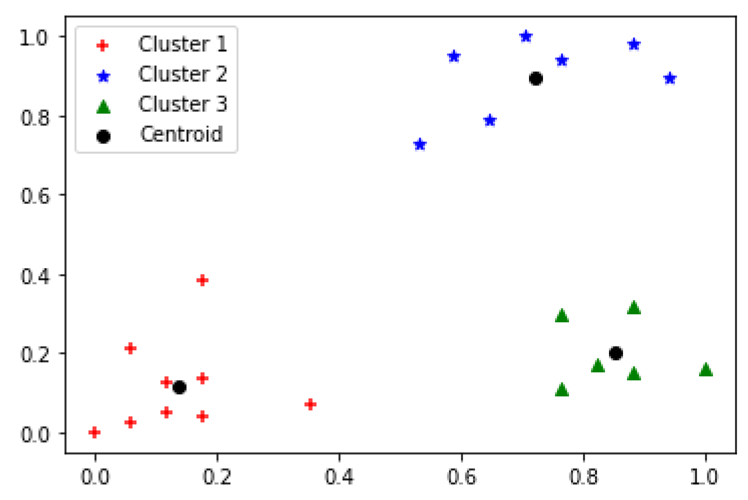
	Name	Age	Income($)	cluster
16	Dipika	0.823529	0.170940	2
17	Priyanka	0.882353	0.153846	2
18	Nick	1.000000	0.162393	2
19	Alia	0.764706	0.299145	2
20	Sid	0.882353	0.316239	2
21	Abdul	0.764706	0.111111	2

In [28]:
```
km.cluster_centers_
```

Out[28]:
```
array([[0.1372549 , 0.11633428],
       [0.72268908, 0.8974359 ],
       [0.85294118, 0.2022792 ]])
```

In [35]:
```
p1 = plt.scatter(df0['Age'], df0['Income($)'], marker='+', color='red')
p2 = plt.scatter(df1['Age'], df1['Income($)'], marker='*', color='blue')
p3 = plt.scatter(df2['Age'], df2['Income($)'], marker='^', color='green')
c = plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:,1], color='black')
plt.legend((p1, p2, p3, c),
           ('Cluster 1', 'Cluster 2', 'Cluster 3', 'Centroid'))
```

Out[35]:
```
<matplotlib.legend.Legend at 0x7f37042f1ca0>
```



In [ ]:

K-Means clustering without builtins

In [1]:
```python
import math;
import sys;
import pandas as pd
import numpy as np
from random import choice
from matplotlib import pyplot
from random import shuffle, uniform;
```

In [2]:
```python
def ReadData(fileName):
    f = open(fileName,'r')
    lines = f.read().splitlines()
    f.close()

    items = []

    for i in range(1,len(lines)):
        line = lines[i].split(',')
        itemFeatures = []

        for j in range(len(line)-1):
            v = float(line[j])
            itemFeatures.append(v)
        items.append(itemFeatures)

    shuffle(items)

    return items
```

In [3]:
```python
def FindColMinMax(items):
    n = len(items[0])
    minima = [float('inf') for i in range(n)]
    maxima = [float('-inf') -1 for i in range(n)]

    for item in items:
        for f in range(len(item)):
            if(item[f] < minima[f]):
                minima[f] = item[f]

            if(item[f] > maxima[f]):
                maxima[f] = item[f]

    return minima,maxima
```

In [12]:
```python
def EuclideanDistance(x,y):
    S = 0
    for i in range(len(x)):
        S += math.pow(x[i]-y[i],2)

    return math.sqrt(S)
```

In [4]:
```python
def InitializeMeans(items,k,cMin,cMax):
    f = len(items[0])
    means = [[0 for i in range(f)] for j in range(k)]

    for mean in means:
        for i in range(len(mean)):
            mean[i] = uniform(cMin[i]+1,cMax[i]-1)

    return means
```

In [5]:
```python
def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i]
        m = (m*(n-1)+item[i])/float(n)
        mean[i] = round(m,3)

    return mean
```

In [6]:
```python
def FindClusters(means,items):
    clusters = [[] for i in range(len(means))]

    for item in items:
        index = Classify(means,item)
        clusters[index].append(item)

    return clusters
```

In [7]:
```python
def Classify(means,item):

    minimum = float('inf');
    index = -1

    for i in range(len(means)):
        dis = EuclideanDistance(item,means[i])

        if(dis < minimum):
            minimum = dis
            index = i

    return index
```

In [8]:
```python
def CalculateMeans(k,items,maxIterations=100000):
    cMin, cMax = FindColMinMax(items)

    means = InitializeMeans(items,k,cMin,cMax)

    clusterSizes = [0 for i in range(len(means))]

    belongsTo = [0 for i in range(len(items))]

    for e in range(maxIterations):
        noChange = True;
        for i in range(len(items)):
            item = items[i];
            index = Classify(means,item)
            clusterSizes[index] += 1
            cSize = clusterSizes[index]
            means[index] = UpdateMean(cSize,means[index],item)

            if(index != belongsTo[i]):
                noChange = False
            belongsTo[i] = index

        if (noChange):
            break

    return means
```

In [9]:
```python
def CutToTwoFeatures(items,indexA,indexB):
    n = len(items)
    X = []
    for i in range(n):
        item = items[i]
        newItem = [item[indexA],item[indexB]]
        X.append(newItem)

    return X
```

In [10]:
```python
def PlotClusters(clusters):
    n = len(clusters)
    X = [[] for i in range(n)]

    for i in range(n):
        cluster = clusters[i]
        for item in cluster:
            X[i].append(item)

    colors = ['r','b','g','c','m','y']

    for x in X:
        c = choice(colors)
        colors.remove(c)

        Xa = []
        Xb = []

        for item in x:
            Xa.append(item[0])
            Xb.append(item[1])

        pyplot.plot(Xa,Xb,'o',color=c)

    pyplot.show()
```

In [16]:
```python
items = ReadData('data.txt')
k = 3
items = CutToTwoFeatures(items,2,3)
print(items)
means = CalculateMeans(k,items)
print("\nMeans = ", means)

clusters = FindClusters(means,items)

#PlotClusters(clusters)
newItem = [1.5,0.2]
print(Classify(means,newItem))
```

```
[[6.7, 2.0], [4.7, 1.2], [5.1, 1.9], [5.8, 1.8], [4.0, 1.3], [4.8, 1.4], [5.9, 2.3], [1.4, 0.2], [4.4, 1.4],
[5.5, 2.1], [5.1, 1.8], [1.2, 0.2], [5.0, 2.0], [1.0, 0.2], [5.0, 1.5], [1.4, 0.2], [4.8, 1.8], [1.9, 0.2], [4.
0, 1.0], [4.9, 1.5], [6.1, 1.9], [5.2, 2.3], [5.1, 1.9], [1.5, 0.2], [5.6, 2.1], [5.3, 2.3], [1.5, 0.1], [4.8,
1.8], [4.8, 1.8], [1.6, 0.2], [5.0, 1.7], [5.0, 1.9], [4.2, 1.5], [3.3, 1.0], [4.7, 1.6], [4.7, 1.5], [4.1, 1.
3], [4.0, 1.3], [1.5, 0.2], [1.6, 0.6], [6.6, 2.1], [1.5, 0.4], [4.5, 1.5], [5.9, 2.1], [1.4, 0.2], [5.8, 2.2],
[4.1, 1.0], [4.6, 1.5], [5.2, 2.0], [3.9, 1.2], [1.4, 0.2], [4.5, 1.5], [3.5, 1.0], [6.1, 2.5], [5.6, 1.8], [5.
6, 2.4], [5.7, 2.3], [1.5, 0.1], [1.5, 0.3], [1.3, 0.3], [1.5, 0.4], [1.3, 0.2], [6.4, 2.0], [5.1, 2.3], [3.3,
1.0], [4.9, 2.0], [6.7, 2.2], [1.1, 0.1], [1.5, 0.2], [4.0, 1.3], [1.9, 0.4], [1.3, 0.2], [1.6, 0.2], [1.4, 0.
3], [4.4, 1.2], [1.5, 0.4], [1.3, 0.2], [3.7, 1.0], [4.3, 1.3], [4.9, 1.5], [3.5, 1.0], [4.4, 1.3], [4.1, 1.3],
[1.4, 0.1], [4.0, 1.2], [1.5, 0.2], [1.3, 0.4], [5.6, 2.2], [3.8, 1.1], [1.5, 0.2], [1.5, 0.1], [4.5, 1.5], [4.
9, 1.8], [5.7, 2.1], [5.6, 1.4], [5.4, 2.3], [4.5, 1.7], [5.1, 2.0], [3.6, 1.3], [4.2, 1.2], [1.3, 0.2], [1.2,
0.2], [1.6, 0.2], [1.6, 0.4], [5.1, 2.4], [4.7, 1.4], [1.7, 0.3], [3.0, 1.1], [5.6, 2.4], [1.5, 0.2], [1.6, 0.
2], [5.1, 1.6], [4.7, 1.4], [4.5, 1.6], [4.6, 1.4], [1.4, 0.3], [3.9, 1.1], [5.5, 1.8], [1.4, 0.2], [5.8, 1.6],
[4.5, 1.3], [5.5, 1.8], [5.4, 2.1], [6.0, 2.5], [4.5, 1.5], [1.7, 0.5], [3.9, 1.4], [1.5, 0.1], [6.1, 2.3], [5.
7, 2.5], [4.3, 1.3], [1.3, 0.3], [1.7, 0.4], [6.3, 1.8], [4.4, 1.4], [6.0, 1.8], [1.6, 0.2], [4.2, 1.3], [1.4,
0.2], [1.4, 0.2], [6.9, 2.3], [4.5, 1.5], [1.4, 0.3], [5.1, 1.5], [5.3, 1.9], [4.9, 1.8], [4.6, 1.3], [4.2, 1.
3], [1.7, 0.2]]

Means =  [[4.352, 1.388], [5.689, 2.082], [1.47, 0.261]]
2
```

Apply EM algorithm to cluster a set of data stored in a .CSV file. Compare the results of k-Means algorithm and EM algorithm.

In [1]:

```
import matplotlib.pyplot as plt
from sklearn import datasets
from sklearn.cluster import KMeans
import sklearn.metrics as sm
import pandas as pd
import numpy as np
```

In [2]:

```
iris = datasets.load_iris()

X = pd.DataFrame(iris.data)
X.columns = ['Sepal_Length','Sepal_Width','Petal_Length','Petal_Width']

y = pd.DataFrame(iris.target)
y.columns = ['Targets']

model = KMeans(n_clusters=3)
model.fit(X)
```

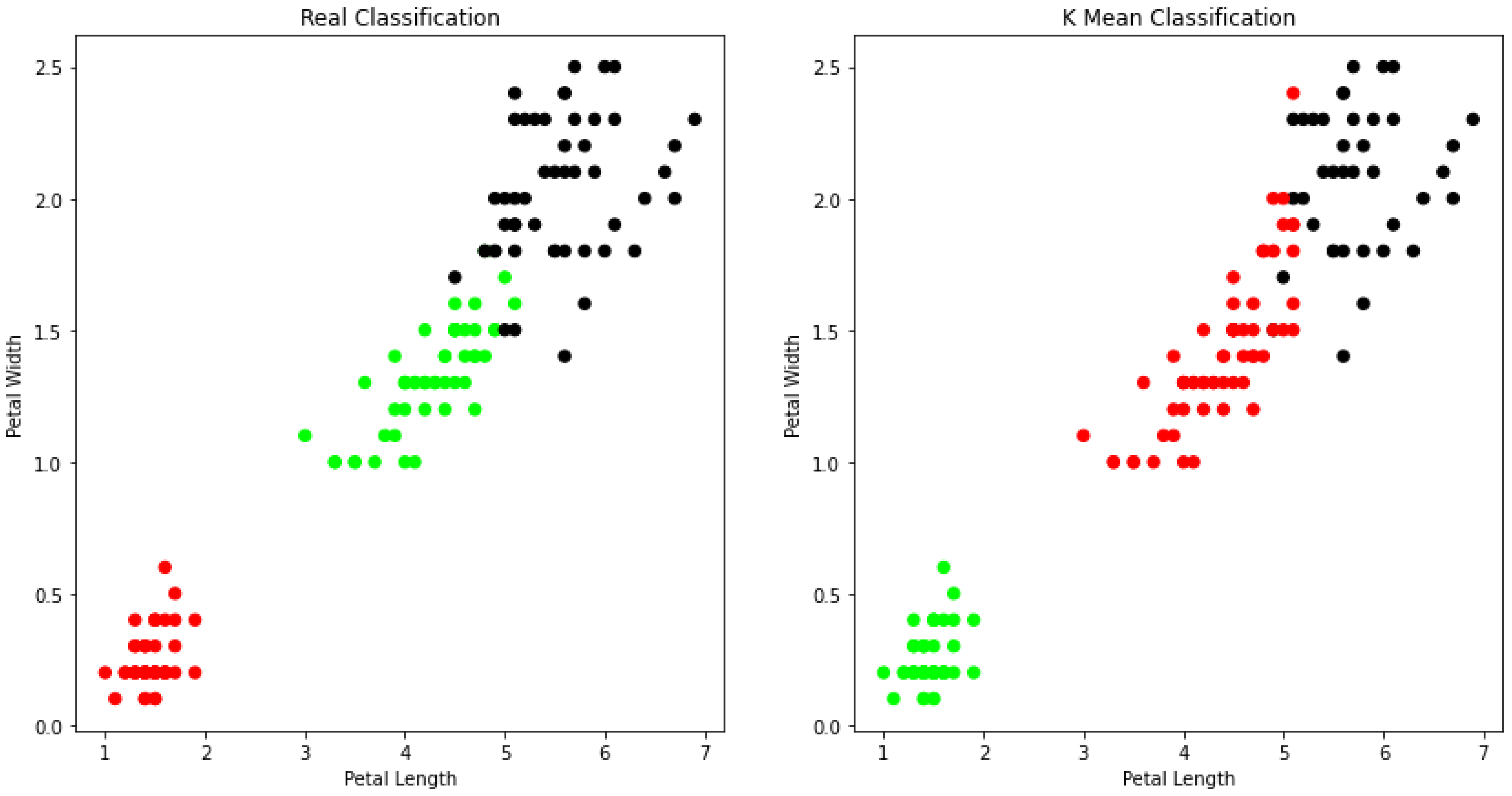
Out[2]:

```
KMeans(n_clusters=3)
```

In [3]:

```
plt.figure(figsize=(14,7))

colormap = np.array(['red', 'lime', 'black'])

# Plot the Original Classifications
plt.subplot(1, 2, 1)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y.Targets], s=40)
plt.title('Real Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')


# Plot the Models Classifications
plt.subplot(1, 2, 2)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[model.labels_], s=40)
plt.title('K Mean Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')
print('The accuracy score of K-Mean: ',sm.accuracy_score(y, model.labels_))
print('The Confusion matrixof K-Mean: ',sm.confusion_matrix(y, model.labels_))
```

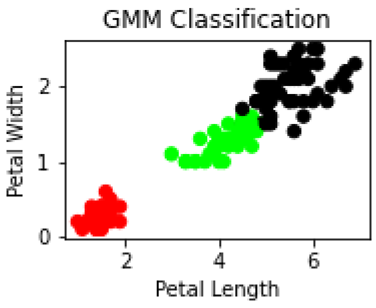
```
The accuracy score of K-Mean:  0.24
The Confusion matrixof K-Mean:  [[ 0 50  0]
 [48  0  2]
 [14  0 36]]
```



In [4]:

```
from sklearn import preprocessing
scaler = preprocessing.StandardScaler()
scaler.fit(X)
xsa = scaler.transform(X)
xs = pd.DataFrame(xsa, columns = X.columns)
#xs.sample(5)

from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3)
gmm.fit(xs)

y_gmm = gmm.predict(xs)
#y_cluster_gmm

plt.subplot(2, 2, 3)
plt.scatter(X.Petal_Length, X.Petal_Width, c=colormap[y_gmm], s=40)
plt.title('GMM Classification')
plt.xlabel('Petal Length')
plt.ylabel('Petal Width')

print('The accuracy score of EM: ',sm.accuracy_score(y, y_gmm))
print('The Confusion matrix of EM: ',sm.confusion_matrix(y, y_gmm))
```

```
The accuracy score of EM:  0.9666666666666667
The Confusion matrix of EM:  [[50  0  0]
 [ 0 45  5]
 [ 0  0 50]]
```



EM Algorithm without builtins

In [1]:
```
import numpy as np
from scipy import stats
np.random.seed(110)
```

In [2]:
```
red_mean = 3
red_std = 0.8
blue_mean = 7
blue_std = 1
```

In [3]:
```
red = np.random.normal(red_mean, red_std, size=40)
blue = np.random.normal(blue_mean, blue_std, size=40)
both_colours = np.sort(np.concatenate((red, blue)))
```

In [4]:
```
red_mean_guess = 2.1
blue_mean_guess = 6
red_std_guess = 1.5
blue_std_guess = 0.8
```

In [5]:
```
for i in range(10):
    likelihood_of_red = stats.norm(red_mean_guess, red_std_guess).pdf(both_colours)
    likelihood_of_blue = stats.norm(blue_mean_guess,blue_std_guess).pdf(both_colours)
```

In [10]:
```
likelihood_total = likelihood_of_red + likelihood_of_blue
red_weight = likelihood_of_red / likelihood_total
blue_weight = likelihood_of_blue / likelihood_total
```

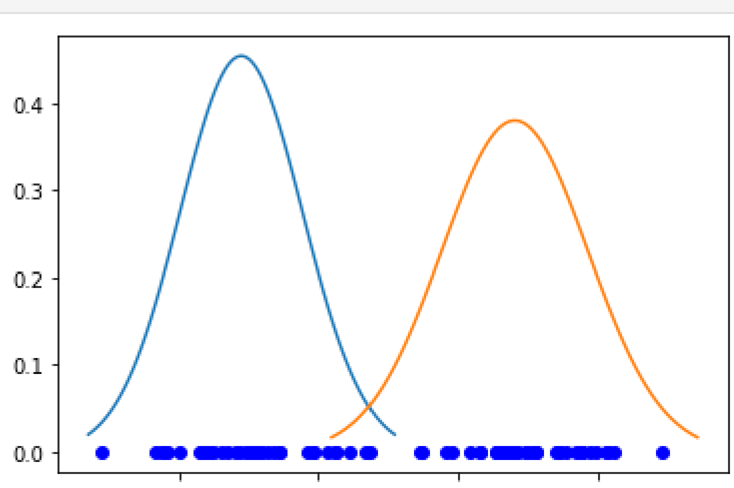
In [11]:
```
def estimate_mean(data, weight):
    return np.sum(data * weight) / np.sum(weight)
```

In [12]:
```
def estimate_std(data, weight, mean):
    variance = np.sum(weight * (data - mean)**2) / np.sum(weight)
    return np.sqrt(variance)
```

In [15]:
```
blue_std_guess = estimate_std(both_colours, blue_weight, blue_mean_guess)
red_std_guess = estimate_std(both_colours, red_weight, red_mean_guess)

red_mean_guess = estimate_mean(both_colours, red_weight)
blue_mean_guess = estimate_mean(both_colours, blue_weight)

print("red mean:", red_mean_guess, "              ", "blue mean:", blue_mean_guess)
print("red std:", red_std_guess, "                ", "blue std:", blue_std_guess)
```

```
red mean: 2.8939486098495264                 blue mean: 6.817385954777204
red std: 0.878660944654475                   blue std: 1.0501824727778526
```

In [20]:
```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib.mlab as mlab

y = np.zeros(len(both_colours))
mured = red_mean_guess
sigmared = red_std_guess
x = np.linspace(mured - 2.5*sigmared, mured + 2.5*sigmared, 100)
plt.plot(x,stats.norm.pdf(x, mured, sigmared))
mublue = blue_mean_guess
sigmablue = blue_std_guess
y = np.linspace(mublue - 2.5*sigmablue, mublue + 2.5*sigmablue, 100)
plt.plot(y,stats.norm.pdf(y, mublue, sigmablue))

for i in range(len(both_colours)):
    plt.plot(both_colours[i],0,"bo")
plt.show()
```



Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions.

In [2]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import datasets

iris=datasets.load_iris()

x = iris.data
y = iris.target

print ('sepal-length', 'sepal-width', 'petal-length', 'petal-width')
print(x)
print('class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica')
print(y)

x_train, x_test, y_train, y_test = train_test_split(x,y,test_size=0.3)

#To Training the model and Nearest nighbors K=5
classifier = KNeighborsClassifier(n_neighbors=5)
classifier.fit(x_train, y_train)
```

```
sepal-length sepal-width petal-length petal-width
[[5.1 3.5 1.4 0.2]
 [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]
 [5.4 3.9 1.7 0.4]
 [4.6 3.4 1.4 0.3]
 [5.  3.4 1.5 0.2]
 [4.4 2.9 1.4 0.2]
 [4.9 3.1 1.5 0.1]
 [5.4 3.7 1.5 0.2]
 [4.8 3.4 1.6 0.2]
 [4.8 3.  1.4 0.1]
 [4.3 3.  1.1 0.1]
 [5.8 4.  1.2 0.2]
 [5.7 4.4 1.5 0.4]
 [5.4 3.9 1.3 0.4]
 [5.1 3.5 1.4 0.3]
 [5.7 3.8 1.7 0.3]
 [5.1 3.8 1.5 0.3]
 [5.4 3.4 1.7 0.2]
 [5.1 3.7 1.5 0.4]
 [4.6 3.6 1.  0.2]
 [5.1 3.3 1.7 0.5]
 [4.8 3.4 1.9 0.2]
 [5.  3.  1.6 0.2]
 [5.  3.4 1.6 0.4]
 [5.2 3.5 1.5 0.2]
 [5.2 3.4 1.4 0.2]
 [4.7 3.2 1.6 0.2]
 [4.8 3.1 1.6 0.2]
 [5.4 3.4 1.5 0.4]
 [5.2 4.1 1.5 0.1]
 [5.5 4.2 1.4 0.2]
 [4.9 3.1 1.5 0.2]
 [5.  3.2 1.2 0.2]
 [5.5 3.5 1.3 0.2]
 [4.9 3.6 1.4 0.1]
 [4.4 3.  1.3 0.2]
 [5.1 3.4 1.5 0.2]
 [5.  3.5 1.3 0.3]
 [4.5 2.3 1.3 0.3]
 [4.4 3.2 1.3 0.2]
 [5.  3.5 1.6 0.6]
 [5.1 3.8 1.9 0.4]
 [4.8 3.  1.4 0.3]
 [5.1 3.8 1.6 0.2]
 [4.6 3.2 1.4 0.2]
 [5.3 3.7 1.5 0.2]
 [5.  3.3 1.4 0.2]
 [7.  3.2 4.7 1.4]
 [6.4 3.2 4.5 1.5]
 [6.9 3.1 4.9 1.5]
 [5.5 2.3 4.  1.3]
 [6.5 2.8 4.6 1.5]
 [5.7 2.8 4.5 1.3]
 [6.3 3.3 4.7 1.6]
 [4.9 2.4 3.3 1. ]
 [6.6 2.9 4.6 1.3]
 [5.2 2.7 3.9 1.4]
 [5.  2.  3.5 1. ]
 [5.9 3.  4.2 1.5]
 [6.  2.2 4.  1. ]
 [6.1 2.9 4.7 1.4]
 [5.6 2.9 3.6 1.3]
 [6.7 3.1 4.4 1.4]
 [5.6 3.  4.5 1.5]
 [5.8 2.7 4.1 1. ]
 [6.2 2.2 4.5 1.5]
 [5.6 2.5 3.9 1.1]
 [5.9 3.2 4.8 1.8]
 [6.1 2.8 4.  1.3]
 [6.3 2.5 4.9 1.5]
 [6.1 2.8 4.7 1.2]
 [6.4 2.9 4.3 1.3]
 [6.6 3.  4.4 1.4]
 [6.8 2.8 4.8 1.4]
 [6.7 3.  5.  1.7]
 [6.  2.9 4.5 1.5]
 [5.7 2.6 3.5 1. ]
 [5.5 2.4 3.8 1.1]
 [5.5 2.4 3.7 1. ]
 [5.8 2.7 3.9 1.2]
 [6.  2.7 5.1 1.6]
 [5.4 3.  4.5 1.5]
 [6.  3.4 4.5 1.6]
 [6.7 3.1 4.7 1.5]
 [6.3 2.3 4.4 1.3]
 [5.6 3.  4.1 1.3]
 [5.5 2.5 4.  1.3]
 [5.5 2.6 4.4 1.2]
 [6.1 3.  4.6 1.4]
 [5.8 2.6 4.  1.2]
 [5.  2.3 3.3 1. ]
 [5.6 2.7 4.2 1.3]
 [5.7 3.  4.2 1.2]
 [5.7 2.9 4.2 1.3]
 [6.2 2.9 4.3 1.3]
 [5.1 2.5 3.  1.1]
 [5.7 2.8 4.1 1.3]
 [6.3 3.3 6.  2.5]
 [5.8 2.7 5.1 1.9]
 [7.1 3.  5.9 2.1]
 [6.3 2.9 5.6 1.8]
 [6.5 3.  5.8 2.2]
 [7.6 3.  6.6 2.1]
 [4.9 2.5 4.5 1.7]
 [7.3 2.9 6.3 1.8]
 [6.7 2.5 5.8 1.8]
 [7.2 3.6 6.1 2.5]
 [6.5 3.2 5.1 2. ]
 [6.4 2.7 5.3 1.9]
 [6.8 3.  5.5 2.1]
 [5.7 2.5 5.  2. ]
 [5.8 2.8 5.1 2.4]
 [6.4 3.2 5.3 2.3]
 [6.5 3.  5.5 1.8]
 [7.7 3.8 6.7 2.2]
 [7.7 2.6 6.9 2.3]
 [6.  2.2 5.  1.5]
 [6.9 3.2 5.7 2.3]
 [5.6 2.8 4.9 2. ]
 [7.7 2.8 6.7 2. ]
 [6.3 2.7 4.9 1.8]
 [6.7 3.3 5.7 2.1]
 [7.2 3.2 6.  1.8]
 [6.2 2.8 4.8 1.8]
 [6.1 3.  4.9 1.8]
 [6.4 2.8 5.6 2.1]
 [7.2 3.  5.8 1.6]
 [7.4 2.8 6.1 1.9]
 [7.9 3.8 6.4 2. ]
 [6.4 2.8 5.6 2.2]
 [6.3 2.8 5.1 1.5]
 [6.1 2.6 5.6 1.4]
 [7.7 3.  6.1 2.3]
 [6.3 3.4 5.6 2.4]
 [6.4 3.1 5.5 1.8]
 [6.  3.  4.8 1.8]
 [6.9 3.1 5.4 2.1]
 [6.7 3.1 5.6 2.4]
 [6.9 3.1 5.1 2.3]
 [5.8 2.7 5.1 1.9]
 [6.8 3.2 5.9 2.3]
 [6.7 3.3 5.7 2.5]
 [6.7 3.  5.2 2.3]
 [6.3 2.5 5.  1.9]
 [6.5 3.  5.2 2. ]
 [6.2 3.4 5.4 2.3]
 [5.9 3.  5.1 1.8]]
class: 0-Iris-Setosa, 1- Iris-Versicolour, 2- Iris-Virginica
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2]
```

Out[2]: KNeighborsClassifier()

In [3]:
```
#To make predictions on our test data
y_pred=classifier.predict(x_test)

print('Confusion Matrix')
print(confusion_matrix(y_test,y_pred))
print('Accuracy Metrics')
print(classification_report(y_test,y_pred))
```

```
Confusion Matrix
[[19  0  0]
 [ 0  7  0]
 [ 0  2 17]]
Accuracy Metrics
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        19
           1       0.78      1.00      0.88         7
           2       1.00      0.89      0.94        19

    accuracy                           0.96        45
   macro avg       0.93      0.96      0.94        45
weighted avg       0.97      0.96      0.96        45
```

KNN without builtins

In [1]:

```
import numpy as np
import scipy.spatial
from collections import Counter
```

In [2]:

```
from sklearn import datasets
from sklearn.model_selection import train_test_split
iris = datasets.load_iris()
X_train, X_test, y_train, y_test = train_test_split(iris.data, iris.target, random_state = 42, test_size = 0.2)
```

In [3]:

```
class KNN:
    def __init__(self, k):
        self.k = k

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y

    def distance(self, X1, X2):
        distance = scipy.spatial.distance.euclidean(X1, X2)

    def predict(self, X_test):
        final_output = []
        for i in range(len(X_test)):
            d = []
            votes = []
            for j in range(len(X_train)):
                dist = scipy.spatial.distance.euclidean(X_train[j] , X_test[i])
                d.append([dist, j])
            d.sort()
            d = d[0:self.k]
            for d, j in d:
                votes.append(y_train[j])
            ans = Counter(votes).most_common(1)[0][0]
            final_output.append(ans)

        return final_output

    def score(self, X_test, y_test):
        predictions = self.predict(X_test)
        return (predictions == y_test).sum() / len(y_test)
```

In [4]:

```
clf = KNN(3)
clf.fit(X_train, y_train)
prediction = clf.predict(X_test)
for i in prediction:
    print(i, end= ' ')
```

```
1 0 2 1 1 0 1 2 1 1 2 0 0 0 0 1 2 1 1 2 0 2 0 2 2 2 2 2 0 0
```

In [5]:

```
clf.score(X_test, y_test)
```

Out[5]:

```
1.0
```

Implement the Linear Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

In [3]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
```

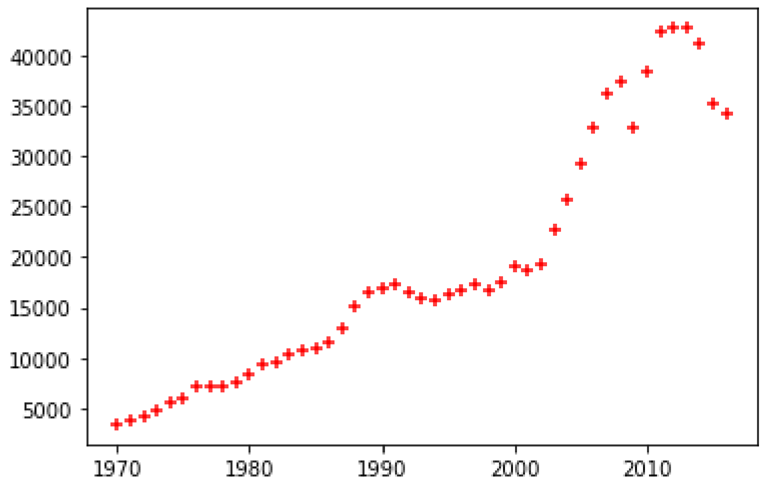
In [4]:
```
df = pd.read_csv('canada_per_capita_income.csv')
df.head(10)
```

Out[4]:

	year	per capita income (US$)
0	1970	3399.299037
1	1971	3768.297935
2	1972	4251.175484
3	1973	4804.463248
4	1974	5576.514583
5	1975	5998.144346
6	1976	7062.131392
7	1977	7100.126170
8	1978	7247.967035
9	1979	7602.912681

In [5]:
```
%matplotlib inline
plt.xlabel = "Year"
plt.ylabel = "Per capita income($)"
plt.scatter(df['year'], df['per capita income (US$)'], color='red', marker='+')
```

Out[5]:
```
<matplotlib.collections.PathCollection at 0x7f2463204a90>
```



In [6]:
```
reg = linear_model.LinearRegression()
reg.fit(df[['year']], df['per capita income (US$)'])
```

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

In [7]:
```
reg.intercept_
```

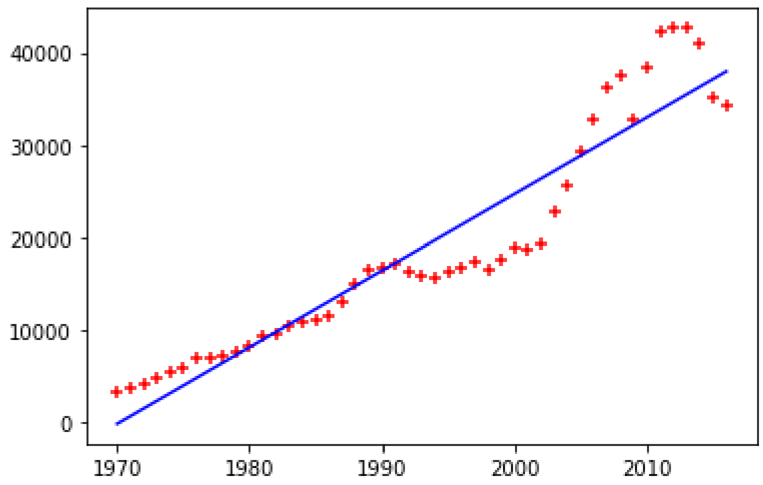
Out[7]:
```
-1632210.7578554575
```

In [8]:
```
reg.coef_
```

Out[8]:
```
array([828.46507522])
```

In [9]:
```
%matplotlib inline
plt.xlabel = "Year"
plt.ylabel = "Per capita income($)"
plt.scatter(df['year'], df['per capita income (US$)'], color='red', marker='+')
plt.plot(df['year'], reg.predict(df[['year']]), color='blue')
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x7f246295c4f0>]
```



In [10]:
```
reg.predict([[2018]])
```

Out[10]:
```
array([39631.76394397])
```

In [11]:
```
reg.coef_ * 2018 + reg.intercept_
```

Out[11]:
```
array([39631.76394397])
```

The prediction of per capita income in the year 2018 is 39631.76

In [21]:
```
preg.predict([[2021]])
```

Out[21]:
```
array([42117.15916964])
```

Linear Regression without builtins

In [4]:
```
import numpy as np
import pandas as pd
```

In [5]:
```
df = pd.read_csv('test_scores.csv')
df
```

Out[5]:

	name	math	cs
0	david	92	98
1	laura	56	68
2	sanjay	88	81
3	wei	70	80
4	jeff	80	83
5	aamir	49	52
6	venkat	65	66
7	virat	35	30
8	arthur	66	68
9	paul	67	73

In [6]:
```
x = np.array(df['math'])
x
```

Out[6]:
```
array([92, 56, 88, 70, 80, 49, 65, 35, 66, 67])
```

In [7]:
```
y = np.array(df['cs'])
y
```

Out[7]:
```
array([98, 68, 81, 80, 83, 52, 66, 30, 68, 73])
```

Gradient descent function:

In [8]:
```
import math
m_curr = b_curr = 0
iterations = 1000000
n = len(x)
learning_rate = 0.0002
cost_previous = 0
for i in range(iterations):
    y_predicted = m_curr * x + b_curr
    m_derivative = -(2/n) * sum(x * (y - y_predicted))
    b_derivative = -(2/n) * sum(y - y_predicted)
    m_curr = m_curr - learning_rate * m_derivative
    b_curr = b_curr - learning_rate * b_derivative
    cost = (1/n) * sum([val ** 2 for val in (y - y_predicted)])
    if math.isclose(cost, cost_previous, rel_tol=1e-20):
        break
    cost_previous = cost
#    print(f'Iteration {i}:  m = {m_curr}  b = {b_curr}  cost = {cost}')
```

In [9]:
```
def Linear_Regression(x_value):
    y_value = m_curr * x_value + b_curr
    return y_value
```

In [10]:
```
x_value = int(input('Enter the marks in math: '))
y_value = Linear_Regression(x_value)
print(f'The predicted marks in \'cs\' is: {y_value} ~ {math.floor(y_value)}')
```

```
Enter the marks in math: 45
The predicted marks in 'cs' is: 47.71330011964905 ~ 47
```

Implement the non-parametric Locally Weighted Regression algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.

In [1]:

```
import numpy as np
from bokeh.plotting import figure, show, output_notebook
from bokeh.layouts import gridplot
from bokeh.io import push_notebook
```

In [4]:

```
def local_regression(x0, X, Y, tau):
    x0 = np.r_[1, x0] # Add one to avoid the loss in information
    X = np.c_[np.ones(len(X)), X]
    xw = X.T * radial_kernel(x0, X, tau) # XTranspose * W
    beta = np.linalg.pinv(xw @ X) @ xw @ Y #@ Matrix Multiplication or Dot Product
    return x0 @ beta # @ Matrix Multiplication or Dot Product for prediction

def radial_kernel(x0, X, tau):
    return np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau * tau))
```

In [5]:

```
n = 1000

X = np.linspace(-3, 3, num=n)
print("The Data Set ( 10 Samples) X :\n",X[1:10])
Y = np.log(np.abs(X ** 2 - 1) + .5)
print("The Fitting Curve Data Set (10 Samples) Y :\n",Y[1:10])
X += np.random.normal(scale=.1, size=n)
print("Normalised (10 Samples) X :\n",X[1:10])
domain = np.linspace(-3, 3, num=300)
print(" Xo Domain Space(10 Samples) :\n",domain[1:10])

def plot_lwr(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plot = figure(plot_width=400, plot_height=400)
    plot.title.text='tau=%g' % tau
    plot.scatter(X, Y, alpha=.3)
    plot.line(domain, prediction, line_width=2, color='red')
    return plot

show(gridplot([
 [plot_lwr(10.), plot_lwr(1.)],
 [plot_lwr(0.1), plot_lwr(0.01)]]))
```

```
The Data Set ( 10 Samples) X :
 [-2.99399399 -2.98798799 -2.98198198 -2.97597598 -2.96996997 -2.96396396
 -2.95795796 -2.95195195 -2.94594595]
The Fitting Curve Data Set (10 Samples) Y :
 [2.13582188 2.13156806 2.12730467 2.12303166 2.11874898 2.11445659
 2.11015444 2.10584249 2.10152068]
Normalised (10 Samples) X :
 [-2.90641362 -2.96633401 -3.11173524 -2.99576271 -3.0929394  -3.11781791
 -2.7381535  -2.85074556 -2.99311933]
 Xo Domain Space(10 Samples) :
 [-2.97993311 -2.95986622 -2.93979933 -2.91973244 -2.89966555 -2.87959866
 -2.85953177 -2.83946488 -2.81939799]
```

Locally Weighted Regression without builtIns

In [8]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [10]:
```
def kernel(point,xmat, k):
    m,n = np.shape(xmat)
    weights = np.mat(np.eye((m))) # eye - identity matrix
    for j in range(m):
        diff = point - X[j]
        weights[j,j] = np.exp(diff*diff.T/(-2.0*k**2))
    return weights

def localWeight(point,xmat,ymat,k):
    wei = kernel(point,xmat,k)
    W = (X.T*(wei*X)).I*(X.T*(wei*ymat.T))
    return W
```

In [15]:
```
def localWeightRegression(xmat,ymat,k):
    m,n = np.shape(xmat)
    ypred = np.zeros(m)
    for i in range(m):
        ypred[i] = xmat[i]*localWeight(xmat[i],xmat,ymat,k)
    return ypred

def graphPlot(X,ypred):
    sortindex = X[:,1].argsort(0) #argsort - index of the smallest
    xsort = X[sortindex][:,0]
    fig = plt.figure()
    ax = fig.add_subplot(1,1,1)
    ax.scatter(bill,tip, color='purple')
    ax.plot(xsort[:,1],ypred[sortindex], color='brown', linewidth=5)
    plt.xlabel('Total bill')
    plt.ylabel('Tip')
    plt.show()
```

In [16]:
```
data = pd.read_csv('tips.csv')
data.head()
```

Out[16]:

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

In [17]:
```
bill = np.array(data.total_bill)
tip = np.array(data.tip)

mbill = np.mat(bill)
mtip = np.mat(tip)
m= np.shape(mbill)[1]
one = np.mat(np.ones(m))
X = np.hstack((one.T,mbill.T))

ypred = localWeightRegression(X,mtip,3)
graphPlot(X,ypred)
```