arvind

July 24, 2024

[1]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab1.csv')
data
features=np.array(data)[:,:-1]
features
target=np.array(data)[:,-1]
target
for i,val in enumerate(target):
    if val =='yes':
        specific_h=features[i].copy()
        break
print(specific_h)
for i,val in enumerate(features):
    if target[i]=='yes':
        for x in range(len(specific_h)):
            if val[x]!=specific_h[x]:
                specific_h[x]='?'
print(specific_h)
```

```
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
['sunny' 'warm' '?' 'strong' '?' '?']
```

[2]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab2.csv')
data
features=np.array(data)[:,:-1]
features
target=np.array(data)[:,-1]
target
for i,val in enumerate(target):
    if val =='yes':
        specific_h=features[i].copy()
        break
print(specific_h)
for i,val in enumerate(features):
    if target[i]=='yes':
```

```
        for x in range(len(specific_h)):
            if val[x]!=specific_h[x]:
                specific_h[x]='?'
print(specific_h)
```

```
['round' 'triangle' 'round' 'purple' 'yes']
['?' '?' 'round' '?' 'yes']
```

[12]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab3.csv')
data
features=np.array(data)[:,:-1]
features
target=np.array(data)[:,-1]
target
for i,val in enumerate(target):
    if val =='yes':
        specific_h=features[i].copy()
        break
print(specific_h)
for i,val in enumerate(features):
    if target[i]=='yes':
        for x in range(len(specific_h)):
            if val[x]!=specific_h[x]:
                specific_h[x]='?'
print(specific_h)
```

```
['yes' 'yes' 'yes']
['yes' 'yes' '?']
```

[19]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab4.csv')
data
features=np.array(data)[:,:-1]
features
target=np.array(data)[:,-1]
target
for i,val in enumerate(target):
    if val =='yes':
        specific_h=features[i].copy()
        break
print(specific_h)
for i,val in enumerate(features):
    if target[i]=='yes':
        for x in range(len(specific_h)):
```

```
            if val[x]!=specific_h[x]:
                specific_h[x]='?'
print(specific_h)
```

```
['many' 'big' 'no' 'expensive' 'one']
['many' '?' 'no' '?' '?']
```

[1]:
```
import pandas as pd
import numpy as np
data=pd.read_csv('lab1.csv')
data
features=np.array(data)[:,:-1]
features
target=np.array(data)[:,-1]
target
specific_h = features[0].copy()
print("Initialization of specific_h and general_h")
print(specific_h)
general_h = [["?"for i in range(len(specific_h))] for i in␣
 ↪range(len(specific_h))]
print(general_h)
```

```
Initialization of specific_h and general_h
['sunny' 'warm' 'normal' 'strong' 'warm' 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?',
'?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', '?']]
```

[2]:
```
for i,h in enumerate(features):
    if target[i]=="yes":
        for x in range(len(specific_h)):
            if h[x]!=specific_h[x]:
                specific_h[x]='?'
                general_h[x][x]='?'
    if target[i] =="no":
        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_h,"\n")
            print(general_h,"\n")
indices=[i for i,val in enumerate(general_h)if val==['?','?','?','?','?','?']]
for i in indices:
    general_h.remove(['?','?','?','?','?','?'])
print("\nfinal specific_h:",specific_h,sep="n")
print("final genral_h:",general_h,sep="\n")
```

```
['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?',
'?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?']]

['sunny' 'warm' '?' 'strong' 'warm' 'same']

[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?',
'?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', 'same']]


final specific_h:n['sunny' 'warm' '?' 'strong' '?' '?']
final genral_h:
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?']]
```

[4]:
```
import pandas as pd
from collections import Counter
import math
tennis= pd.read_csv('lab10.csv')
print('\n Given play Tenis data set:\n\n',tennis)
def entropy(alist):
```

```
    c=Counter(x for x in alist)
    instances=len(alist)
    prob=[x/instances for x in c.values()]
    return sum([-p*math.lag(p,2)for p in prob])
def information_gain(d,split,target):
    splitting=d.groupby(split)
    n=len(d.index)
    agent=splitting.agg({target:[entropy,lambda x:len(x)/n]})[target]
    newentropy=sum(agent['Entrpoy']*agent['observation'])
    oldentropy=entropy(d[target])
    return oldentropy-newentropy
def id3(sub,target,a):
    count=counter(x for x in sub[target])
    if len(count)==1:
        return next(iter(count))
    else:
        gain=[information-gain(sub,attr,target)for attr in a]
        print("/n Gain=",gain)
        maximum=gain.index(max(gain))
        best=a[maximum]
        print=a[maximum]
        print("/n Best Attribute:",best)
        tree={best:{}}
        remaining=[i for i in a if i!=best]
        for val,subset in sub.groupby(best):
            subtree=id3(subset,target,remaining)
            tree[best][val]=subtree
            return tree
        names=list(tennis.columns)
        print("\n List of Attributes:",names)
        names.remove('playTennis')
        print("/n/n Predicting Attributes:",names)
        tree=id3(tennis,'playTennis',names)
        print("/n/n The Resultant Decision Tree is:/n/n")
        print(tree)
```

Given play Tenis data set:

	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
5	No	Rain	Cool	Normal	Strong
6	Yes	Overcast	Cool	Normal	Strong

```
7     No          Sunny      Mild        High    Weak
8     Yes         Sunny      Cool        Normal  Weak
9     Yes         Rain       Mild        Normal  Weak
10  Yes         Sunny      Mild        Normal  Strong
11  Yes         Overcast   Mild        High    Strong
12    Yes         Overcast   Hot         Normal  Weak
13  No          Rain       Mild        High    Strong
```

[5]:
```
import numpy as np
X=np.array(([2,9],[1,5],[3,6]),dtype=float)
y=np.array(([92],[86],[89]),dtype=float)
X=X/np.amax(X,axis=0)
y=y/100

def sigmoid(x):
  return 1/(1+np.exp(-x))

def derivatives_sigmoid(x):
  return x*(1-x)

epoch=7000
lr=0.1
inputlayer_neurons=2
hiddenlayer_neurons=3
output_neurons=1
wh=np.random.uniform(size=(inputlayer_neurons,hiddenlayer_neurons))
bh=np.random.uniform(size=(1,hiddenlayer_neurons))
wout=np.random.uniform(size=(hiddenlayer_neurons,output_neurons))
bout=np.random.uniform(size=(1,output_neurons))
for i in range(epoch):
  hinp1=np.dot(X,wh)
  hinp=hinp1+bh
  hlayer_act=sigmoid(hinp)
  outinp1=np.dot(hlayer_act,wout)
  outinp=outinp1+bout
  output=sigmoid(outinp)

  EO=y-output
  outgrad=derivatives_sigmoid(output)
  d_output=EO*outgrad
  EH=d_output.dot(wout.T)
  hiddengrad=derivatives_sigmoid(hlayer_act)
  d_hiddenlayer=EH*hiddengrad
  wout+=hlayer_act.T.dot(d_output)*lr

print("Input:\n"+str(X))
print("Actual Output:\n"+str(y))
```

```
print("Predicted Output:\n",output)
```

```
Input:
[[0.66666667 1.        ]
 [0.33333333 0.55555556]
 [1.         0.66666667]]
Actual Output:
[[0.92]
 [0.86]
 [0.89]]
Predicted Output:
 [[0.89317747]
 [0.88443897]
 [0.89241563]]
```

[20]:
```
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score
from sklearn import metrics

data=pd.read_csv('textdata.csv',names=['message','lable'])
print('The dataset is',data)
print('The dimensions of the dataset',data.shape)
data['labelnum']=data.lable.map({'pos':1,'neg':0})
x=data.message
y=data.labelnum
print(x)
print(y)
vectorizer=TfidfVectorizer()
data=vectorizer.fit_transform(x)
print('\n the Features of dataset:\n')
df=pd.DataFrame(data.toarray(),columns=vectorizer.get_feature_names_out())
df.head()
print('\nTrain Test Split')
xtrain,xtest,ytrain,ytest=train_test_split(data,y,test_size=0.3,random_state=42)
print('\n the total number of training data:',ytrain.shape)
print('\n the total number of test data:',ytest.shape)
clf=MultinomialNB().fit(xtrain,ytrain)
predict=clf.predict(xtest)
predicted=clf.predict(xtest)
print('\n Accuracy of the classifier is',metrics.
 ↪accuracy_score(ytest,predicted))
print('\n Confusion Matrix is\n',metrics.confusion_matrix(ytest,predicted))
print('\n classification report is\n',metrics.
 ↪classification_report(ytest,predicted))
```

```
print('\n Value of precision is\n',metrics.precision_score(ytest,predicted))
print('\n Value of recall is\n',metrics.recall_score(ytest,predicted))
```

```
The dataset is                                      message lable
0                         i love sandwitch   pos
1                 this is an amazing place   pos
2       i feel very good about these beers   pos
3                     this is my best work   pos
4                     what an awesome view   pos
5              i do not like this restrunt   neg
6                 i am tried of this stuff   neg
7                   i can't deal with this   neg
8                     he is my sworn enemy   neg
9                      my boss is horrible   neg
10                this is an awesome place   pos
11  i do not like the taste of this juice   neg
12                        i love to dance   pos
13      i am sink and tired of this place   neg
14                    what a great holiday   pos
15           that is bad locality to stay   neg
16         we will have good fun tommorrow   pos
17        i went to my enemy's house today   neg
The dimensions of the dataset (18, 2)
0                          i love sandwitch
1                  this is an amazing place
2        i feel very good about these beers
3                      this is my best work
4                      what an awesome view
5                i do not like this restrunt
6                   i am tried of this stuff
7                     i can't deal with this
8                       he is my sworn enemy
9                        my boss is horrible
10                 this is an awesome place
11    i do not like the taste of this juice
12                           i love to dance
13        i am sink and tired of this place
14                      what a great holiday
15             that is bad locality to stay
16           we will have good fun tommorrow
17          i went to my enemy's house today
Name: message, dtype: object
0     1
1     1
2     1
3     1
4     1
```

```
5     0
6     0
7     0
8     0
9     0
10    1
11    0
12    1
13    0
14    1
15    0
16    1
17    0
Name: labelnum, dtype: int64

 the Features of dataset:


Train Test Split

 the total number of training data: (12,)

 the total number of test data: (6,)

 Accuracy of the classifier is 0.8333333333333334

 Confusion Matrix is
 [[3 0]
 [1 2]]

 classification report is
               precision    recall  f1-score   support

           0       0.75      1.00      0.86         3
           1       1.00      0.67      0.80         3

    accuracy                           0.83         6
   macro avg       0.88      0.83      0.83         6
weighted avg       0.88      0.83      0.83         6


 Value of precision is
 1.0

 Value of recall is
 0.6666666666666666
```

[25]:
```
import pandas as pd
col=['Age','Gender','FamilyHist','Diet','LifeStyle','Cholesterol','HeartDisease']
data = pd.read_csv('lab8.csv',names =col )
print(data)
#encoding
from sklearn.preprocessing import LabelEncoder
encoder = LabelEncoder()
for i in range(len(col)):
    data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
#spliting data
X = data.iloc[:,0:6]
y = data.iloc[:,-1]
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)
#prediction
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
#confusion mtx output
from sklearn.metrics import confusion_matrix
print('Confusion matrix',confusion_matrix(y_test, y_pred))
```

```
                  Age  Gender FamilyHist    Diet LifeStyle Cholesterol  \
0   SuperSeniorCitizen    Male        Yes  Medium  Sedetary        High
1   SuperSeniorCitizen  Female        Yes  Medium  Sedetary        High
2        SeniorCitizen    Male         No    High  Moderate  BorderLine
3                 Teen    Male        Yes  Medium  Sedetary      Normal
4                Youth  Female        Yes    High   Athlete      Normal
5           MiddleAged    Male        Yes  Medium    Active        High
6                 Teen    Male        Yes    High  Moderate        High
7   SuperSeniorCitizen    Male        Yes  Medium  Sedetary        High
8                Youth  Female        Yes    High   Athlete      Normal
9        SeniorCitizen  Female         No    High   Athlete      Normal
10                Teen  Female         No  Medium  Moderate        High
11                Teen    Male        Yes  Medium  Sedetary      Normal
12          MiddleAged  Female         No    High   Athlete        High
13          MiddleAged    Male        Yes  Medium    Active        High
14               Youth  Female        Yes    High   Athlete  BorderLine
15  SuperSeniorCitizen    Male        Yes    High   Athlete      Normal
16       SeniorCitizen  Female         No  Medium  Moderate  BorderLine
17               Youth  Female        Yes  Medium   Athlete  BorderLine
18                Teen    Male        Yes  Medium  Sedetary      Normal

   HeartDisease
0           Yes
1           Yes
```

```
2           Yes
3            No
4            No
5           Yes
6           Yes
7           Yes
8            No
9           Yes
10          Yes
11           No
12           No
13          Yes
14           No
15          Yes
16          Yes
17           No
18           No
Confusion matrix [[0 2]
 [0 2]]
```

```
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
```

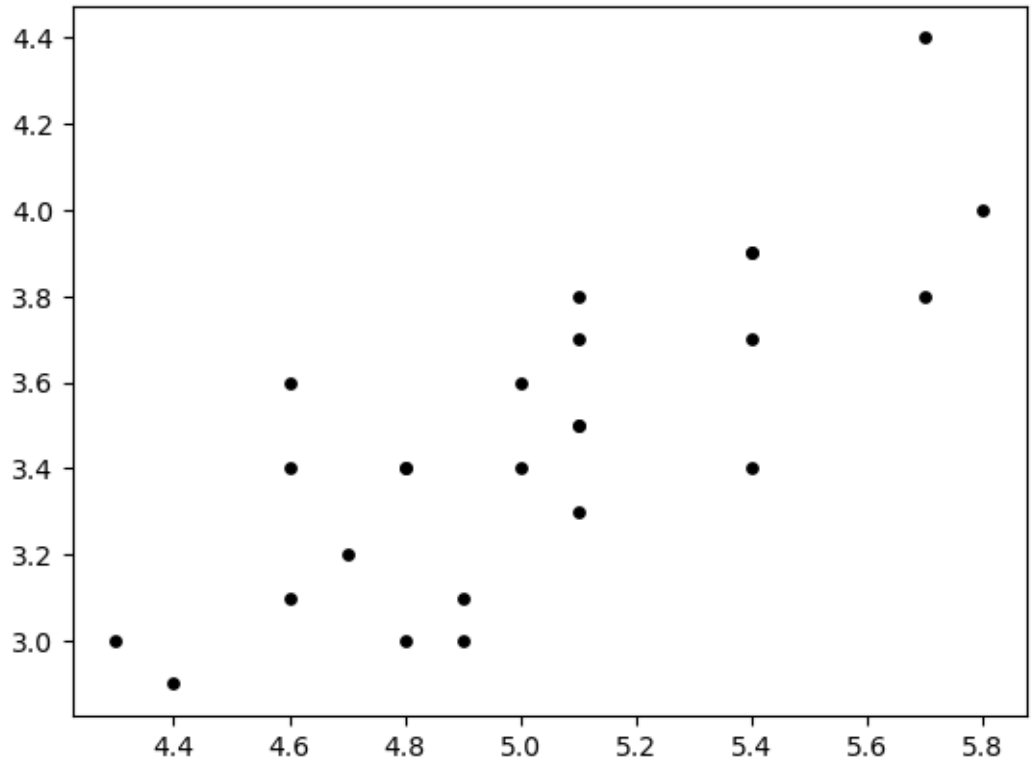
```
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
C:\Users\MicroApt\AppData\Local\Temp\ipykernel_10060\2837126411.py:9:
DeprecationWarning: In a future version, `df.iloc[:, i] = newvals` will attempt
to set the values inplace instead of always setting a new array. To retain the
old behavior, use either `df[df.columns[i]] = newvals` or, if columns are non-
unique, `df.isetitem(i, newvals)`
  data.iloc[:,i] = encoder.fit_transform(data.iloc[:,i])
```

[2]:

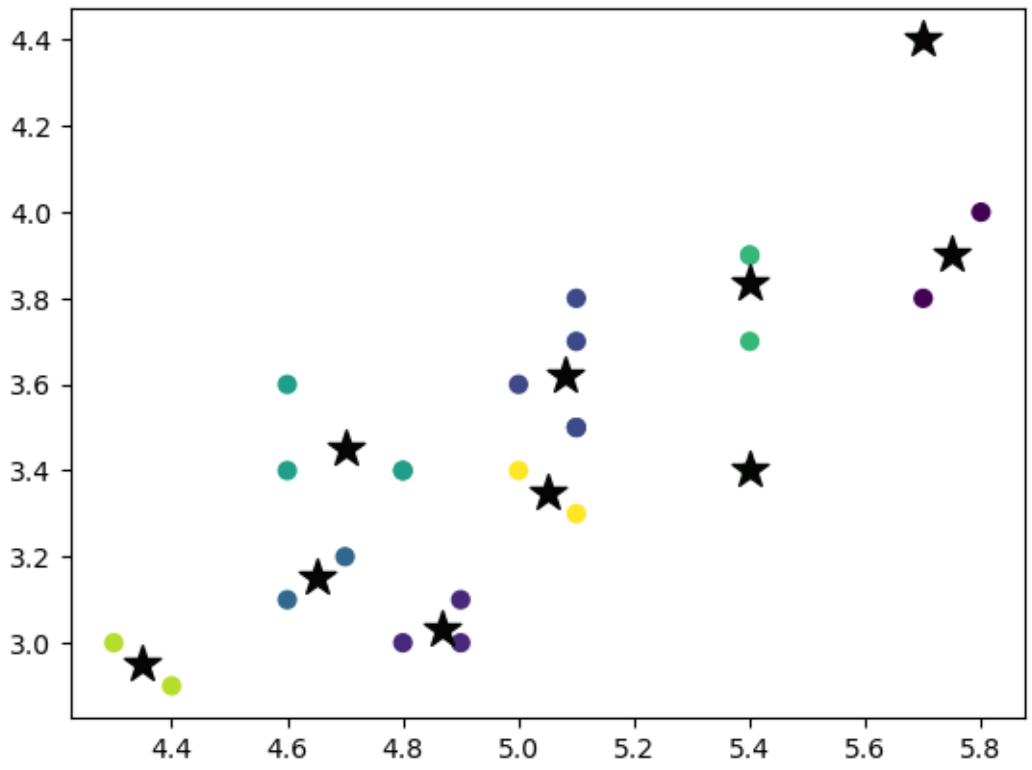
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from sklearn.mixture import GaussianMixture
from sklearn.cluster import KMeans
data = pd.read_csv('lab11.csv')
print("Input Data and Shape")
print(data.shape)
data.head()

f1 = data['V1'].values
f2 = data['V2'].values
X = np.array(list(zip(f1, f2)))

print("X ", X)
print('Graph for whole dataset')
plt.scatter(f1, f2, c='black', s=15)
plt.show()

kmeans = KMeans(10, random_state=42)
labels = kmeans.fit(X).predict(X)
print("labels  ",labels)
centroids = kmeans.cluster_centers_
print("centroids  ",centroids)
plt.scatter(X[:, 0], X[:, 1], c=labels, s=40, cmap='viridis');
print('Graph using Kmeans Algorithm')
plt.scatter(centroids[:, 0], centroids[:, 1], marker='*', s=200, c='#050505')
plt.show()

gmm = GaussianMixture(n_components=3).fit(X)
labels = gmm.predict(X)
```

```
probs = gmm.predict_proba(X)
size = 10 * probs.max(1) ** 3
print('Graph using EM Algorithm')

plt.scatter(X[:, 0], X[:, 1], c=labels, s=size, cmap='viridis');
plt.show()
```

```
Input Data and Shape
(25, 2)
X  [[5.1 3.5]
 [4.9 3. ]
 [4.7 3.2]
 [4.6 3.1]
 [5.  3.6]
 [5.4 3.9]
 [4.6 3.4]
 [5.  3.4]
 [4.4 2.9]
 [4.9 3.1]
 [5.4 3.7]
 [4.8 3.4]
 [4.8 3. ]
 [4.3 3. ]
 [5.8 4. ]
 [5.7 4.4]
 [5.4 3.9]
 [5.1 3.5]
 [5.7 3.8]
 [5.1 3.8]
 [5.4 3.4]
 [5.1 3.7]
 [4.6 3.6]
 [5.1 3.3]
 [4.8 3.4]]
Graph for whole dataset
```

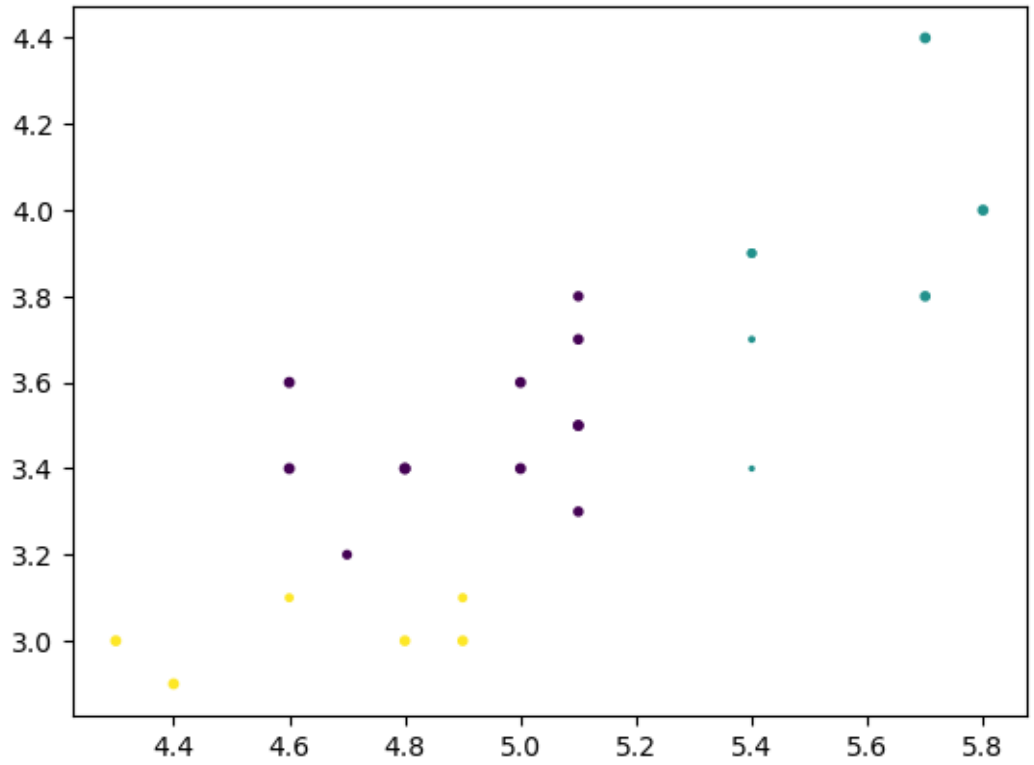



```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:870:
FutureWarning: The default value of `n_init` will change from 10 to 'auto' in
1.4. Set the value of `n_init` explicitly to suppress the warning
  warnings.warn(
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(

labels   [2 1 3 3 2 6 5 9 8 1 6 5 1 8 0 4 6 2 0 2 7 2 5 9 5]
centroids   [[5.75       3.9       ]
 [4.86666667 3.03333333]
 [5.08       3.62      ]
 [4.65       3.15      ]
 [5.7        4.4       ]
 [4.7        3.45      ]
 [5.4        3.83333333]
 [5.4        3.4       ]
 [4.35       2.95      ]
 [5.05       3.35      ]]
Graph using Kmeans Algorithm
```



Graph using EM Algorithm

```
C:\ProgramData\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1382:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



[1]:
```
import csv
import random
import math
import operator
def loadDataset(filename, split, trainingSet=[], testSet=[]):
    with open(filename) as csvfile:
        lines = csv.reader(csvfile)
        dataset = list(lines)
        for x in range(len(dataset)-1):
            for y in range(4):
                dataset[x][y] = float(dataset[x][y])
            if random.random() < split:
                trainingSet.append(dataset[x])
            else:
                testSet.append(dataset[x])
def euclideanDistance(instance1, instance2, length):
    distance = 0
    for x in range(length):
        distance += pow((instance1[x] - instance2[x]), 2)
    return math.sqrt(distance)

def getNeighbors(trainingSet, testInstance, k):
```

```
    distances = []
    length = len(testInstance)-1
    for x in range(len(trainingSet)):
        dist = euclideanDistance(testInstance, trainingSet[x], length)
        distances.append((trainingSet[x], dist))
    distances.sort(key=operator.itemgetter(1))
    neighbors = []
    for x in range(k):
        neighbors.append(distances[x][0])
    return neighbors

def getResponse(neighbors):
    classVotes = {}
    for x in range(len(neighbors)):
        response = neighbors[x][-1]
        if response in classVotes:
            classVotes[response] += 1
        else:
            classVotes[response] = 1
    sortedVotes = sorted(classVotes.items(), key=operator.itemgetter(1),␣
 ↪reverse=True)
    return sortedVotes[0][0]

def getAccuracy(testSet, predictions):
    correct = 0
    for x in range(len(testSet)):
        if testSet[x][-1] == predictions[x]:
            correct += 1
    return (correct/float(len(testSet))) * 100.0

def main():

    trainingSet=[]
    testSet=[]
    split = 0.67
    loadDataset('iris_data.csv', split, trainingSet, testSet)
    print ('\n Number of Training data: ' + (repr(len(trainingSet))))
    print (' Number of Test Data: ' + (repr(len(testSet))))

    predictions=[]
    k = 3
    print('\n The predictions are: ')
    for x in range(len(testSet)):
        neighbors = getNeighbors(trainingSet, testSet[x], k)
        result = getResponse(neighbors)
        predictions.append(result)
        print(' predicted=' + repr(result) + ', actual=' + repr(testSet[x][-1]))
```

```
    accuracy = getAccuracy(testSet, predictions)
    print('\n The Accuracy is: ' + repr(accuracy) + '%')

main()
```

```
Number of Training data: 103
Number of Test Data: 46

The predictions are:
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-setosa', actual='Iris-setosa'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-versicolor', actual='Iris-versicolor'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-versicolor', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
```

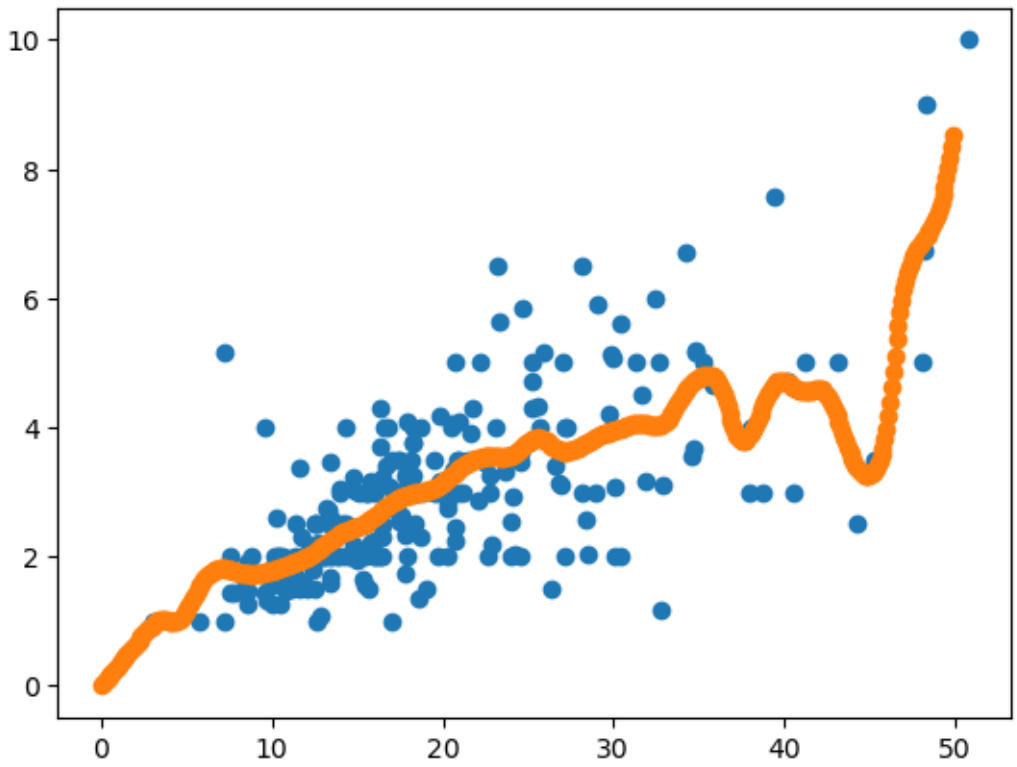
```
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'
predicted='Iris-virginica', actual='Iris-virginica'

The Accuracy is: 93.47826086956522%
```

[18]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
tou = 1
data=pd.read_csv("tips10.csv")
X_train = np.array(data.total_bill)
print(X_train)
X_train = X_train[:, np.newaxis]
print(len(X_train))
y_train = np.array(data.tip)

X_test = np.array([i /10 for i in range(500)])
X_test = X_test[:, np.newaxis]
y_test = []
count = 0
for r in range(len(X_test)):
    wts = np.exp(-np.sum((X_train - X_test[r]) ** 2, axis=1) / (2 * tou ** 2))
    W = np.diag(wts)
    factor1 = np.linalg.inv(X_train.T.dot(W).dot(X_train))
    parameters = factor1.dot(X_train.T).dot(W).dot(y_train)
    prediction = X_test[r].dot(parameters)
    y_test.append(prediction)
    count += 1
    print(len(y_test))
y_test = np.array(y_test)
plt.plot(X_train.squeeze(), y_train, 'o')

plt.plot(X_test.squeeze(), y_test, 'o')
plt.show()
```

```
[16.99 10.34 21.01 23.68 24.59 25.29  8.77 26.88 15.04 14.78 10.27 35.26
 15.42 18.43 14.83 21.58 10.33 16.29 16.97 20.65 17.92 20.29 15.77 39.42
 19.82 17.81 13.37 12.69 21.7  19.65  9.55 18.35 15.06 20.69 17.78 24.06
 16.31 16.93 18.69 31.27 16.04 17.46 13.94  9.68 30.4  18.29 22.23 32.4
 28.55 18.04 12.54 10.29 34.81  9.94 25.56 19.49 38.01 26.41 11.24 48.27
 20.29 13.81 11.02 18.29 17.59 20.08 16.45  3.07 20.23 15.01 12.02 17.07
```

```
 26.86 25.28 14.73 10.51 17.92 27.2  22.76 17.29 19.44 16.66 10.07 32.68
 15.98 34.83 13.03 18.28 24.71 21.16 28.97 22.49  5.75 16.32 22.75 40.17
 27.28 12.03 21.01 12.46 11.35 15.38 44.3  22.42 20.92 15.36 20.49 25.21
 18.24 14.31 14.    7.25 38.07 23.95 25.71 17.31 29.93 10.65 12.43 24.08
 11.69 13.42 14.26 15.95 12.48 29.8   8.52 14.52 11.38 22.82 19.08 20.27
 11.17 12.26 18.26  8.51 10.33 14.15 16.   13.16 17.47 34.3  41.19 27.05
 16.43  8.35 18.64 11.87  9.78  7.51 14.07 13.13 17.26 24.55 19.77 29.85
 48.17 25.   13.39 16.49 21.5  12.66 16.21 13.81 17.51 24.52 20.76 31.71
 10.59 10.63 50.81 15.81  7.25 31.85 16.82 32.9  17.89 14.48  9.6  34.63
 34.65 23.33 45.35 23.17 40.55 20.69 20.9  30.46 18.15 23.1  15.69 19.81
 28.44 15.48 16.58  7.56 10.34 43.11 13.   13.51 18.71 12.74 13.   16.4
 20.53 16.47 26.59 38.73 24.27 12.76 30.06 25.89 48.33 13.27 28.17 12.9
 28.15 11.59  7.74 30.14 12.16 13.42  8.58 15.98 13.42 16.27 10.09 20.45
 13.28 22.12 24.01 15.69 11.61 10.77 15.53 10.07 12.6  32.83 35.83 29.03
 27.18 22.67 17.82 18.78]
244
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
```

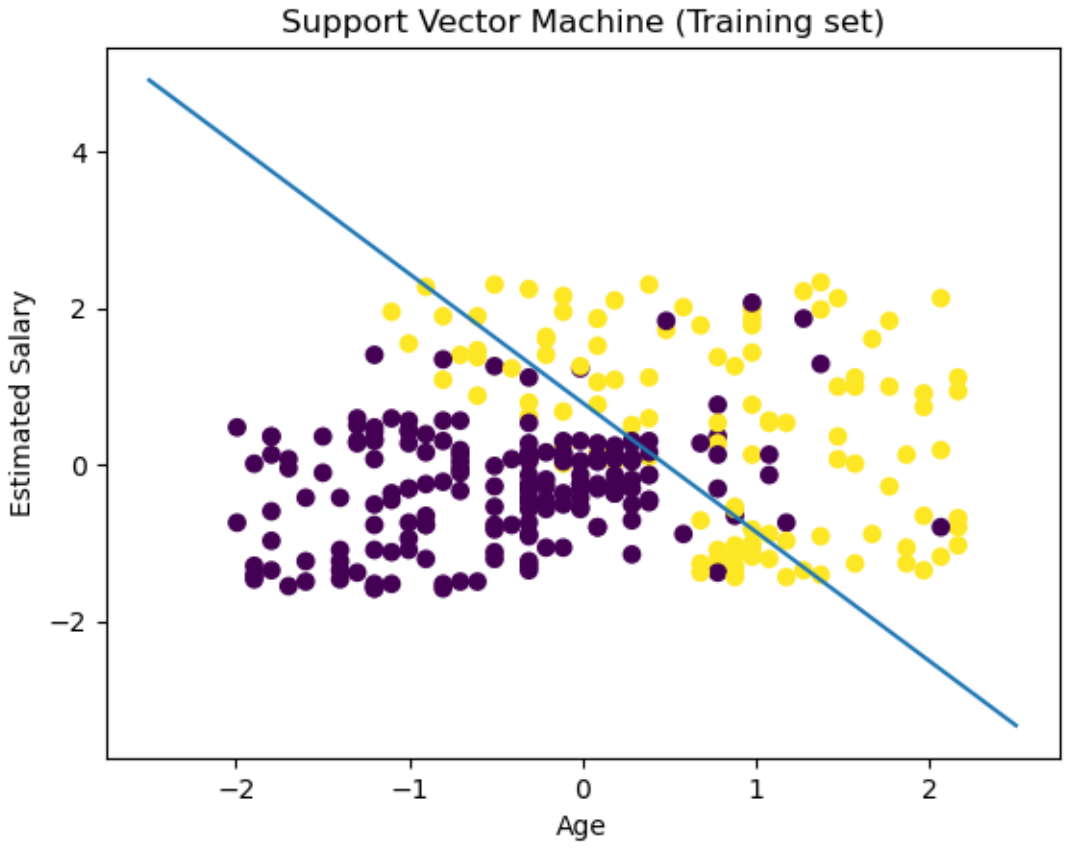
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224

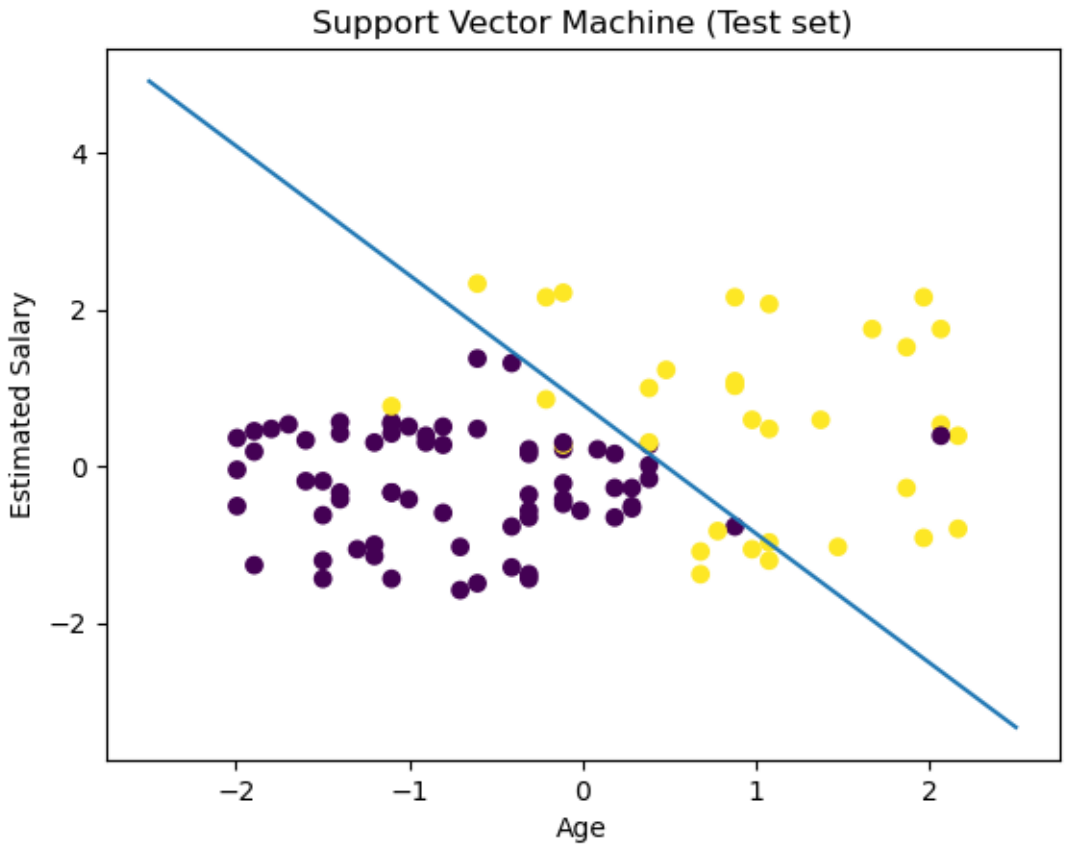


[15]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
datasets = pd.read_csv('lab12.csv')
X = datasets.iloc[:, [2,3]].values
Y = datasets.iloc[:, 4].values
from sklearn.model_selection import train_test_split
X_Train, X_Test, Y_Train, Y_Test = train_test_split(X, Y, test_size = 0.25,
random_state = 0)
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_Train = sc_X.fit_transform(X_Train)
X_Test = sc_X.transform(X_Test)
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_Train, Y_Train)
Y_Pred = classifier.predict(X_Test)
from sklearn import metrics
print("Accuracy score ",metrics.accuracy_score(Y_Test, Y_Pred))
plt.scatter(X_Train[:,0], X_Train[:, 1],c=Y_Train)
plt.title('Support Vector Machine (Training set)')
plt.xlabel('Age')
```

```
plt.ylabel('Estimated Salary')
w=classifier.coef_[0]
a=-w[0]/w[1]
xx=np.linspace(-2.5,2.5)
yy=a*xx -(classifier.intercept_[0])/w[1]
plt.plot(xx,yy)
plt.show();
plt.scatter(X_Test[:,0], X_Test[:, 1],c=Y_Test)
plt.title('Support Vector Machine (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
w=classifier.coef_[0]
a=-w[0]/w[1]
xx=np.linspace(-2.5,2.5)
yy=a*xx -(classifier.intercept_[0])/w[1]
plt.plot(xx,yy)
plt.show();
```

```
Accuracy score  0.9
```




Support Vector Machine (Test set)
Estimated Salary
Age
4
2
0
−2
−2
−1
0
1
2

[ ]:

[ ]:

[ ]: