MACHINE LEARNING LABORATORY

DIGITAL ASSIGNMENT 1

Name: SHASHANK VENKAT

Register Number: 20BCE2961

Slot: L39+L40

Course Code: CSE 4020

Submitted To: Prof. JAISANKAR N

Number of Pages: 16

1. Find S algorithm

AIM: To perform Find S algorithm

ALGORITHM:

```
Initialize hypothesis to the most specific hypothesis

For every positive training instance x:
    For each attribute constraint a, in h:
         If a is fulfilled by x:
              Do nothing
         Else:
              Replace a in h by general constraint.
Return Hypothesis
```

TRAINING EXAMPLE:

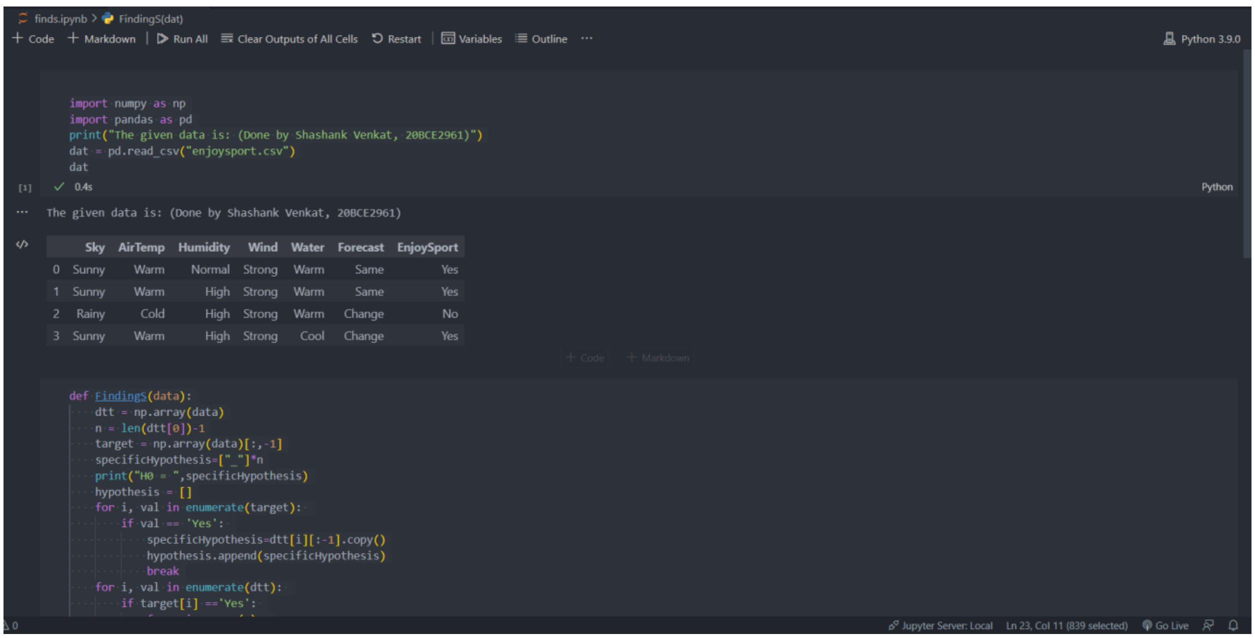
	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

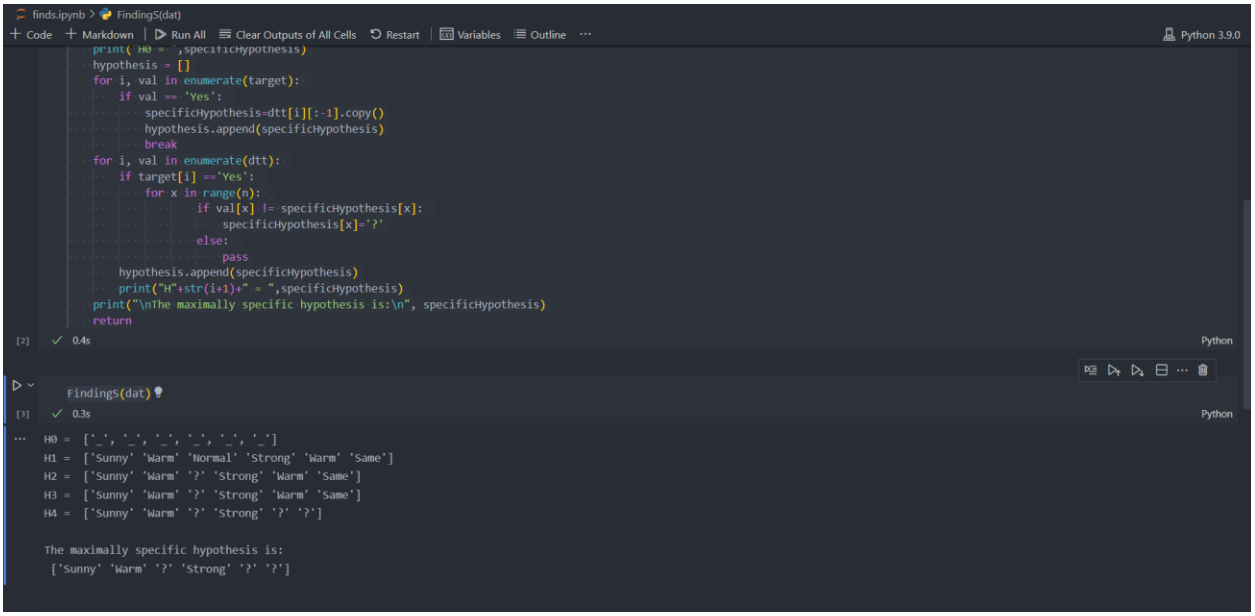
CODE:

```
import numpy as np
import pandas as pd
print("The given data is: (Done by Shashank Venkat, 20BCE2961)")
dat = pd.read_csv("enjoysport.csv")
dat
def FindingS(data):
    dtt = np.array(data)
```

```
    n = len(dtt[0])-1
    target = np.array(data)[:,-1]
    specificHypothesis=["_"]*n
    print("H0 = ",specificHypothesis)
    hypothesis = []
    for i, val in enumerate(target):
        if val == 'Yes':
            specificHypothesis=dtt[i][:-1].copy()
            hypothesis.append(specificHypothesis)
            break
    for i, val in enumerate(dtt):
        if target[i] =='Yes':
            for x in range(n):
                    if val[x] != specificHypothesis[x]:
                        specificHypothesis[x]='?'
                    else:
                        pass
        hypothesis.append(specificHypothesis)
        print("H"+str(i+1)+" = ",specificHypothesis)
    print("\nThe maximally specific hypothesis is:\n", specificHypothesis)
    return
FindingS(dat)
```

OUTPUT





2. Candidate Elimination Algorithm

AIM: To perform Candidate Elimination Algorithm

ALGORITHM:

```
Load Data set
Initialize General Hypothesis and Specific Hypothesis.
For each training example
 If example is positive
         if attribute_value == hypothesis_value:
             continue
         else:
             replace with '?'
 If example is Negative example
          Make generalize hypothesis more specific.
```

TRAINING EXAMPLE:

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

CODE:

```
import pandas as pd
import numpy as np
dat = pd.read_csv("enjoysport.csv")
dat
def candidateElimination(data):
    dataset = data.values.tolist()
    print("\nThe dataset is :\n",dataset)
    S=dataset[0][0:-1]
    print("The initial value of s is :\n",S)
    G=[['?' for i in range(len(S))] for j in range(len(S))]
    print("The initial value of g is :\n",G)
    for xrow in dataset:
        if xrow[-1]=="Yes":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    S[j]='?'
                    G[j][j]='?'
        elif xrow[-1]=="No":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    G[j][j]=S[j]
                else:
                    G[j][j]="?"
        print("\nAfter",dataset.index(xrow)+1,"th insatnce")
        print("Specific boundary  :",S)
        print("General boundary  :",G)

candidateElimination(dat)
```

OUTPUT:

candidateElimination.ipynb > candidateElimination(dat)

+ Code + Markdown | Run All Clear Outputs of All Cells Restart | Variables Outline ···

```python
import pandas as pd
import numpy as np
dat = pd.read_csv("enjoysport.csv")
dat
```

[5]

	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
0	Sunny	Warm	Normal	Strong	Warm	Same	Yes
1	Sunny	Warm	High	Strong	Warm	Same	Yes
2	Rainy	Cold	High	Strong	Warm	Change	No
3	Sunny	Warm	High	Strong	Cool	Change	Yes

```python
def candidateElimination(data):
    dataset = data.values.tolist()
    print("\nThe dataset is :\n",dataset)
    S=dataset[0][0:-1]
    print("The initial value of s is :\n",S)
    G=[['?' for i in range(len(S))] for j in range(len(S))]
    print("The initial value of g is :\n",G)
    for xrow in dataset:
        if xrow[-1]=="Yes":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    S[j]='?'
                    G[j][j]='?'
        elif xrow[-1]=="No":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    G[j][j]=S[j]
```

```python
def candidateElimination(data):
    dataset = data.values.tolist()
    print("\nThe dataset is :\n",dataset)
    S=dataset[0][0:-1]
    print("The initial value of s is :\n",S)
    G=[['?' for i in range(len(S))] for j in range(len(S))]
    print("The initial value of g is :\n",G)
    for xrow in dataset:
        if xrow[-1]=="Yes":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    S[j]='?'
                    G[j][j]='?'
        elif xrow[-1]=="No":
            for j in range(len(S)):
                if xrow[j]!=S[j]:
                    G[j][j]=S[j]
                else:
                    G[j][j]="?"
        print("\nAfter",dataset.index(xrow)+1,"th insatnce")
        print("Specific boundary  :",S)
        print("General boundary  :",G)
```

Python

```
The dataset is :
 [['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same', 'Yes'], ['Sunny', 'Warm', 'High', 'Strong', 'Warm', 'Same', 'Yes'], ['Rainy', 'Cold', 'High', 'Strong', 'Warm', 'Change', 'No'], ['Sunny', 'Warm', 'High', 'Strong', 'Cool', 'Change', 'Yes']]
The initial value of s is :
 ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
The initial value of g is :
 [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 1 th insatnce
Specific boundary  : ['Sunny', 'Warm', 'Normal', 'Strong', 'Warm', 'Same']
General boundary  : [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 2 th insatnce
Specific boundary  : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General boundary  : [['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]

After 3 th insatnce
Specific boundary  : ['Sunny', 'Warm', '?', 'Strong', 'Warm', 'Same']
General boundary  : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', 'Same']]

After 4 th insatnce
Specific boundary  : ['Sunny', 'Warm', '?', 'Strong', '?', '?']
General boundary  : [['Sunny', '?', '?', '?', '?', '?'], ['?', 'Warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
```

3. Simple Linear Regression

AIM: To perform Simple Linear Regression

ALGORITHM:

```
Read Number of Data (n)

For i=1 to n:
     Read Xi and Yi
   Next i

Initialize:
     sumX = 0
     sumX2 = 0
     sumY = 0
     sumXY = 0
Calculate Required Sum
   For i=1 to n:
     sumX = sumX + Xi
     sumX2 = sumX2 + Xi * Xi
     sumY = sumY + Yi
     sumXY = sumXY + Xi * Yi
   Next i

 Calculate Required Constant a and b of y = a + bx:
```

```
b = (n * sumXY - sumX * sumY)/(n*sumX2 - sumX * sumX)
a = (sumY - b*sumX)/n
```

TRAINING EXAMPLE:

	YearsOfExperience	Salary
0	1.2	38976
1	1.3	45897
2	1.5	36987
3	1.4	40587
4	1.3	42984
5	1.7	47986
6	2.0	44578
7	2.2	38789
8	2.4	46986
9	2.6	47986
10	2.9	56642
11	3.0	60150

CODE:

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
dat = pd.read_csv('salary_data.csv')
dat
X = dat.iloc[:, :-1].values
y = dat.iloc[:, 1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3,
random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
y_pred
viz_train = plt
viz_train.scatter(X_train, y_train, color='purple')
viz_train.plot(X_train, regressor.predict(X_train), color='orange')
```

```
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.title('Salary VS Experience (Training set)')
viz_train.show()
viz_test = plt
viz_test.scatter(X_test, y_test, color='purple')
viz_test.plot(X_train, regressor.predict(X_train), color='orange')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
print("Equation of the resulting regression line is: y = ",
regressor.coef_,"*x + ",regressor.intercept_)
pd.DataFrame({'x_test':list(X_test), 'y_test':list(y_test),
'y_pred':list(y_pred)})
```

OUTPUT:

+ Code + Markdown | Run All Clear Outputs of All Cells Restart | Variables

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
```

[9] ✓ 0.3s

```
dat = pd.read_csv('salary_data.csv')
dat
```

[10] ✓ 0.7s

	YearsOfExperience	Salary
0	1.2	38976
1	1.3	45897
2	1.5	36987
3	1.4	40587
4	1.3	42984
5	1.7	47986
6	2.0	44578
7	2.2	38789
8	2.4	46986
9	2.6	47986
10	2.9	56642
11	3.0	60150
12	3.2	54445
13	3.3	58763
14	3.5	56498
15	3.9	63218
16	3.2	63987
17	3.6	58736
18	3.9	62948
19	4.0	54874

```
X = dat.iloc[:, :-1].values
y = dat.iloc[:, 1].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=1/3, random_state=0)
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
y_pred
```

✓ 0.5s

```
array([ 74112.71939557,  82662.88348193, 104513.30281375,  81712.86525012,
        54162.33652739,  39912.06305012,  90263.02933648, 108313.37574102,
        93113.08403193,  63662.51884557,  38012.02658648,  53212.31829557,
        76012.75585921, 100713.22988648,  82662.88348193, 102613.26635012,
       113063.46690012,  46562.19067285,  58912.42768648,  83612.90171375])
```

```
viz_train = plt
viz_train.scatter(X_train, y_train, color='purple')
viz_train.plot(X_train, regressor.predict(X_train), color='orange')
viz_train.xlabel('Year of Experience')
viz_train.ylabel('Salary')
viz_train.title('Salary VS Experience (Training set)')
viz_train.show()
viz_test = plt
viz_test.scatter(X_test, y_test, color='purple')
viz_test.plot(X_train, regressor.predict(X_train), color='orange')
viz_test.title('Salary VS Experience (Test set)')
viz_test.xlabel('Year of Experience')
viz_test.ylabel('Salary')
viz_test.show()
print("Equation of the resulting regression line is: y = ",
regressor.coef_,"*x + ",regressor.intercept_)
pd.DataFrame({'x_test':list(X_test), 'y_test':list(y_test),
'y_pred':list(y_pred)})
```

✓ 0.2s

```
Equation of the resulting regression line is: y =  [9500.18231818] *x +  25661.789572846363
```

	x_test	y_test	y_pred
0	[5.1]	67938	74112.719396
1	[6.0]	91029	82662.883482
2	[8.3]	108374	104513.302814
3	[5.9]	81293	81712.865250
4	[3.0]	60150	54162.336527
5	[1.5]	36987	39912.063050
6	[6.8]	93847	90263.029336
7	[8.7]	109893	108313.375741
8	[7.1]	98376	93113.084032
9	[4.0]	57643	63662.518846
10	[1.3]	42984	38012.026586
11	[2.9]	56642	53212.318296
12	[5.3]	82903	76012.755859
13	[7.9]	102893	100713.229886
14	[6.0]	92839	82662.883482
15	[8.1]	114938	102613.266350
16	[9.2]	119384	113063.466900
17	[2.2]	38789	46562.190673
18	[3.5]	56498	58912.427686
19	[6.1]	94038	83612.901714

4. **Multiple Regression**

AIM: To perform Linear Regression

ALGORITHM:

```
Read Number of Data (n)

For i=1 to n:
     Read Xi and Yi
   Next i
```

```
Initialize:
     sumX = 0
     sumX2 = 0
     sumY = 0
     sumXY = 0
Calculate Required Sum
   For i=1 to n:
     sumX = sumX + Xi
     sumX2 = sumX2 + Xi * Xi
     sumY = sumY + Yi
     sumXY = sumXY + Xi * Yi
   Next i

 Calculate Required Constant a and b of y = a + bx:
   b = (n * sumXY - sumX * sumY)/(n*sumX2 - sumX * sumX)
   a = (sumY - b*sumX)/n
```

TRAINING EXAMPLE:

	area	bedrooms	age	price
0	2600	3	20	550000
1	3000	4	15	565000
2	3200	4	18	610000
3	3600	3	30	595000
4	4000	5	8	760000
5	4100	6	8	810000

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import math
dat=pd.read_csv("house_price.csv")
dat.bedrooms.median()
median_bedrooms=math.floor(dat.bedrooms.median())
dat.bedrooms=dat.bedrooms.fillna(median_bedrooms)
dat
reg = linear_model.LinearRegression()
```

```
reg.fit(dat[['area','bedrooms','age']],dat.price)
reg.coef_
print("coefficients of x in line are:",reg.coef_)
reg.intercept_
print("intercept of line",reg.intercept_)
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house")
print(reg.predict([[3000, 3, 40]]))
print("price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house")
print(reg.predict([[2500, 4, 5]]))
```

OUTPUT

```
multipleRegrression.ipynb > import pandas as pd
+ Code  + Markdown | Run All  Clear All Outputs  Restart | Variables  Outline ...        Python 3.9.0

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
import math
dat=pd.read_csv("house_price.csv")
dat.bedrooms.median()
median_bedrooms=math.floor(dat.bedrooms.median())
dat.bedrooms=dat.bedrooms.fillna(median_bedrooms)
dat
reg = linear_model.LinearRegression()
reg.fit(dat[['area','bedrooms','age']],dat.price)
reg.coef_
print("coefficients of x in line are:",reg.coef_)
reg.intercept_
print("intercept of line",reg.intercept_)
print("price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house")
print(reg.predict([[3000, 3, 40]]))
print("price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house")
print(reg.predict([[2500, 4, 5]]))
[3]  ✓ 0.0s                                                              Python

... coefficients of x in line are: [  112.06244194 23388.88007794 -3231.71790863]
intercept of line 221323.0018654043
price of home with 3000 sqr ft area, 3 bedrooms, 40 year old house
[498408.25158031]
price of home with 2500 sqr ft area, 4 bedrooms, 5 year old house
[578876.03748933]
```

5. Logistic Regression

AIM: To perform Logistic Regression

ALGORITHM:

Load Data set.
Plot the dataset.
Create a logistic regression model using sklearn.
Fit the training data to the model.
Test the model with training data.

TRAINING EXAMPLE:

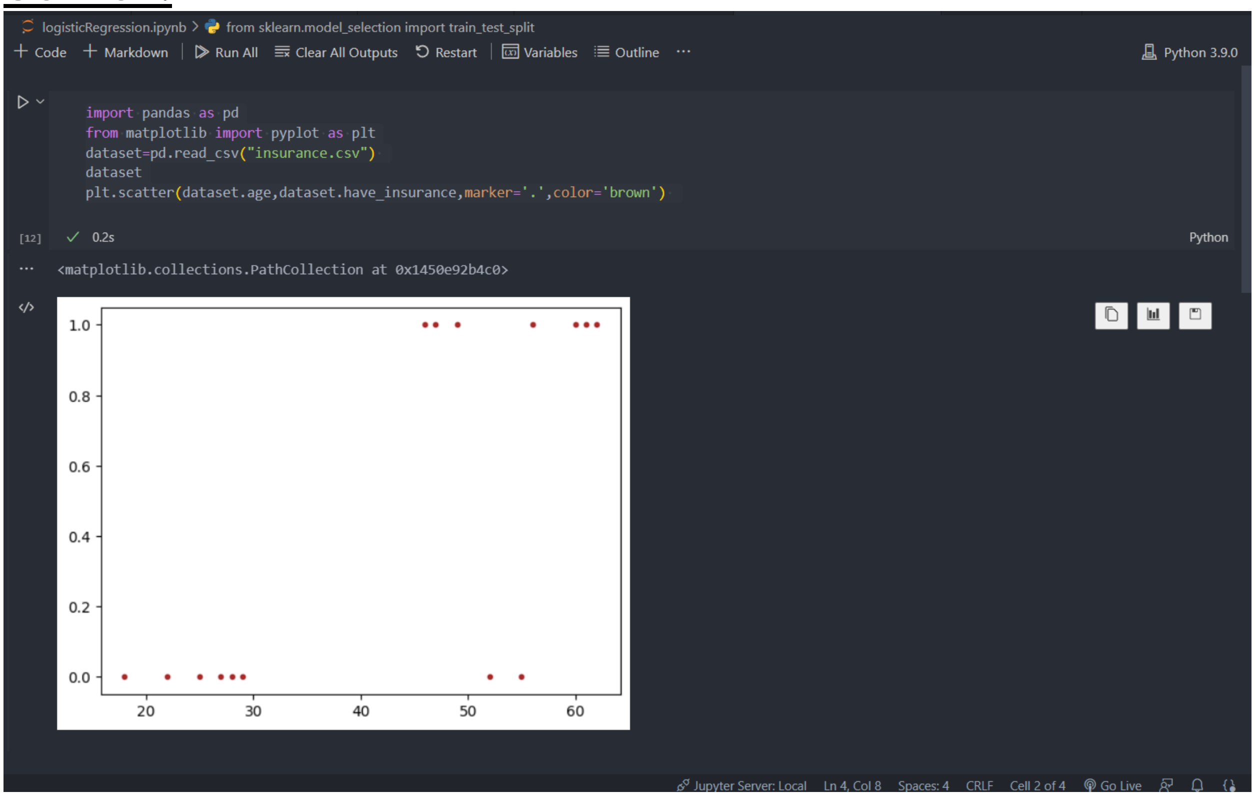
	age	have_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0
11	28	0
12	27	0
13	29	0
14	49	1

CODE:

```
import pandas as pd
from matplotlib import pyplot as plt
dataset=pd.read_csv("insurance.csv")
dataset
plt.scatter(dataset.age,dataset.have_insurance,marker='.',color='brown')
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test =
train_test_split(dataset[['age']],dataset.have_insurance,train_size=0.8)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='auto',
n_jobs=None, penalty='12', random_state=None, solver='lbfgs', tol=0.0001,
verbose=0, warm_start=False)
print("coefficient of x is",model.coef_)
print("intercept of line is",model.intercept_)
import math
```

```python
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z= 0.280409*age -7.942535
    y=sigmoid(z)
    return y
age=33
y=prediction_function(age)
print("Probability of person with age 33 having insurance is",y)
print("As 0.787 is greater than 0.5 which means person with age 33 has insurance
")
```

OUTPUT:



```python
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(dataset[['age']],dataset.have_insurance,train_size=0.8)
from sklearn.linear_model import LogisticRegression
model=LogisticRegression()
model.fit(x_train, y_train)
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True, intercept_scaling=1, l1_ratio=None, max_iter=100, multi_class='aut
print("coefficient of x is",model.coef_)
print("intercept of line is",model.intercept_)
import math
def sigmoid(x):
    return 1/(1+math.exp(-x))
def prediction_function(age):
    z= 0.280409*age -7.942535
    y=sigmoid(z)
    return y
age=33
y=prediction_function(age)
print("Probability of person with age 33 having insurance is",y)
print("As 0.787 is greater than 0.5 which means person with age 33 has insurance ")
```

[13] 0.0s Python

```
coefficient of x is [[0.22856234]]
intercept of line is [-11.78753145]
Probability of person with age 33 having insurance is 0.78767408876316
As 0.787 is greater than 0.5 which means person with age 33 has insurance
```

MACHINE LEARNING LABORATORY

DIGITAL ASSIGNMENT 2

Name: SHASHANK VENKAT

Register Number: 20BCE2961

Slot: L39+L40

Course Code: CSE 4020

Submitted To: Prof. JAISANKAR N

Number of Pages: 18

Regno: 20BCE2961

Submitted To: Jaisankar N

1. ID3 ALGORITHM

AIM: To perform ID3 Algorithm

ALGORITHM:

```
Create a Root node for the decision tree
   If all examples have the same value for Target_Attribute,
return the single-node tree Root, with label = this value
   If the Attributes list is empty, return the single-node tree
Root, with label = most common value of Target_Attribute in the
examples
   Otherwise, choose the best attribute to split the examples
      The best attribute is the one with the highest information
gain
      Information gain can be calculated using entropy or
information gain ratio
   Add a new decision node Root, corresponding to the best
attribute
   For each possible value of the best attribute, create a new
subset of examples that have that value
      If the subset is empty, add a new leaf node with label =
most common value of Target_Attribute in the examples
      Else, call ID3 recursively with the subset as the new
examples and repeat from step 2
   Return Root
```

TRAINING EXAMPLE:

```
Sample Dataset -
       a1    a2      a3 classification
0   True   Hot    High             No
1   True   Hot    High             No
2  False   Hot    High            Yes
3  False  Cool  Normal            Yes
4  False  Cool  Normal            Yes
5   True  Cool    High             No
6   True   Hot    High             No
7   True   Hot  Normal            Yes
8  False  Cool  Normal            Yes
9  False  Cool    High            Yes
```

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('id3.csv')
print("Sample Dataset - \n",data,"\n")
La = LabelEncoder()
data['a1_n'] = La.fit_transform(data['a1'])
le_a2 = LabelEncoder()
data['a2_n'] = La.fit_transform(data['a2'])
le_a3 = LabelEncoder()
data['a3_n'] = La.fit_transform(data['a3'])
print("Given Data after Encoding - \n",data,"\n")
X = data[['a1_n','a2_n','a3_n']]
print("X - Values\n",X,"\n")
y = data['classification']
print("Y - Values\n",y,"\n")
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train,y_train)
print("Values predicted from test dataset - ",model.predict(X_test))
print("Original Values of test dataset - ",y_test.values)
print("Accuracy of Model",model.score(X_test,y_test))
```

OUTPUT

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

✓ 4.3s

```
data = pd.read_csv('id3.csv')
print("Sample Dataset - \n",data,"\n")
La = LabelEncoder()
data['a1_n'] = La.fit_transform(data['a1'])
le_a2 = LabelEncoder()
data['a2_n'] = La.fit_transform(data['a2'])
le_a3 = LabelEncoder()
data['a3_n'] = La.fit_transform(data['a3'])
```

✓ 0.1s

```
Sample Dataset -
       a1    a2      a3 classification
0   True   Hot    High             No
1   True   Hot    High             No
2  False   Hot    High            Yes
3  False  Cool  Normal            Yes
4  False  Cool  Normal            Yes
5   True  Cool    High             No
6   True   Hot    High             No
7   True   Hot  Normal            Yes
8  False  Cool  Normal            Yes
9  False  Cool    High            Yes
```

```
print("Given Data after Encoding - \n",data,"\n")
X = data[['a1_n','a2_n','a3_n']]
print("X - Values\n",X,"\n")
y = data['classification']
print("Y - Values\n",y,"\n")
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.3)
model = DecisionTreeClassifier(criterion='entropy')
model.fit(X_train,y_train)
```

✓ 0.0s

```
Output exceeds the size limit. Open the full output data in a text editor
Given Data after Encoding -
      a1    a2      a3 classification  a1_n  a2_n  a3_n
0   True   Hot    High             No     1     1     0
1   True   Hot    High             No     1     1     0
2  False   Hot    High            Yes     0     1     0
3  False  Cool  Normal            Yes     0     0     1
4  False  Cool  Normal            Yes     0     0     1
5   True  Cool    High             No     1     0     0
6   True   Hot    High             No     1     1     0
7   True   Hot  Normal            Yes     1     1     1
8  False  Cool  Normal            Yes     0     0     1
9  False  Cool    High            Yes     0     0     0

X - Values
   a1_n  a2_n  a3_n
0     1     1     0
1     1     1     0
2     0     1     0
3     0     0     1
4     0     0     1
5     1     0     0
6     1     1     0
7     1     1     1
8     0     0     1
9     0     0     0
...
8    Yes
9    Yes
Name: classification, dtype: object
```

```
DecisionTreeClassifier
DecisionTreeClassifier(criterion='entropy')
```

```
print("Values predicted from test dataset - ",model.predict(X_test))
print("Original Values of test dataset - ",y_test.values)
print("Accuracy of Model",model.score(X_test,y_test))
```

✓ 0.0s

```
Values predicted from test dataset -  ['No' 'No' 'No']
Original Values of test dataset -  ['No' 'No' 'No']
Accuracy of Model 1.0
```

2. <u>CART ALGORITHM</u>

<u>AIM:</u> To perform CART Algorithm

<u>ALGORITHM:</u>

```
Create a Root node for the decision tree
   If all examples have the same value for Target_Attribute,
return the single-node tree Root, with label = this value
   If the Attributes list is empty, return the single-node tree
Root, with label = mean value of Target_Attribute in the examples
   Otherwise, choose the best attribute to split the examples
      The best attribute is the one that minimizes the cost of the
split
      The cost can be calculated using the Gini index or the
misclassification error rate for classification problems
      The cost can be calculated using the mean squared error for
regression problems
   Add a new decision node Root, corresponding to the best
attribute
   For each possible value of the best attribute, create a new
subset of examples that have that value
      If the subset is empty, add a new leaf node with label =
mean value of Target_Attribute in the examples
      Else, call CART recursively with the subset as the new
examples and repeat from step 2
   Return Root
```

TRAINING EXAMPLE:

	age	job	house	credit	loan_approved
0	young	False	No	Fair	No
1	young	False	No	Good	No
2	young	True	No	Good	Yes
3	young	True	Yes	Fair	Yes
4	young	False	No	Fair	No
5	middle	False	No	Fair	No
6	middle	False	No	Good	No
7	middle	True	Yes	Good	Yes
8	middle	False	Yes	Excellent	Yes
9	middle	False	Yes	Excellent	Yes
10	old	False	Yes	Excellent	Yes
11	old	False	Yes	Good	Yes
12	old	True	No	Good	Yes
13	old	True	No	Excellent	Yes
14	old	False	No	Fair	No

CODE:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
data = pd.read_csv('datasets\CART.csv')
print("Sample Dataset - \n",data,"\n")
le_age = LabelEncoder()
data['age_n'] = le_age.fit_transform(data['age'])
le_job = LabelEncoder()
data['job_n'] = le_job.fit_transform(data['job'])
le_house = LabelEncoder()
data['house_n'] = le_house.fit_transform(data['house'])
le_credit = LabelEncoder()
data['credit_n'] = le_credit.fit_transform(data['credit'])
le_loan = LabelEncoder()
data['loan_n'] = le_loan.fit_transform(data['loan_approved'])
print("Given Data after Encoding - \n",data,"\n")
X = data[['age_n','job_n','house_n','credit_n']]
print("X - Values\n",X,"\n")
y = data['loan_approved']
print("Y - Values\n",y,"\n")
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)
```

```
model = DecisionTreeClassifier(criterion='gini')
model.fit(X_train,y_train)
print("Pedicted Values - ",model.predict(X_test))
print("Original Values of Predicted Values - ",y_test.values)
print("Predicting for - [young,False,No,Good] - ",model.predict([[2,0,0,2]]))
print("Accuracy of Model",model.score(X_test,y_test))
```

OUTPUT:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.preprocessing import LabelEncoder
```

✓ 1.0s

```
data = pd.read_csv('CART.csv')
print("Sample Dataset - \n",data,"\n")
le_age = LabelEncoder()
data['age_n'] = le_age.fit_transform(data['age'])
le_job = LabelEncoder()
data['job_n'] = le_job.fit_transform(data['job'])
le_house = LabelEncoder()
data['house_n'] = le_house.fit_transform(data['house'])
le_credit = LabelEncoder()
data['credit_n'] = le_credit.fit_transform(data['credit'])
le_loan = LabelEncoder()
data['loan_n'] = le_loan.fit_transform(data['loan_approved'])
print("Given Data after Encoding - \n",data,"\n")
```

✓ 0.0s

```
Output exceeds the size limit. Open the full output data in a text editor
Sample Dataset -
       age    job house     credit loan_approved
0   young  False    No       Fair            No
1   young  False    No       Good            No
2   young   True    No       Good           Yes
3   young   True   Yes       Fair           Yes
4   young  False    No       Fair            No
5  middle  False    No       Fair            No
6  middle  False    No       Good            No
7  middle   True   Yes       Good           Yes
8  middle  False   Yes  Excellent           Yes
9  middle  False   Yes  Excellent           Yes
10    old  False   Yes  Excellent           Yes
11    old  False   Yes       Good           Yes
```

```python
X = data[['age_n','job_n','house_n','credit_n']]
print("X - Values\n",X,"\n")
y = data['loan_approved']
print("Y - Values\n",y,"\n")
X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.25)
model = DecisionTreeClassifier(criterion='gini')
```

[4] ✓ 0.0s

```
Output exceeds the size limit. Open the full output data in a text editor
X - Values
     age_n  job_n  house_n  credit_n
0       2      0        0         1
1       2      0        0         2
2       2      1        0         2
3       2      1        1         1
4       2      0        0         1
5       0      0        0         1
6       0      0        0         2
7       0      1        1         2
8       0      0        1         0
9       0      0        1         0
10      1      0        1         0
11      1      0        1         2
12      1      1        0         2
13      1      1        0         0
14      1      0        0         1

Y - Values
 0      No
1      No
2     Yes
3     Yes
4      No
5      No
...
13    Yes
14     No
Name: loan_approved, dtype: object
```

```
model.fit(X_train,y_train)
print("Pedicted Values - ",model.predict(X_test))
print("Original Values of Predicted Values - ",y_test.values)
print("Predicting for - [young,False,No,Good] - ",model.predict([[2,0,0,2]]))
print("Accuracy of Model",model.score(X_test,y_test))
```

✓ 0.0s

```
Pedicted Values -  ['No' 'No' 'Yes' 'Yes']
Original Values of Predicted Values -  ['No' 'No' 'Yes' 'Yes']
Predicting for - [young,False,No,Good] -  ['No']
Accuracy of Model 1.0
```

3. **KNN Algorithm**

AIM: To perform KNN Algorithm

ALGORITHM:

```
For each instance in the Training_Set, calculate its distance from
the Test_Instance
      The distance can be calculated using Euclidean, Manhattan,
or any other distance metric
   Sort the Training_Set instances based on their distances from
the Test_Instance
   Select the k nearest neighbors from the sorted Training_Set
   If the problem is a classification problem, predict the class
of the Test_Instance as the majority class among the k nearest
neighbors
   If the problem is a regression problem, predict the value of
the Test_Instance as the mean value of the k nearest neighbors
   Return the predicted class or value
```

TRAINING EXAMPLE:

	Height	Weight	Class
0	167	51	Underweight
1	182	62	Normal
2	176	69	Normal
3	173	64	Normal
4	172	65	Normal
5	174	56	Underweight
6	169	58	Normal
7	173	57	Normal
8	170	55	Normal

CODE:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
dataset
print("x",x)
print("y",y)
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(x, y)
predictions = knn.predict([[170,57]])
print("Prediction for - [Height=170, Weight=57] for k=3 is ",predictions)
print("Accuracy of Model",knn.score(x,y))
```

OUTPUT:

```
import numpy as np
import pandas as pd
from sklearn.neighbors import KNeighborsClassifier
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
dataset
```

✓ 0.0s

	Height	Weight	Class
0	167	51	Underweight
1	182	62	Normal
2	176	69	Normal
3	173	64	Normal
4	172	65	Normal
5	174	56	Underweight
6	169	58	Normal
7	173	57	Normal
8	170	55	Normal

```
print("x",x)
print("y",y)
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(x, y)
predictions = knn.predict([[170,57]])
print("Prediction for - [Height=170, Weight=57] for k=3 is ",predictions)
print("Accuracy of Model",knn.score(x,y))
```

✓ 0.0s


```
print("x",x)
print("y",y)
knn = KNeighborsClassifier(n_neighbors = 4)
knn.fit(x, y)
predictions = knn.predict([[170,57]])
print("Prediction for - [Height=170, Weight=57] for k=3 is ",predictions)
print("Accuracy of Model",knn.score(x,y))
```

✓ 0.0s


```
x [[167  51]
 [182  62]
 [176  69]
 [173  64]
 [172  65]
 [174  56]
 [169  58]
 [173  57]
 [170  55]]
y ['Underweight' 'Normal' 'Normal' 'Normal' 'Normal' 'Underweight' 'Normal'
 'Normal' 'Normal']
Prediction for - [Height=170, Weight=57] for k=3 is  ['Normal']
Accuracy of Model 0.7777777777777778
```

Regno: 20BCE2961

Submitted To: Jaisankar N

4. KNN ALGORITHM WITHOUT PYTHON LIBRARIES

AIM: To perform KNN Algorithm without python libraries.

ALGORITHM:

```
For each instance in the Training_Set, calculate its distance from
the Test_Instance
     The distance can be calculated using Euclidean, Manhattan, or
     any other distance metric
     Sort the Training_Set instances based on their distances from
     the Test_Instance
     Select the k nearest neighbors from the sorted Training_Set
     If the problem is a classification problem, predict the class
     of the Test_Instance as the majority class among the k
     nearest neighbors
     If the problem is a regression problem, predict the value of
     the Test_Instance as the mean value of the k nearest
     neighbors
Return the predicted class or value
```

TRAINING EXAMPLE:

	Height	Weight	Class
0	167	51	Underweight
1	182	62	Normal
2	176	69	Normal
3	173	64	Normal
4	172	65	Normal
5	174	56	Underweight
6	169	58	Normal
7	173	57	Normal
8	170	55	Normal

CODE:

```
import numpy as np
import pandas as pd
import scipy.spatial
```

```
import math
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
print("x",x)
print("y",y)
def most_frequent(List):
    counter = 0
    num = List[0]
    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency> counter):
            counter = curr_frequency
            num = i
    return num
def cal_distance(x,y,x_pred,y_pred):
    distance = math.sqrt((x-x_pred)**2+(y-y_pred)**2)
    return distance
def knn(x,k):
    x_pred = 170
    y_pred = 57
    dist = []
    res = []
    for i in range(len(x)):
        dist.append(cal_distance(int(x[i][0]),int(x[i][1]),x_pred,y_pred))
    ranks = pd.Series(dist).rank().tolist()
    for i in range(1,k+1):
        res.append(y[ranks.index(i)])
    return most_frequent(res)

print("The result for Height = 170 and Weight = 57 is ", knn(x,3))
```

OUTPUT

```
import numpy as np
import pandas as pd
import scipy.spatial
import math
dataset = pd.read_csv('knn.csv')
x=dataset.iloc[:,0:-1].values
y=dataset.iloc[:,-1].values
print("x",x)
print("y",y)
```

✓ 0.0s

```
x [[167  51]
 [182  62]
 [176  69]
 [173  64]
 [172  65]
 [174  56]
 [169  58]
 [173  57]
 [170  55]]
y ['Underweight' 'Normal' 'Normal' 'Normal' 'Normal' 'Underweight' 'Normal'
 'Normal' 'Normal']
```

```
def most_frequent(List):
    counter = 0
    num = List[0]
    for i in List:
        curr_frequency = List.count(i)
        if(curr_frequency> counter):
            counter = curr_frequency
            num = i
    return num
def cal_distance(x,y,x_pred,y_pred):
    distance = math.sqrt((x-x_pred)**2+(y-y_pred)**2)
    return distance
def knn(x,k):
    x_pred = 170
    y_pred = 57

    dist = []
    res = []
    for i in range(len(x)):
        dist.append(cal_distance(int(x[i][0]),int(x[i][1]),x_pred,y_pred))
    ranks = pd.Series(dist).rank().tolist()
    for i in range(1,k+1):
        res.append(y[ranks.index(i)])
    return most_frequent(res)
print("The result for Height = 170 and Weight = 57 is ", knn(x,3))
```

✓ 0.0s

```
The result for Height = 170 and Weight = 57 is  Normal
```

5. NAÏVE BAYES ALGORITHM

AIM: To perform Naïve Bayes Algorithm

ALGORITHM:

```
For each class in the target attribute, calculate the likelihood
of each attribute value given the class
      The likelihood can be calculated using the frequency of each
      value in the Training_Set for that class
      The likelihood can be smoothed using Laplace smoothing to
      avoid zero probabilities
      Calculate the prior probability of each class in the target
      attribute
      The prior probability can be calculated as the frequency of
      each class in the Training_Set
      Calculate the posterior probability of each class given the
      attribute values of the Test_Instance
      The posterior probability is calculated as the product of
      the likelihood of each attribute value given the class and
      the prior probability of the class
      Predict the class of the Test_Instance as the class with the
      highest posterior probability
Return the predicted class
```

TRAINING EXAMPLE:

Sample Dataset -

	Day	Outlook	Temperature	Humidity	Wind	PlayTennis
0	D1	Sunny	Hot	High	Weak	No
1	D2	Sunny	Hot	High	Strong	No
2	D3	Overcast	Hot	High	Weak	Yes
3	D4	Rain	Mild	High	Weak	Yes
4	D5	Rain	Cool	Normal	Weak	Yes
5	D6	Rain	Cool	Normal	Strong	No
6	D7	Overcast	Cool	Normal	Strong	Yes
7	D8	Sunny	Mild	High	Weak	No
8	D9	Sunny	Cool	Normal	Weak	Yes
9	D10	Rain	Mild	Normal	Weak	Yes
10	D11	Sunny	Mild	Normal	Strong	Yes
11	D12	Overcast	Mild	High	Strong	Yes
12	D13	Overcast	Hot	Normal	Weak	Yes
13	D14	Rain	Mild	High	Strong	No

CODE:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
from sklearn.preprocessing import LabelEncoder
dataset = pd.read_csv('naive.csv')
print("Sample Dataset - \n",dataset,"\n")
le_outlook = LabelEncoder()
dataset['outlook_n'] = le_outlook.fit_transform(dataset['Outlook'])
le_temperature = LabelEncoder()
dataset['temperature_n'] = le_temperature.fit_transform(dataset['Temperature'])
le_humidity = LabelEncoder()
dataset['humidity_n'] = le_humidity.fit_transform(dataset['Humidity'])
le_wind = LabelEncoder()
dataset['wind_n'] = le_wind.fit_transform(dataset['Wind'])
print("Given Data after Encoding - \n",dataset,"\n")
x = dataset[['outlook_n','temperature_n','humidity_n','wind_n']]
print("X - Values\n",x,"\n")
y = dataset['PlayTennis']
print("Y - Values\n",y,"\n")
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15,
random_state = 0)
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x, y)
y_pred = gnb.predict(x_test)
print("Testing values for play tennis\n",y_test)
print("Predicted values for play tennis",y_pred)
```

OUTPUT:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
dataset = pd.read_csv('naive.csv')
print("Sample Dataset - \n",dataset,"\n")
le_outlook = LabelEncoder()
dataset['outlook_n'] = le_outlook.fit_transform(dataset['Outlook'])
le_temperature = LabelEncoder()
dataset['temperature_n'] = le_temperature.fit_transform(dataset['Temperature'])
le_humidity = LabelEncoder()
dataset['humidity_n'] = le_humidity.fit_transform(dataset['Humidity'])
le_wind = LabelEncoder()
dataset['wind_n'] = le_wind.fit_transform(dataset['Wind'])
print("Given Data after Encoding - \n",dataset,"\n")
x = dataset[['outlook_n','temperature_n','humidity_n','wind_n']]
print("X - Values\n",x,"\n")
y = dataset['PlayTennis']
print("Y - Values\n",y,"\n")
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.15, random_state = 0)
```

✓ 2.0s

```
Output exceeds the size limit. Open the full output data in a text editor
Sample Dataset -
     Day   Outlook Temperature Humidity    Wind PlayTennis
0    D1     Sunny         Hot     High    Weak         No
1    D2     Sunny         Hot     High  Strong         No
2    D3  Overcast         Hot     High    Weak        Yes
3    D4      Rain        Mild     High    Weak        Yes
4    D5      Rain        Cool   Normal    Weak        Yes
5    D6      Rain        Cool   Normal  Strong         No
6    D7  Overcast        Cool   Normal  Strong        Yes
7    D8     Sunny        Mild     High    Weak         No
8    D9     Sunny        Cool   Normal    Weak        Yes
9   D10      Rain        Mild   Normal    Weak        Yes
10  D11     Sunny        Mild   Normal  Strong        Yes
11  D12  Overcast        Mild     High  Strong        Yes
12  D13  Overcast         Hot   Normal    Weak        Yes
13  D14      Rain        Mild     High  Strong         No
```

```
Given Data after Encoding -
     Day   Outlook Temperature Humidity    Wind PlayTennis  outlook_n  \
0    D1     Sunny         Hot     High    Weak         No          2
1    D2     Sunny         Hot     High  Strong         No          2
2    D3  Overcast         Hot     High    Weak        Yes          0
3    D4      Rain        Mild     High    Weak        Yes          1
4    D5      Rain        Cool   Normal    Weak        Yes          1
5    D6      Rain        Cool   Normal  Strong         No          1
...
12    Yes
13     No
Name: PlayTennis, dtype: object
```

```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
gnb.fit(x, y)
y_pred = gnb.predict(x_test)
print("Testing values for play tennis\n",y_test)
print("Predicted values for play tennis",y_pred)
```

✓ 0.0s

```
Testing values for play tennis
 8    Yes
6    Yes
4    Yes
Name: PlayTennis, dtype: object
Predicted values for play tennis ['Yes' 'Yes' 'Yes']
```

MACHINE LEARNING LABORATORY

DIGITAL ASSIGNMENT 3

Name: SHASHANK VENKAT

Register Number: 20BCE2961

Slot: L39+L40

Course Code: CSE 4020

Submitted To: Prof. JAISANKAR N

Number of Pages: 26

1. LINEAR SVM

AIM: To perform Linear Algorithm

ALGORITHM:

```
Initialize the weight vector w to zeros with dimensions (n x 1)
and the bias term b to zero.
Set the learning rate alpha to a small value.
Repeat until convergence:
For each training example (xi, yi) in the training data:
Compute the margin yi(w^Txi + b) = yi(w1x1 + w2x2 + ... + wnxn +
b).
If the margin is less than 1, update the weight vector and bias
term:
w <- w + alpha * (yixi - 2Cw)
b <- b + alpha * yi
If the margin is greater than or equal to 1, update the weight
vector only:
w <- w + alpha * (-2Cw)
Once convergence is reached, return the weight vector w and bias
term b.
```

TRAINING EXAMPLE:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

CODE:

```
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
dataset=pd.read_csv('SVM.csv')
x=dataset.iloc[:, 2:-1].values
y=dataset.iloc[:, -1].values
dataset

from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.27,random_state=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)

from sklearn.svm import SVC
classifier=SVC(kernel='linear',random_state=0)
classifier.fit(x_train,y_train)
classifier.predict(sc.transform([[30,87000]]))
y_pred=classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))


from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print('Accuracy Score: ',accuracy_score(y_test,y_pred))


from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
```

```
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

OUTPUT

```
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.27,random_state=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

[4] ✓ 2.2s Python

```
from sklearn.svm import SVC
classifier=SVC(kernel='linear',random_state=0)
classifier.fit(x_train,y_train)
classifier.predict(sc.transform([[30,87000]]))
y_pred=classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

[5] ✓ 0.2s Python

```
Output exceeds the size limit. Open the full output data in a text editor
[[0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
 [1 1]
 [0 0]
 [0 0]
 [0 0]
 [0 0]
```

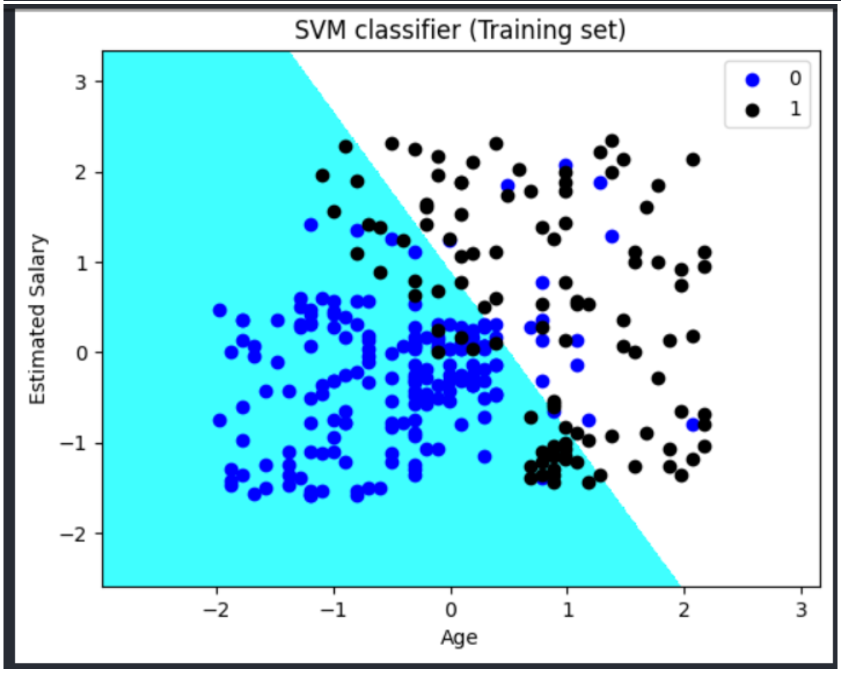
SVM_linear.ipynb

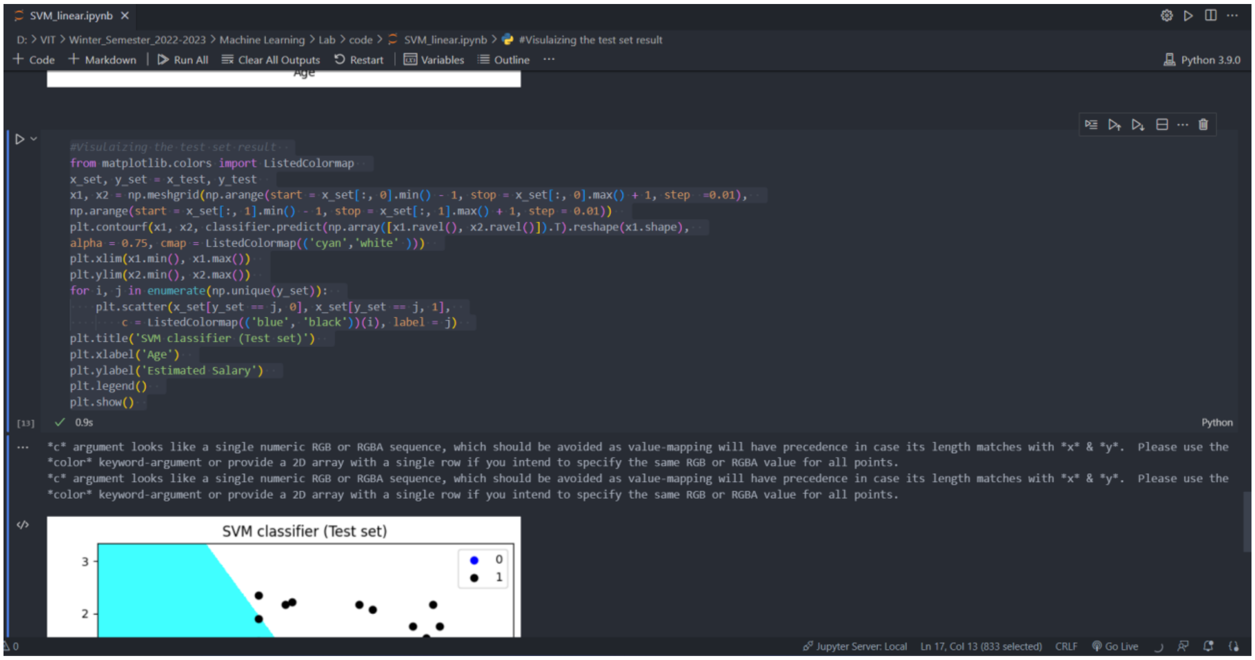
D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_linear.ipynb > #Visulaizing the test set result

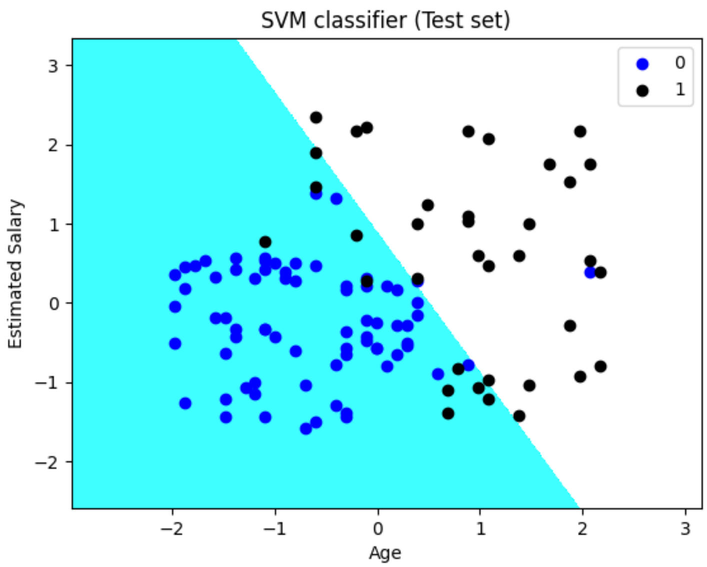
```python
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
```

```
[[70  2]
 [10 26]]
Accuracy Score:  0.8888888888888888
```

```python
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```




SVM_linear.ipynb
D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_linear.ipynb > #Visulaizing the test set result
+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...
Python 3.9.0
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
[13] 0.9s
Python
c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
c argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence in case its length matches with *x* & *y*. Please use the *color* keyword-argument or provide a 2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
SVM classifier (Test set)
Jupyter Server: Local
Ln 17, Col 13 (833 selected)
CRLF
Go Live


SVM classifier (Test set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−2
−1
0
1
2
3

2. NON-LINEAR SVM ALGORITHM

AIM: To perform NON-LINEAR SVM Algorithm

ALGORITHM:

```
Compute the kernel matrix K with dimensions (m x m) where K(i,j) =
kernel(xi, xj) for all i, j in [1, m].
Initialize the vector of Lagrange multipliers alpha to zeros with
dimensions (m x 1) and the bias term b to zero.
Set the learning rate eta to a small value.
Repeat until convergence:
For each training example i in the training data:
Compute the predicted label yi:
yi = sign(sum(alpha(j)*y(j)*kernel(xi, xj)) + b)
Compute the error for example i:
E(i) = yi - y(i)
If E(i) is not within the tolerance range [-tolerance, tolerance],
update alpha(i):
alpha(i) <- alpha(i) - eta * (E(i) + C * y(i) *
sum(alpha(j)*y(j)*kernel(xi, xj) for j != i))
Compute the bias term b:
Find the set of support vectors with non-zero Lagrange multipliers
alpha:
support_vectors = {i : alpha(i) > 0}
For each support vector i, compute the bias term bi:
bi = y(i) - sum(alpha(j)*y(j)*kernel(xi, xj) for j in
support_vectors)
Set the final bias term to the average of the support vector
biases:
b = mean(bi for i in support_vectors)
Once convergence is reached, return the vector of Lagrange
multipliers alpha and bias term b.
```

TRAINING EXAMPLE:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

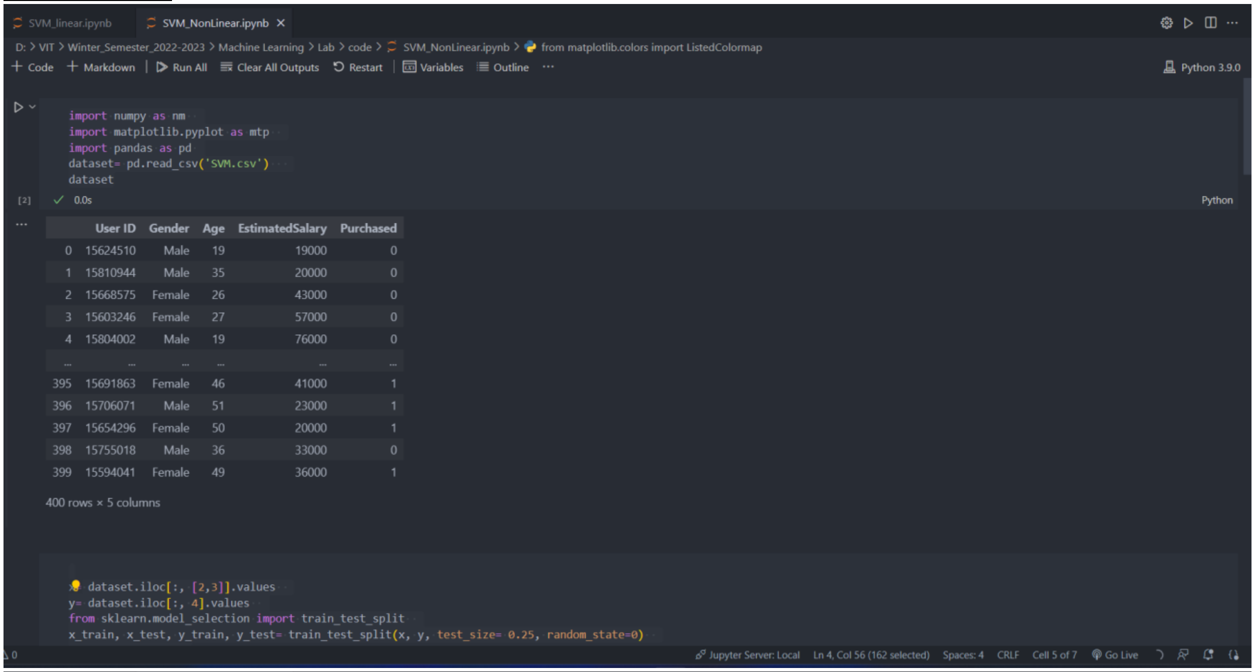
CODE:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
dataset= pd.read_csv('SVM.csv')
dataset

x= dataset.iloc[:, [2,3]].values
y= dataset.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
stx= StandardScaler()
x_train= stx.fit_transform(x_train)
x_test= stx.transform(x_test)
from sklearn.svm import SVC
classifier = SVC(kernel='poly', random_state=0)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
from sklearn.metrics import confusion_matrix  , accuracy_score
cm= confusion_matrix(y_test, y_pred)
cm
```

```
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

OUTPUT:

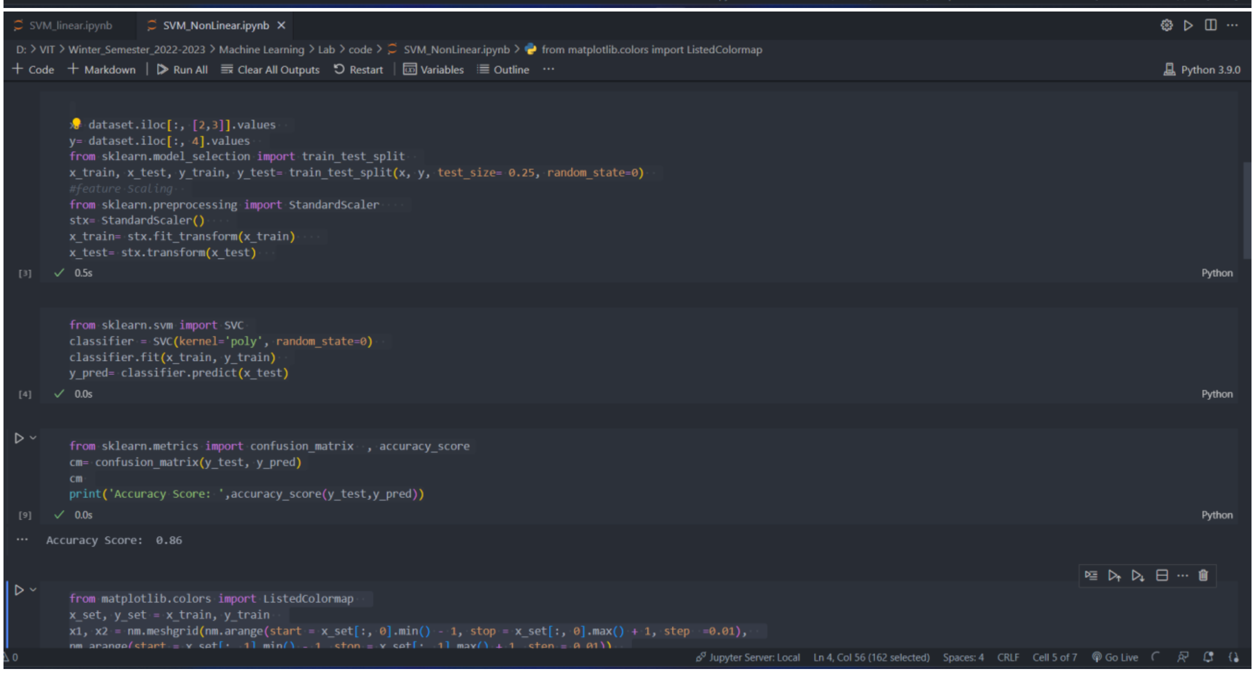


```python
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
dataset= pd.read_csv('SVM.csv')
dataset
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```python
dataset.iloc[:, [2,3]].values
y= dataset.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)
```



```python
dataset.iloc[:, [2,3]].values
y= dataset.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)
#feature Scaling
from sklearn.preprocessing import StandardScaler
stx= StandardScaler()
x_train= stx.fit_transform(x_train)
x_test= stx.transform(x_test)
```

```python
from sklearn.svm import SVC
classifier = SVC(kernel='poly', random_state=0)
classifier.fit(x_train, y_train)
y_pred= classifier.predict(x_test)
```

```python
from sklearn.metrics import confusion_matrix , accuracy_score
cm= confusion_matrix(y_test, y_pred)
cm
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
```

Accuracy Score: 0.86

```python
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
```

SVM_linear.ipynb   SVM_NonLinear.ipynb ×

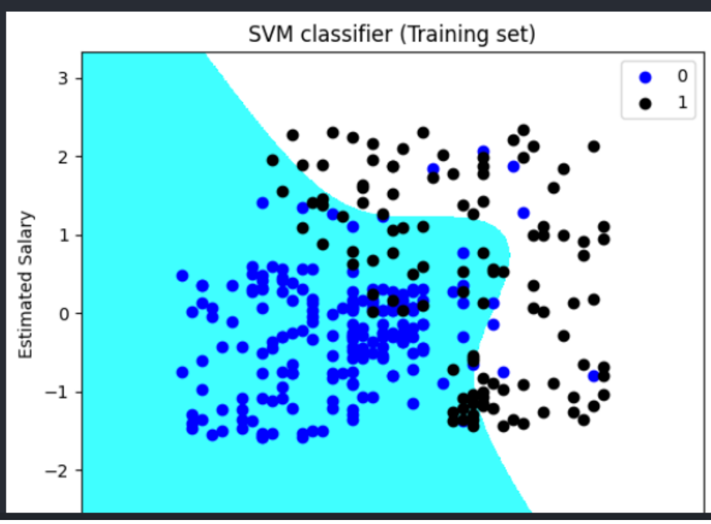
D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_NonLinear.ipynb > from matplotlib.colors import ListedColormap

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ···

```
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('SVM classifier (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

[6] ✓ 1.0s

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



SVM_linear.ipynb   SVM_NonLinear.ipynb ×

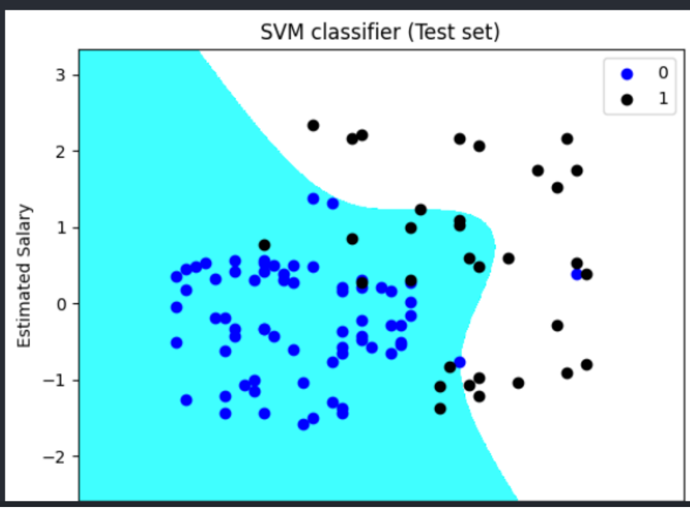
D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_NonLinear.ipynb > from matplotlib.colors import ListedColormap

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ···

```python
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = nm.meshgrid(nm.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
nm.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
mtp.contourf(x1, x2, classifier.predict(nm.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
mtp.xlim(x1.min(), x1.max())
mtp.ylim(x2.min(), x2.max())
for i, j in enumerate(nm.unique(y_set)):
    mtp.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
mtp.title('SVM classifier (Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

[7] ✓ 0.9s

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence i
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided as value-mapping will have precedence i
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```



Regno: 20BCE2961

Submitted To: Jaisankar N

3. KERNEL SVM Algorithm

AIM: To perform Kernel SVM Algorithm

ALGORITHM:

```
Compute the kernel matrix K with dimensions (m x m) where K(i,j) =
kernel(xi, xj) for all i, j in [1, m].
Initialize the vector of Lagrange multipliers alpha to zeros with
dimensions (m x 1) and the bias term b to zero.
Set the learning rate eta to a small value.
Repeat until convergence:
For each training example i in the training data:
Compute the predicted label yi:
yi = sign(sum(alpha(j)*y(j)*K(i,j)) + b)
Compute the error for example i:
E(i) = yi - y(i)
If E(i) is not within the tolerance range [-tolerance, tolerance], update
alpha(i):
alpha(i) <- alpha(i) - eta * (E(i) + C * y(i) * sum(alpha(j)*y(j)*K(i,j)
for j != i))
Compute the bias term b:
Find the set of support vectors with non-zero Lagrange multipliers alpha:
support_vectors = {i : alpha(i) > 0}
For each support vector i, compute the bias term bi:
bi = y(i) - sum(alpha(j)*y(j)*K(i,j) for j in support_vectors)
Set the final bias term to the average of the support vector biases:
b = mean(bi for i in support_vectors)
Once convergence is reached, return the vector of Lagrange multipliers
alpha and bias term b.
```

TRAINING EXAMPLE:

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

CODE:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
dataset=pd.read_csv('SVM.csv')
x=dataset.iloc[:, 2:-1].values
y=dataset.iloc[:, -1].values
dataset
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.27,random_state=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
from sklearn.svm import SVC
classifier=SVC(kernel='rbf',random_state=0)
classifier.fit(x_train,y_train)
classifier.predict(sc.transform([[30,87000]]))
y_pred=classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1),
y_test.reshape(len(y_test),1)),1))
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
```

```
print(cm)
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:,
0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step =
0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(),
x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

OUTPUT:

SVM_linear.ipynb | SVM_NonLinear.ipynb | SVM_Kernel.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_Kernel.ipynb > #Visulaizing the test set result

+ Code + Markdown | Run All | Clear All Outputs | Restart | Variables | Outline ...

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

[4] ✓ 0.0s

```python
dataset=pd.read_csv('SVM.csv')
x=dataset.iloc[:, 2:-1].values
y=dataset.iloc[:, -1].values
dataset
```

[5] ✓ 0.0s

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
3	15603246	Female	27	57000	0
4	15804002	Male	19	76000	0
...	...	...	...	...	...
395	15691863	Female	46	41000	1
396	15706071	Male	51	23000	1
397	15654296	Female	50	20000	1
398	15755018	Male	36	33000	0
399	15594041	Female	49	36000	1

400 rows × 5 columns

```python
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.27,random_state=0)
sc=StandardScaler()
x_train=sc.fit_transform(x_train)
x_test=sc.transform(x_test)
```

[6] ✓ 0.0s

SVM_linear.ipynb | SVM_NonLinear.ipynb | SVM_Kernel.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_Kernel.ipynb > #Visulaizing the test set r

+ Code + Markdown | Run All | Clear All Outputs | Restart | Variables | Outline ...

```python
from sklearn.svm import SVC
classifier=SVC(kernel='rbf',random_state=0)
classifier.fit(x_train,y_train)
```

[7] ✓ 0.0s

```
SVC
SVC(random_state=0)
```

```python
classifier.predict(sc.transform([[30,87000]]))
```

[8] ✓ 0.0s

```
array([0], dtype=int64)
```

```python
y_pred=classifier.predict(x_test)
print(np.concatenate((y_pred.reshape(len(y_pred),1), y_test.reshape(len(y_test),1)),1))
```

[9] ✓ 0.0s

Regno: 20BCE2961

Submitted To: Jaisankar N

SVM_linear.ipynb SVM_NonLinear.ipynb SVM_Kernel.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_Kernel.ipynb > #Visulaizing the test set result

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...

```python
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
print(cm)
```

[10] ✓ 0.0s

```
[[67  5]
 [ 3 33]]
```

```python
print('Accuracy Score: ',accuracy_score(y_test,y_pred))
```

[11] ✓ 0.0s

```
Accuracy Score:  0.9259259259259259
```

```python
from matplotlib.colors import ListedColormap
x_set, y_set = x_train, y_train
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan', 'white')))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

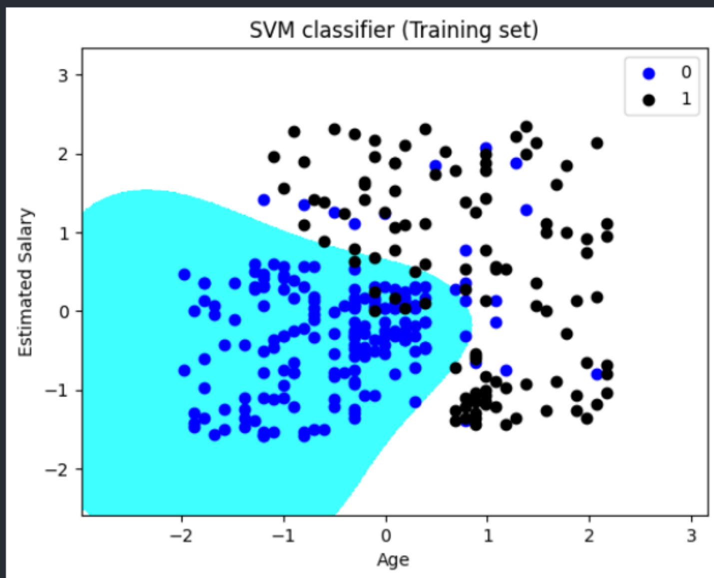
[12] ✓ 2.8s

SVM_linear.ipynb SVM_NonLinear.ipynb SVM_Kernel.ipynb ×

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_Kernel.ipynb > from matplotlib.colors import ListedColormap

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ···

```
2D array with a single row if you intend to specify the same RGB or RGBA value for all points.
```




```
#Visulaizing the test set result
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0].max() + 1, step  =0.01),
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0.01))
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).reshape(x1.shape),
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
```

Regno: 20BCE2961

Submitted To: Jaisankar N

SVM_linear.ipynb SVM_NonLinear.ipynb SVM_Kernel.ipynb

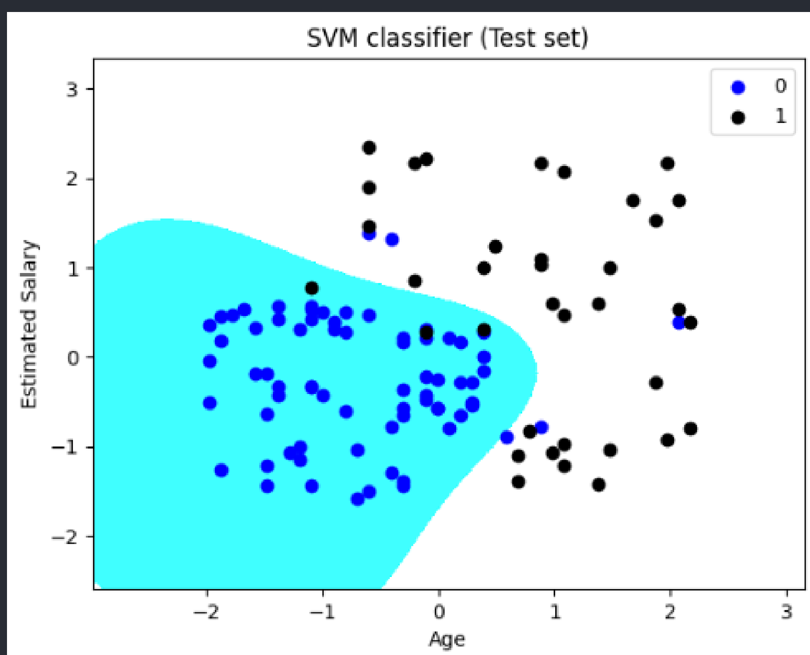
D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > SVM_Kernel.ipynb > #Visulaizing

Code Markdown Run All Clear All Outputs Restart Variables Outline

```python
from matplotlib.colors import ListedColormap
x_set, y_set = x_test, y_test
x1, x2 = np.meshgrid(np.arange(start = x_set[:, 0].min() - 1, stop = x_set[:, 0
np.arange(start = x_set[:, 1].min() - 1, stop = x_set[:, 1].max() + 1, step = 0
plt.contourf(x1, x2, classifier.predict(np.array([x1.ravel(), x2.ravel()]).T).r
alpha = 0.75, cmap = ListedColormap(('cyan','white' )))
plt.xlim(x1.min(), x1.max())
plt.ylim(x2.min(), x2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(x_set[y_set == j, 0], x_set[y_set == j, 1],
        c = ListedColormap(('blue', 'black'))(i), label = j)
plt.title('SVM classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

[13] ✓ 2.6s

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoi
specify the same RGB or RGBA value for all points.
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoi
specify the same RGB or RGBA value for all points.
```



4. K MEANS CLUSTERING

AIM: To perform K Means Clustering Algorithm.

ALGORITHM:

```
Initialize K centroids from the data points.
Repeat until convergence or max_iter is reached:
     Assign each data point to the closest centroid using the
     Euclidean distance metric:
          For each data point x(i), find the closest centroid:
               c(i) = argmin_j ||x(i) - centroids(j)||^2
     Update each centroid to the mean of the data points assigned
     to it:
          For each centroid j, update its position to the mean of
          the data points assigned to it:
               centroids(j) = mean(x(i) for i where c(i) = j)
Once convergence is reached, return the final matrix of cluster
centroids.
```

TRAINING EXAMPLE:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

CODE:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("kmeans.csv")
x = dataset.iloc[:,[3,4]].values
dataset
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
wcss_list= []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state= 42)
    kmeans.fit(X)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
#visulaizing the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'black', label
= 'Cluster 1-NonTarget') #for first cluster
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'grey', label
= 'Cluster 2-Careful') #for second cluster
plt.scatter(X[y_kmeans== 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'cyan', label =
'Cluster 3-Sensible') #for third cluster
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'magenta',
label = 'Cluster 4-Careless') #for fourth cluster
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'purple',
label = 'Cluster 5-Target') #for fifth cluster
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s =
300, c = 'red', label = 'Centroid')
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

OUTPUT

SVM_linear.ipynb | SVM_NonLinear.ipynb | SVM_Kernel.ipynb | KMeansClustering.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > KMeansClustering.ipynb > kmeans = KMeans(n_clusters

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

dataset = pd.read_csv("kmeans.csv")
X = dataset.iloc[:,[3,4]].values
dataset
```

[3] ✓ 0.0s

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40
...	...	...	...	...	...
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

200 rows × 5 columns

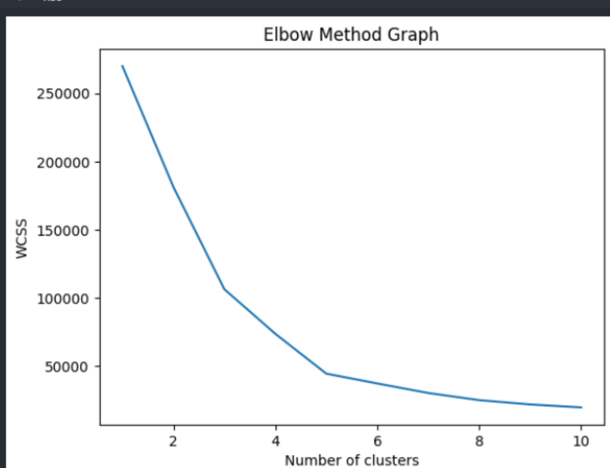
SVM_linear.ipynb | SVM_NonLinear.ipynb | SVM_Kernel.ipynb | KMeansClustering.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > KMeansClustering.ipynb > kmeans = KMeans(n_clusters = 5, i

+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...

```
from sklearn.cluster import KMeans
import warnings
warnings.filterwarnings("ignore")
wcss_list= []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++',random_state= 42)
    kmeans.fit(X)
    wcss_list.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss_list)
plt.title('Elbow Method Graph')
plt.xlabel('Number of clusters')
plt.ylabel('WCSS')
plt.show()
```

[4] ✓ 1.5s



SVM_linear.ipynb | SVM_NonLinear.ipynb | SVM_Kernel.ipynb | KMeansClustering.ipynb

D: > VIT > Winter_Semester_2022-2023 > Machine Learning > Lab > code > KMeansClustering.ipynb > from sklearn.clu

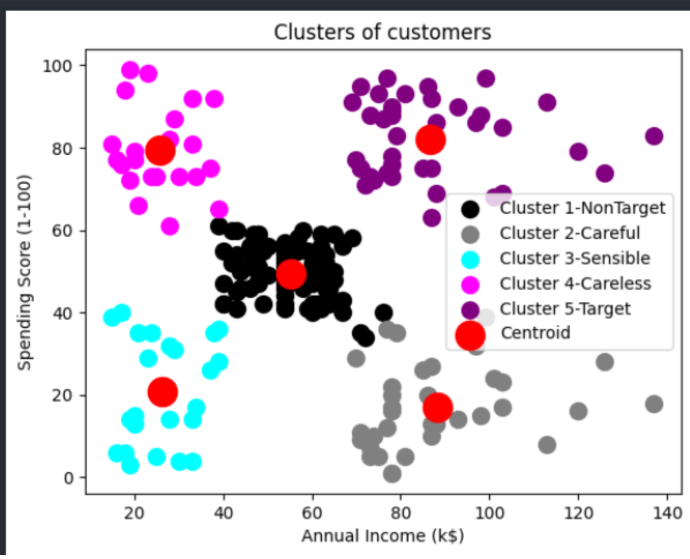
+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...

```python
kmeans = KMeans(n_clusters = 5, init = 'k-means++', random_state = 42)
y_kmeans = kmeans.fit_predict(X)
```

[5] ✓ 0.0s

```python
#visulaizing the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'black', label = 'C
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'grey', label = 'Cl
plt.scatter(X[y_kmeans== 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'cyan', label = 'Clu
plt.scatter(X[y_kmeans == 3, 0], X[y_kmeans == 3, 1], s = 100, c = 'magenta', label =
plt.scatter(X[y_kmeans == 4, 0], X[y_kmeans == 4, 1], s = 100, c = 'purple', label = '
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c =
plt.title('Clusters of customers')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

[9] ✓ 0.2s



5. **KMODES CLUSTERING**

AIM: To implement KModes Clustering algorithm on categorical data.

ALGORITHM:

```
Initialize k modes by randomly selecting k distinct objects
from the dataset.
Assign each object to the closest mode based on the
categorical distance (e.g., Hamming distance).
While not converged:
   Update the modes as the most frequent values of each
categorical feature among the objects assigned to the mode.
   Assign each object to the closest mode based on the
categorical distance.
Output the final clusters based on the assignments to the
modes.
```

TRAINING EXAMPLE:

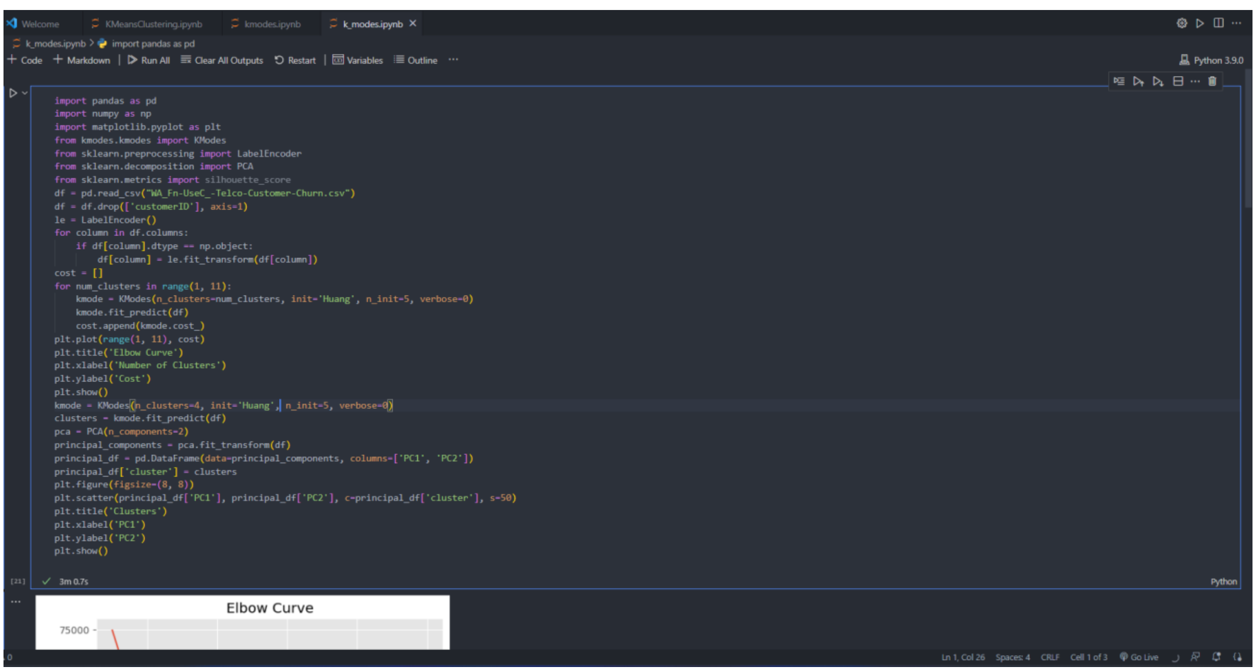
	gender	SeniorCitizen	Partner	Dependents	tenure	PhoneService	MultipleLines	InternetService	OnlineSecurity	OnlineBackup	DeviceProtection	TechSupport	StreamingTV	StreamingMovies
0	0	0	1	0	1	0	1	0	0	2	0	0	0	0
1	1	0	0	0	34	1	0	0	2	0	2	0	0	0
2	1	0	0	0	2	1	0	0	2	2	0	0	0	0
3	1	0	0	0	45	0	1	0	2	0	2	2	0	0
4	0	0	0	0	2	1	0	1	0	0	0	0	0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
038	1	0	1	1	24	1	2	0	2	0	2	2	2	2
039	0	0	1	1	72	1	2	1	0	2	2	0	2	2
040	0	0	1	1	11	0	1	0	2	0	0	0	0	0
041	1	1	1	0	4	1	2	1	0	0	0	0	0	0
042	1	0	0	0	66	1	0	1	2	0	2	2	2	2

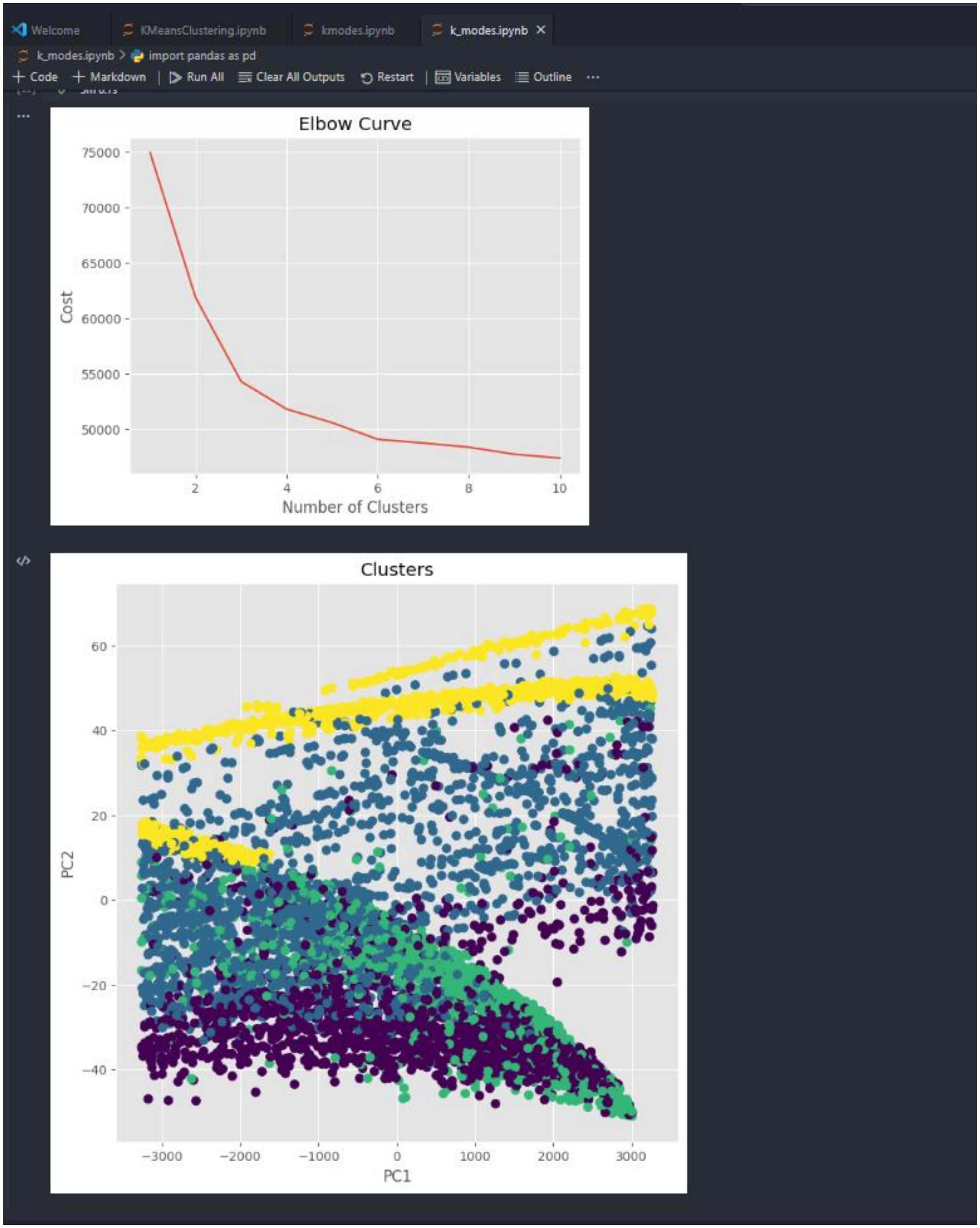
CODE:

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from kmodes.kmodes import KModes
from sklearn.preprocessing import LabelEncoder
from sklearn.decomposition import PCA
from sklearn.metrics import silhouette_score
df = pd.read_csv("WA_Fn-UseC_-Telco-Customer-Churn.csv")
df = df.drop(['customerID'], axis=1)
le = LabelEncoder()
for column in df.columns:
    if df[column].dtype == np.object:
        df[column] = le.fit_transform(df[column])
cost = []
for num_clusters in range(1, 11):
    kmode = KModes(n_clusters=num_clusters, init='Huang', n_init=5, verbose=0)
```

```
    kmode.fit_predict(df)
    cost.append(kmode.cost_)
plt.plot(range(1, 11), cost)
plt.title('Elbow Curve')
plt.xlabel('Number of Clusters')
plt.ylabel('Cost')
plt.show()
kmode = KModes(n_clusters=4, init='Huang', n_init=5, verbose=0)
clusters = kmode.fit_predict(df)
pca = PCA(n_components=2)
principal_components = pca.fit_transform(df)
principal_df = pd.DataFrame(data=principal_components, columns=['PC1', 'PC2'])
principal_df['cluster'] = clusters
plt.figure(figsize=(8, 8))
plt.scatter(principal_df['PC1'], principal_df['PC2'], c=principal_df['cluster'],
s=50)
plt.title('Clusters')
plt.xlabel('PC1')
plt.ylabel('PC2')
plt.show()
```

OUTPUT:




Welcome
KMeansClustering.ipynb
kmodes.ipynb
k_modes.ipynb
k_modes.ipynb > import pandas as pd
+ Code + Markdown | Run All Clear All Outputs Restart | Variables Outline ...
Elbow Curve
Cost
75000
70000
65000
60000
55000
50000
2
4
6
8
10
Number of Clusters
Clusters
PC2
60
40
20
0
−20
−40
−3000
−2000
−1000
0
1000
2000
3000
PC1