Mother Theresa Institute of Engineering and Technology

Dept of CSE/AI-DS/DS

PPDS Lab Machine Learning Algorithms

knn

June 15, 2023

1 packages

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

[3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

[4]:
```
#dataset = pd.read_csv("User_Data.csv")
#dataset.head()
#path = "https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.
↪data"
path = "Iris.txt"
headernames = ['sepal-length', 'sepal-width', 'petal-length', 'petal-width',␣
↪'Class']
```

[5]:
```
dataset = pd.read_csv(path, names = headernames)
dataset.head()
X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, 4].values
X
y
```

[5]:
```
array(['Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
```

```
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
'Iris-virginica', 'Iris-virginica'], dtype=object)
```

[6]:
```python
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.40)
```

[7]:
```python
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(X_train)
#scaler.fit(y_train)
X_train = scaler.transform(X_train)
#y_train = scaler.transform(y_train)
X_test = scaler.transform(X_test)
#y_test = scaler.transform(y_test)
X_train
X_test
```

```
[7]: array([[-0.06947825,  2.09537672, -1.50738027, -1.37138431],
            [-1.87324041, -0.12587069, -1.56464672, -1.50452841],
            [ 1.01277905, -0.12587069,  0.7832778 ,  1.42464195],
            [-0.79098311,  0.98475302, -1.33558091, -1.37138431],
            [ 0.53177581,  0.54050353,  1.24140941,  1.69093016],
            [-0.91123392,  0.98475302, -1.39284736, -1.2382402 ],
            [ 0.411525  , -1.90286861,  0.38241264,  0.35948909],
            [-1.03148473, -1.68074387, -0.30478478, -0.30623145],
            [-0.91123392,  1.4290025 , -1.33558091, -1.10509609],
            [-1.51248798,  0.09625405, -1.33558091, -1.37138431],
            [-0.18972906,  1.65112724, -1.22104801, -1.2382402 ],
            [ 0.53177581, -0.57012017,  0.72601135,  0.35948909],
            [ 0.53177581, -1.23649439,  0.6687449 ,  0.89206552],
            [-1.15173554, -1.45861913, -0.30478478, -0.30623145],
            [-0.55048149,  1.87325198, -1.22104801, -1.10509609],
            [ 0.53177581, -0.79224491,  0.61147845,  0.75892141],
            [ 0.411525  , -0.34799543,  0.26787974,  0.09320088],
            [-1.39223717,  0.31837879, -1.27831446, -1.37138431],
            [ 0.77227743, -0.12587069,  0.7832778 ,  1.02520963],
            [-0.06947825, -0.79224491,  0.03881393, -0.03994323],
            [ 0.17102338, -0.12587069,  0.55421199,  0.75892141],
            [ 0.53177581,  0.54050353,  0.49694554,  0.4926332 ],
            [-0.55048149, -0.12587069,  0.38241264,  0.35948909],
            [ 0.17102338, -1.90286861,  0.09608038, -0.30623145],
            [ 0.65202662, -0.57012017,  1.01234361,  1.29149784],
            [-0.30997987, -0.57012017,  0.61147845,  1.02520963],
            [ 1.13302986, -0.57012017,  0.55421199,  0.22634498],
            [ 0.17102338,  0.76262828,  0.38241264,  0.4926332 ],
            [ 1.85453473, -0.57012017,  1.29867587,  0.89206552],
            [-1.27198635,  0.09625405, -1.27831446, -1.37138431],
            [ 1.13302986, -0.12587069,  0.95507716,  1.15835373],
            [ 1.01277905,  0.54050353,  1.06961006,  1.15835373],
            [-1.27198635, -0.12587069, -1.39284736, -1.50452841],
            [ 0.65202662,  0.31837879,  0.84054425,  1.42464195],
            [ 0.65202662, -0.34799543,  0.26787974,  0.09320088],
            [-0.43023068, -1.68074387,  0.09608038,  0.09320088],
            [ 1.01277905,  0.09625405,  0.49694554,  0.35948909],
            [ 0.65202662,  0.09625405,  0.95507716,  0.75892141],
            [-1.27198635,  0.76262828, -1.10651511, -1.37138431],
            [-0.55048149,  0.76262828, -1.22104801, -1.37138431],
            [ 0.89252824, -0.12587069,  0.32514619,  0.22634498],
            [ 0.89252824, -0.34799543,  0.43967909,  0.09320088],
            [-1.39223717,  0.31837879, -1.45011382, -1.37138431],
            [-1.03148473,  1.20687776, -1.39284736, -1.37138431],
            [ 0.53177581, -0.34799543,  1.01234361,  0.75892141],
            [ 0.77227743, -0.12587069,  1.12687651,  1.29149784],
            [-0.18972906, -0.34799543,  0.21061328,  0.09320088],
```

```
       [-1.03148473, -0.12587069, -1.27831446, -1.37138431],
       [-1.7529896 , -0.34799543, -1.39284736, -1.37138431],
       [ 0.29127419, -0.12587069,  0.61147845,  0.75892141],
       [-1.15173554, -0.12587069, -1.39284736, -1.37138431],
       [ 1.49378229, -0.12587069,  1.18414296,  1.15835373],
       [-0.79098311,  0.76262828, -1.39284736, -1.37138431],
       [-1.51248798,  0.76262828, -1.39284736, -1.2382402 ],
       [-0.43023068, -1.45861913, -0.01845252, -0.17308734],
       [ 1.01277905, -0.12587069,  0.6687449 ,  0.6257773 ],
       [-0.55048149,  1.87325198, -1.45011382, -1.10509609],
       [-0.91123392,  0.98475302, -1.39284736, -1.37138431],
       [ 1.61403311, -0.12587069,  1.12687651,  0.4926332 ],
       [ 2.21528716, -1.01436965,  1.75680748,  1.42464195]])
```

[8]:
```
#from sklearn.neighbors import KNeighborsClassifier
#classifier = KNeighborsClassifier(n_neighbors = 8)
#classifier.fit(X_train, y_train)
from sklearn.neighbors import KNeighborsClassifier
classifier= KNeighborsClassifier(n_neighbors=3, metric='minkowski', p=2 )
classifier.fit(X_train,y_train)
```

[8]:
```
KNeighborsClassifier(n_neighbors=3)
```

[9]:
```
y_pred= classifier.predict(X_test)
```

[10]:
```
from sklearn.metrics import classification_report, confusion_matrix,␣
 ↪accuracy_score
result = confusion_matrix(y_test, y_pred)
print("Confusion Matrix:")
print(result)
result1 = classification_report(y_test, y_pred)
print("Classification Report:",)
print (result1)
result2 = accuracy_score(y_test,y_pred)
print("Accuracy:",result2)
```

```
Confusion Matrix:
[[22  0  0]
 [ 0 18  0]
 [ 0  2 18]]
Classification Report:
                 precision    recall  f1-score   support

    Iris-setosa       1.00      1.00      1.00        22
Iris-versicolor       0.90      1.00      0.95        18
 Iris-virginica       1.00      0.90      0.95        20
```

```
           accuracy                           0.97        60
          macro avg       0.97      0.97      0.96        60
       weighted avg       0.97      0.97      0.97        60

Accuracy: 0.9666666666666667
```

[ ]:

[ ]:

logistic-regression

June 15, 2023

[1]:
```
from pandas import *
from numpy import *
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

[2]:
```
dataset = pd.read_csv('User_Data.csv')
```

[3]:
```
dataset
```

[3]:
```
      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 x 5 columns]
```

[4]:
```
# input
x = dataset.iloc[:, [2, 3]].values

# output
y = dataset.iloc[:, 4].values
```

[5]:
```
x
```

[5]:
```
array([[    19,  19000],
       [    35,  20000],
       [    26,  43000],
       [    27,  57000],
```

```
[    19,  76000],
[    27,  58000],
[    27,  84000],
[    32, 150000],
[    25,  33000],
[    35,  65000],
[    26,  80000],
[    26,  52000],
[    20,  86000],
[    32,  18000],
[    18,  82000],
[    29,  80000],
[    47,  25000],
[    45,  26000],
[    46,  28000],
[    48,  29000],
[    45,  22000],
[    47,  49000],
[    48,  41000],
[    45,  22000],
[    46,  23000],
[    47,  20000],
[    49,  28000],
[    47,  30000],
[    29,  43000],
[    31,  18000],
[    31,  74000],
[    27, 137000],
[    21,  16000],
[    28,  44000],
[    27,  90000],
[    35,  27000],
[    33,  28000],
[    30,  49000],
[    26,  72000],
[    27,  31000],
[    27,  17000],
[    33,  51000],
[    35, 108000],
[    30,  15000],
[    28,  84000],
[    23,  20000],
[    25,  79000],
[    27,  54000],
[    30, 135000],
[    31,  89000],
[    24,  32000],
```

```
[    42,  64000],
[    48,  33000],
[    44, 139000],
[    49,  28000],
[    57,  33000],
[    56,  60000],
[    49,  39000],
[    39,  71000],
[    47,  34000],
[    48,  35000],
[    48,  33000],
[    47,  23000],
[    45,  45000],
[    60,  42000],
[    39,  59000],
[    46,  41000],
[    51,  23000],
[    50,  20000],
[    36,  33000],
[    49,  36000]], dtype=int64)
```

[6]:
```
y
```

[6]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

[7]:
```
from sklearn.model_selection import train_test_split  # splitting the dataset␣
 ↪into training and testing dataset
xtrain, xtest, ytrain, ytest = train_test_split(x, y, test_size = 0.25,␣
 ↪random_state = 0)
```

[8]:
```
from sklearn.preprocessing import StandardScaler  # Training the data
sc_x = StandardScaler()
xtrain = sc_x.fit_transform(xtrain)
xtest = sc_x.transform(xtest)

print (xtrain[0:10, :])
```

```
[[ 0.58164944 -0.88670699]
 [-0.60673761  1.46173768]
 [-0.01254409 -0.5677824 ]
 [-0.60673761  1.89663484]
 [ 1.37390747 -1.40858358]
 [ 1.47293972  0.99784738]
 [ 0.08648817 -0.79972756]
 [-0.01254409 -0.24885782]
 [-0.21060859 -0.5677824 ]
 [-0.21060859 -0.19087153]]
```

[9]:
```
from sklearn.linear_model import LogisticRegression # Training the data in␣
 ↪logistic Classifier
classifier = LogisticRegression(random_state = 0)
classifier.fit(xtrain, ytrain)
```

[9]:
```
LogisticRegression(random_state=0)
```

[10]:
```
y_pred = classifier.predict(xtest)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(ytest, y_pred)

print ("Confusion Matrix : \n", cm)
```

```
Confusion Matrix :
 [[65  3]
 [ 8 24]]
```

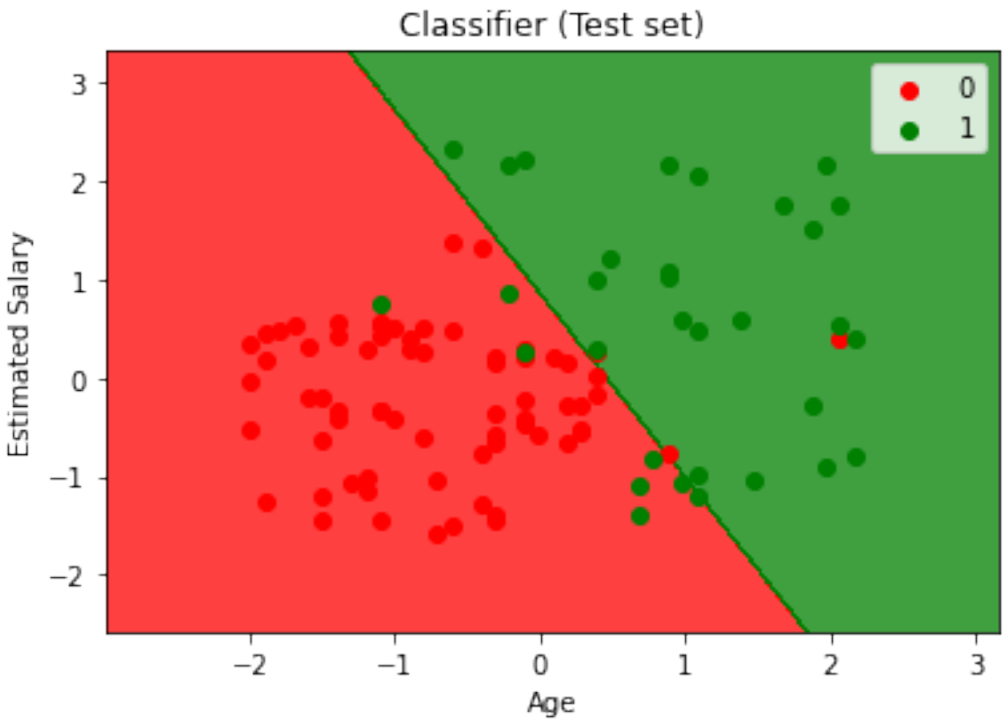
[11]:
```
from sklearn.metrics import accuracy_score  # Accuracy
print ("Accuracy : ", accuracy_score(ytest, y_pred))
```

```
Accuracy :  0.89
```

[13]:
```
from matplotlib.colors import ListedColormap
X_set, y_set = xtest, ytest
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(('red',␣
 ↪'green')))

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(('red', 'green'))(i), label␣
 ↪= j)

plt.title('Classifier (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*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.
```


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

[ ]:

[ ]:

naive-bayes

June 15, 2023

[1]:
```
from numpy import *
from pandas import *
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
```

[2]:
```
dataset = pd.read_csv('Iris.txt')
X = dataset.iloc[:, [1, 2, 3]].values
y = dataset.iloc[:, -1].values
dataset
```

[2]:
```
     5.1  3.5  1.4  0.2     Iris-setosa
0    4.9  3.0  1.4  0.2     Iris-setosa
1    4.7  3.2  1.3  0.2     Iris-setosa
2    4.6  3.1  1.5  0.2     Iris-setosa
3    5.0  3.6  1.4  0.2     Iris-setosa
4    5.4  3.9  1.7  0.4     Iris-setosa
..   …  …  …  …             …
144  6.7  3.0  5.2  2.3  Iris-virginica
145  6.3  2.5  5.0  1.9  Iris-virginica
146  6.5  3.0  5.2  2.0  Iris-virginica
147  6.2  3.4  5.4  2.3  Iris-virginica
148  5.9  3.0  5.1  1.8  Iris-virginica

[149 rows x 5 columns]
```

[3]:
```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
X[:,0] = le.fit_transform(X[:,0])
X
```

[3]:
```
array([[ 9. ,  1.4,  0.2],
       [11. ,  1.3,  0.2],
       [10. ,  1.5,  0.2],
       [15. ,  1.4,  0.2],
       [18. ,  1.7,  0.4],
```

```
       [ 9. ,  5.2,  2. ],
       [13. ,  5.4,  2.3],
       [ 9. ,  5.1,  1.8]])
```

[4]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20,␣
 ↪random_state = 0)
```

[5]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[6]:
```
from sklearn.naive_bayes import GaussianNB
classifier = GaussianNB()
classifier.fit(X_train, y_train)
```

[6]:
```
GaussianNB()
```

[7]:
```
y_pred  =  classifier.predict(X_test)
```

[8]:
```
y_pred
```

[8]:
```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-setosa', 'Iris-setosa', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica'], dtype='<U15')
```

[9]:
```
y_test
```

[9]:
```
array(['Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-setosa', 'Iris-setosa', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-setosa', 'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica'], dtype=object)
```

[10]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
cm = confusion_matrix(y_test, y_pred)
```

```
ac = accuracy_score(y_test,y_pred)
```

[11]:
```
cm
```

[11]:
```
array([[12,  0,  0],
       [ 0, 10,  0],
       [ 0,  3,  5]], dtype=int64)
```

[12]:
```
ac
```

[12]:
```
0.9
```

[ ]:
```

```

decision-tree

June 15, 2023

[1]:
```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

[2]:
```
data_set= pd.read_csv('user_data.csv')
data_set
```

[2]:

	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 x 5 columns]

[3]:
```
x= data_set.iloc[:, [2,3]].values
y= data_set.iloc[:, 4].values
x
y
```

[3]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 0, 0, 0, 1,
```

```
       0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 0, 1, 1, 0,
       1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 1, 1, 0,
       1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1,
       1, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1,
       0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0,
       1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 0, 1,
       1, 1, 0, 1], dtype=int64)
```

[4]:
```
# Splitting the dataset into training and test set.
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)
```

[7]:
```
#feature Scaling
from sklearn.preprocessing import StandardScaler
st_x= StandardScaler()
x_train= st_x.fit_transform(x_train)
x_test= st_x.transform(x_test)
```

[9]:
```
#Fitting Decision Tree classifier to the training set
from sklearn.tree import DecisionTreeClassifier
classifier= DecisionTreeClassifier(criterion='entropy', random_state=0)
classifier.fit(x_train, y_train)
```

[9]:
```
DecisionTreeClassifier(criterion='entropy', random_state=0)
```

[10]:
```
#Predicting the test set result
y_pred= classifier.predict(x_test)
```

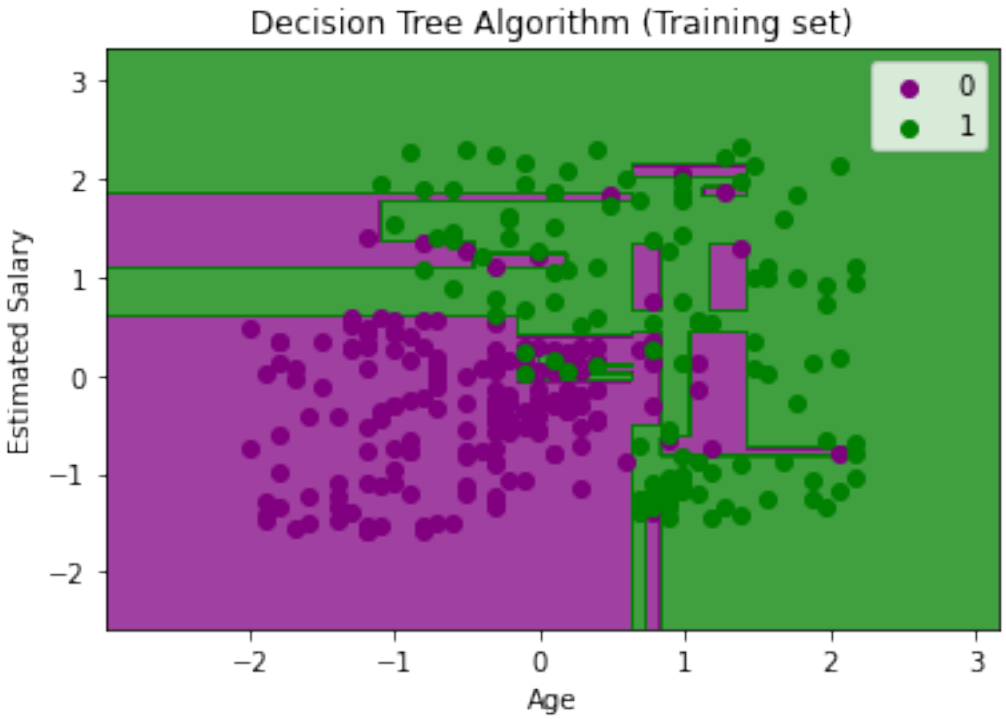
[12]:
```
#Creating the Confusion matrix
from sklearn.metrics import confusion_matrix
cm= confusion_matrix(y_test, y_pred)
cm
```

[12]:
```
array([[62,  6],
       [ 3, 29]], dtype=int64)
```

[15]:
```
#Visulaizing the trianing set result
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(('purple','green' )))
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(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm (Training set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*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.
```



[16]:
```
#Visulaizing the test set result
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(('purple','green' )))
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(('purple', 'green'))(i), label = j)
mtp.title('Decision Tree Algorithm(Test set)')
mtp.xlabel('Age')
mtp.ylabel('Estimated Salary')
mtp.legend()
mtp.show()
```

```
*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.
```


Decision Tree Algorithm(Test set)
Estimated Salary
Age
0
1
3
2
1
0
−1
−2
−2
−1
0
1
2
3

[ ]:

k-means-clustering

June 15, 2023

[7]:
```
# importing libraries
import numpy as np
import matplotlib.pyplot as mtp
import pandas as pd
```

[17]:
```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.txt')
x = dataset.iloc[:, [3,4]].values
x
```

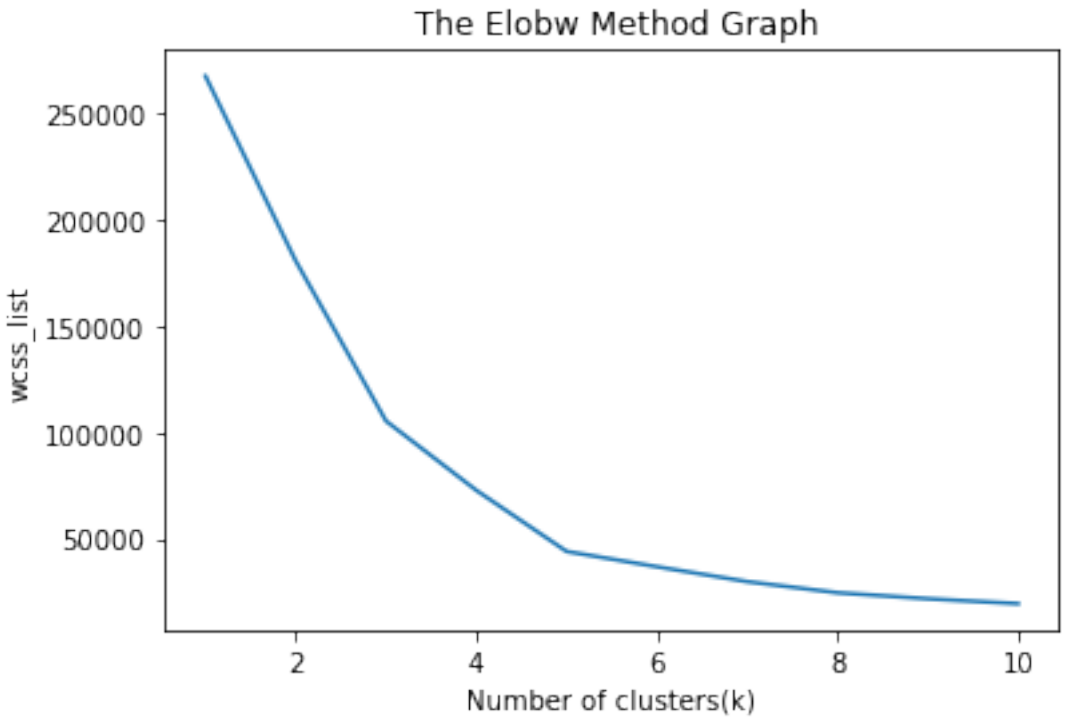
[17]:
```
array([[ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
       [ 28,  61],
```

```
       [ 87,  63],
       [ 87,  13],
       [ 87,  75],
       [ 87,  10],
       [ 87,  92],
       [ 88,  13],
       [ 88,  86],
       [ 88,  15],
       [ 88,  69],
       [ 93,  14],
       [ 93,  90],
       [ 97,  32],
       [ 97,  86],
       [ 98,  15],
       [ 98,  88],
       [ 99,  39],
       [ 99,  97],
       [101,  24],
       [101,  68],
       [103,  17],
       [103,  85],
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]], dtype=int64)
```

[25]:
```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list = [] #Initializing the list for the values of WCSS
```

[34]:
```
#Using for loop for iterations from 1 to 10.
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_)
mtp.plot(range(1,11), wcss_list)
mtp.title(' The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

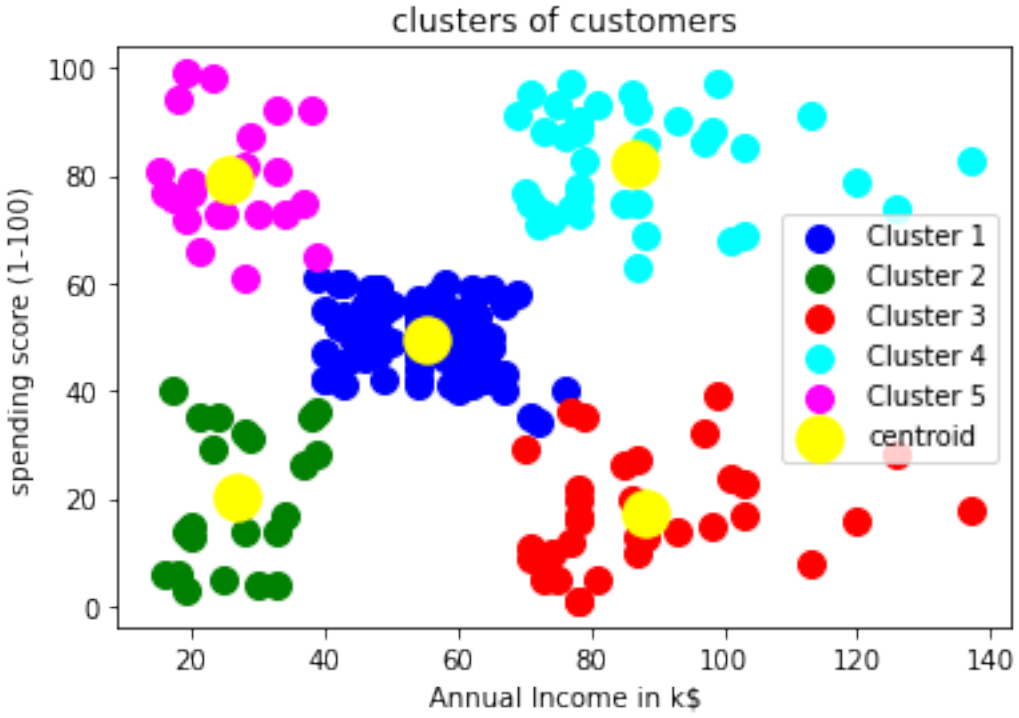
```
C:\Users\HP\anaconda3\lib\site-packages\sklearn\cluster\_kmeans.py:1036:
UserWarning: KMeans is known to have a memory leak on Windows with MKL, when
there are less chunks than available threads. You can avoid it by setting the
environment variable OMP_NUM_THREADS=1.
  warnings.warn(
```



[39]:
```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state=42)
y_predict = kmeans.fit_predict(x)
y_predict
```

[39]:
```
array([4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1,
       4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 4, 1, 0, 1,
       4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, 2, 3, 0, 3, 2, 3, 2, 3, 0,
       3, 2, 3, 2, 3, 2, 3, 2, 3, 0, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2,
       3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2,
       3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2, 3, 2,
       3])
```

[55]:
```
mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue',␣
 ↪label = 'Cluster 1') #for first cluster
mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green',␣
 ↪label = 'Cluster 2') #for second cluster
mtp.scatter(x[y_predict == 2, 0], x[y_predict == 2, 1], s = 100, c = 'red',␣
 ↪label = 'Cluster 3') #for third cluster
mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan',␣
 ↪label = 'Cluster 4') #for fourth cluster
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta',␣
 ↪label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s=␣
 ↪300, c= 'yellow', label = 'centroid')
mtp.title('clusters of customers')
mtp.xlabel('Annual Income in k$')
mtp.ylabel('spending score (1-100)')
mtp.legend()
mtp.show()
#mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:
 ↪, 1], s = 300, c = 'yellow', label = 'Centroid')
```



[ ]: