UTS KECERDASAN BUATAN

Nama : Masagus Muhammad Fajry Mantofani

NIM : 3332200074

MK : Kecerdasan Buatan A

Link Github :

Link Youtube :

1. Analisa algoritma untuk *logistic_regression.py*. Dan analisa algoritmanya dan jalankan di komputer anda. (Untuk Chapter 2)

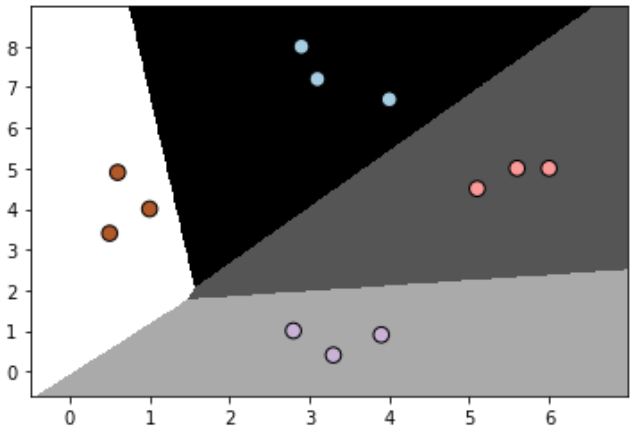
   Jawab :

   Berikut ini adalah code untuk chapter 2 mengenai *logistic_regression* :

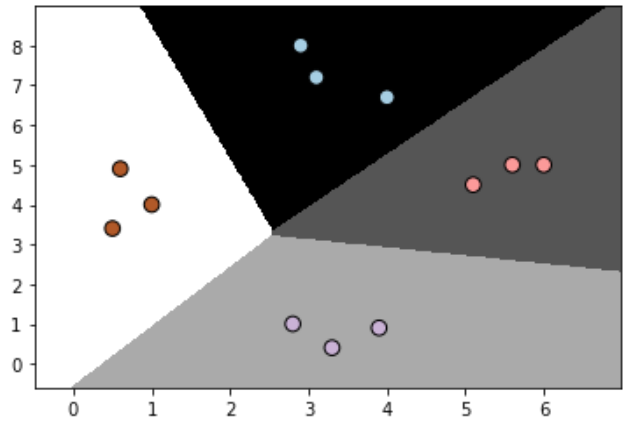
```
import numpy as np
import matplotlib.pyplot as plt
def visualize_classifier(classifier, X, y):
    # Define the minimum and maximum values for X and Y
    # that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01

    # Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh
_step_size), np.arange(min_y, max_y, mesh_step_size))

    # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.
ravel()])

    # Reshape the output array
    output = output.reshape(x_vals.shape)

    # Create a plot
    plt.figure()

    # Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)

    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='blac
k', linewidth=1, cmap=plt.cm.Paired)
```

```
    # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0]
.max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1]
.max() + 1), 1.0)))

    plt.show()

import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt

# Define sample input data
X = np.array([[3.1, 7.2], [4, 6.7], [2.9, 8], [5.1, 4.5], [6,
 5], [5.6, 5], [3.3, 0.4], [3.9, 0.9], [2.8, 1], [0.5, 3.4],
[1, 4], [0.6, 4.9]])
y = np.array([0, 0, 0, 1, 1, 1, 2, 2, 2, 3, 3, 3])

# Create the logistic regression classifier
classifier = linear_model.LogisticRegression(solver='liblinea
r', C=1)
#classifier = linear_model.LogisticRegression(solver='libline
ar', C=100)

# Train the classifier
classifier.fit(X, y)

# Visualize the performance of the classifier
visualize_classifier(classifier, X, y)
```

Hasil pada nilai C = 1 :



Hasil pada nilai C = 100 :



Logistic regression biasa difungsikan untuk melakukan pengelompokan data menjadi beberapa bagian dengan melakukan plot pada sebuah data acak. Pada figure yang dihasilkan dari. *Logistic Regression* sendiri merupakan algoritma klasifikasi untuk mencari hubungan antara fitur (input) diskrit/kontinu dengan probabilitas hasil output diskrit tertentu. Pada figure yang dihasilkan dari program diatas dapat dilihat terdapat perbedaan warna pada setiap datanya dimana perbedaan tersebut pula terdapat pada tiap bagian kelompoknya. Perbedaan warna pada data tersebut seperti pada warna oren, biru, merah muda dan ungu dihasilkan dari induk yang terdapat didalam pengelompokan data dimana tiap data memiliki induk yang memberikan warna untuk membedakan pengelompokan data. Yang berarti data yang mendekati induk tersebut akan diberikan warna untuk dikelompokkan. Pada setiap wilayahnya ditandakan dengan warna yang berbeda seperti putih, hitam, abu tua dan abu muda untuk menandai bahwa 3 data tersebut masuk pada pengelompokkan dengan bagian yang sesuai dalam arti kata pada 3 data berwarna oren tidak masuk pada bagian data yang berwarna biru.

Pada kedua *figure* yang dihasilkan diatas dilakukan dua kali percobaan dengan nilai C (kurva) = 1 dan C = 100. Pada C = 1 dihasilkan titik temu yang kurang baik dimana titik temu tidak halus dan sentral hal itu bisa dilihat pada bagian abu – abu tua masuk ke wilayah lain yang mana bukan kelompoknya. Sedangkan pada kurva C = 100 titik temu yang dihasilkan jauh lebih baik dan cenderung sentral serta titik temu tersebut jauh lebih halus atau *smooth*

dibandingkan dengan titik C = 1, hal tersebut terbukti pada wilayah kelompok warna abu – abu tua dimana wilayah tersebut tidak memakan wilayah lain, walaupun masih sedikit masuk pada wilayah lain hal itu wajar dan dapat dimaklumi karena keakuratan tidak mungkin sampai 100%. Maka dapat disimpulkan bahwa *logistic regression* ini dipengaruhi oleh nilai kurva (C) dimana semakin besar nilai kurva maka hasil titik temunya akan semakin baik.

2. Analisa algoritma untuk *decision_trees.py*. Dan analisa algoritmanya dan jalankan di komputer anda. (Untuk Chapter 3)

   Jawab :

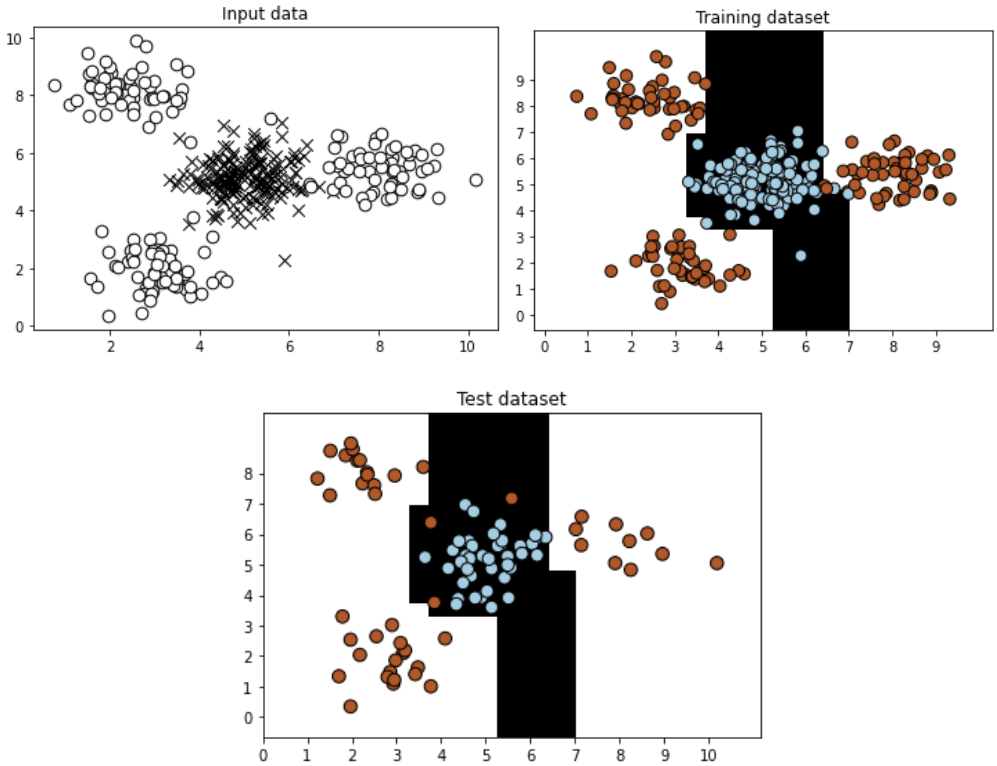
   Berikut ini adalah code yang digunakan pada Chapter 3 mengenai *decision_trees.py*. :

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

def visualize_classifier(classifier, X, y, title=''):
    # Define the minimum and maximum values for X and Y
    # that will be used in the mesh grid
    min_x, max_x = X[:, 0].min() - 1.0, X[:, 0].max() + 1.0
    min_y, max_y = X[:, 1].min() - 1.0, X[:, 1].max() + 1.0

    # Define the step size to use in plotting the mesh grid
    mesh_step_size = 0.01

    # Define the mesh grid of X and Y values
    x_vals, y_vals = np.meshgrid(np.arange(min_x, max_x, mesh
_step_size), np.arange(min_y, max_y, mesh_step_size))

    # Run the classifier on the mesh grid
    output = classifier.predict(np.c_[x_vals.ravel(), y_vals.
ravel()])

    # Reshape the output array
    output = output.reshape(x_vals.shape)

    # Create a plot
    plt.figure()

    # Specify the title
    plt.title(title)

    # Choose a color scheme for the plot
    plt.pcolormesh(x_vals, y_vals, output, cmap=plt.cm.gray)
```

```
    # Overlay the training points on the plot
    plt.scatter(X[:, 0], X[:, 1], c=y, s=75, edgecolors='blac
k', linewidth=1, cmap=plt.cm.Paired)

    # Specify the boundaries of the plot
    plt.xlim(x_vals.min(), x_vals.max())
    plt.ylim(y_vals.min(), y_vals.max())

    # Specify the ticks on the X and Y axes
    plt.xticks((np.arange(int(X[:, 0].min() - 1), int(X[:, 0]
.max() + 1), 1.0)))
    plt.yticks((np.arange(int(X[:, 1].min() - 1), int(X[:, 1]
.max() + 1), 1.0)))

    plt.show()

import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
#from sklearn import cross_validation
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split

# Load input data
input_file = 'data_decision_trees.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]

# Separate input data into two classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
# Visualize input data
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='b
lack',
        edgecolors='black', linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='w
hite',
        edgecolors='black', linewidth=1, marker='o')
plt.title('Input data')

# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)

# Decision Trees classifier
params = {'random_state': 0, 'max_depth': 4}
classifier = DecisionTreeClassifier(**params)
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training
dataset')
```

```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test datase
t')

# Evaluate classifier performance
class_names = ['Class-0', 'Class-1']
print("\n" + "#"*40)
print("\nClassifier performance on training dataset\n")
print(classification_report(y_train, classifier.predict(X_tra
in), target_names=class_names))
print("#"*40 + "\n")

print("#"*40)
print("\nClassifier performance on test dataset\n")
print(classification_report(y_test, y_test_pred, target_names
=class_names))
print("#"*40 + "\n")

plt.show()
```

Hasil yang diperoleh :




```
########################################
Classifier performance on training dataset
```

```
             precision    recall  f1-score   support
Class-0         0.99      1.00      1.00       137
Class-1         1.00      0.99      1.00       133
accuracy                            1.00       270
macro avg       1.00      1.00      1.00       270
weighted avg    1.00      1.00      1.00       270
########################################
Classifier performance on test dataset
            precision    recall  f1-score   support
Class-0         0.93      1.00      0.97        43
Class-1         1.00      0.94      0.97        47
accuracy                            0.97        90
macro avg       0.97      0.97      0.97        90
weighted avg    0.97      0.97      0.97        90
########################################
```

Pada praktek kali ini mengenai *decision tree* atau bisa juga disebut sebagai pohon keputusan yang mana metode ini baik digunakan untuk melakukan pengelompokkan data dengan cabang – cabang yang terorganisir. Pada umumnya metode ini difungsikan untuk melakukan pengelompokkan data yang mana dari data tersebut memiliki nilai atau kondisi *true* atau *false*. Kedua kondisi tersebut kemudian dibagi dua kembali menjadi *true positif*, *true negative*, *false positif* dan *false negative*.

Pada percobaan diatas dihasilkan *figure* yang memiliki garis atau wilayah hitam yang mana bagian tersebut merupakan zona atau bagian pemisah dari suatu kondisi. Warna yang ada pada tiap data pula menginisiasikan bahwa data tersebut masuk dalam suatu kelompok data tertentu. Dimisalkan kembali bahwa terdapat dua data dimana data A merupakan bagian hitam yang berisi *true* dan *false* pada setiap datanya. Dan data B merupakan bagian putih yang berisi data *true* dan *false*. Pada data A yang memiliki dua kondisi dapat disebut *true positif* jika kondisi yang dialami 100% benar yang berarti data terdapat pada wilayah A dan saat kondisi *true negative* yaitu kondisi saat data A yang seharusnya berada pada wilayah A namun masuk pada wilayah B. Hal itu karena nilai yang benar namun sistem membaca bahwa nilai tersebut adalah salah (*False*) bukan dari bagian A sehingga masuk pada bagian B. Pada data wilayah B pula akan

sama, hal itu terlihat pada data arna oren yang 100% benar berada pada wilayah B atau putih maka dapat diartikan data tersebut *true positif*, namun terdapat beberapa data B terbaca oleh sistem bahwa bernilai *false* sehingga data B tersebut masuk pada wilayah A yang diartikan data B bernilai *true negative*.

Decision tree memiliki parameter *precision, recall, f1-score*, dan *support*. *Precision* sendiri merupakan parameter ketepatan data menempati wilayah atau plot yang seharusnya, *recall* adalah banyaknya data yang dipanggil, *f1-score* merupakan hasil rata – rata yang dihasilkan dari *precision* dan *recall* dimana keduanya dijumlahkan kemudian dibagi 2, dan yang terakhir yaitu *support* yang merupakan bagian untuk menampilkan banyaknya jumlah data yang digunakan pada program tersebut. *F1-score* pada percobaan dikategorikan baik atau sempurna bila bernilai 1, namun bisa dikatakan sempurna pula walau dihasilkan nilai *f1-score* 0,97 karena sejatinya tidak memungkinkan percobaan untuk mendapatkan ketepatan 100%

3. Analisa algoritma untuk *mean_shift.py*. Dan analisa algoritmanya dan jalankan di komputer anda. (untuk Chapter 4)

   Jawab :

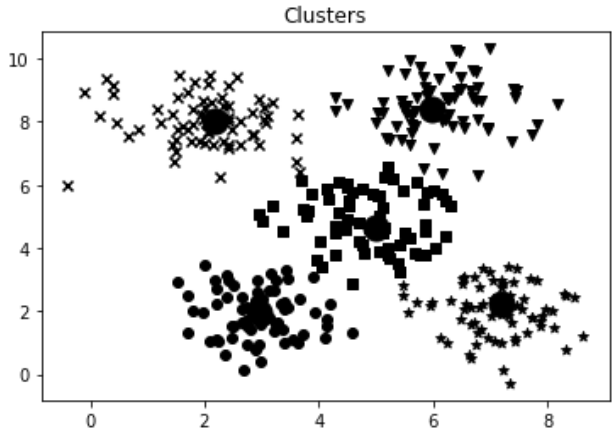
   Berikut ini kode untuk Chapter 4 mengenai mean shift :

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle

# Load data from input file
X = np.loadtxt('data_clustering.txt', delimiter=',')

# Estimate the bandwidth of X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=l
en(X))

# Cluster data with MeanShift
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seedin
g=True)
meanshift_model.fit(X)

# Extract the centers of clusters
cluster_centers = meanshift_model.cluster_centers_
print('\nCenters of clusters:\n', cluster_centers)
```

```
# Estimate the number of clusters
labels = meanshift_model.labels_
num_clusters = len(np.unique(labels))
print("\nNumber of clusters in input data =", num_clusters)

# Plot the points and cluster centers
plt.figure()
markers = 'o*xvs'
for i, marker in zip(range(num_clusters), markers):
    # Plot points that belong to the current cluster
    plt.scatter(X[labels==i, 0], X[labels==i, 1], marker=mark
er, color='black')

    # Plot the cluster center
    cluster_center = cluster_centers[i]
    plt.plot(cluster_center[0], cluster_center[1], marker'o',

             markerfacecolor='black', markeredgecolor='black',
             markersize=15)

plt.title('Clusters')
plt.show()
```

Hasil yang diperoleh :

```
Centers of clusters:
 [[2.95568966 1.95775862]
 [7.20690909 2.20836364]
 [2.17603774 8.03283019]
 [5.97960784 8.39078431]
 [4.99466667 4.65844444]]
Number of clusters in input data = 5
```



Pada percobaan ketiga ini mengenai *Data Clustering* yang berarti mengelompokkan data berdasarkan kluster atau kelompok tertentu. Pada *mean*

shift ini data – data yang mirip atau cenderung sama akan dikelompokkan dan mendekati data yang berperan sebagai kunci supaya data – data yang mirip dan sesuai dengan data kunci yang sudah ditentukan dapat dikelompokkan. Pada percobaan yang ada terdapat 5 buah data yang mana data tersebut masing – masing memiliki kelompok yang berbeda dimana tiap kelompok tersebut terdapat kata kunci atau data yang berperan sebagai kategori dari setiap kelompok tersebut.

Pada hasil yang didapat terdapat beberapa titik besar atau lingkaran besar yang merupakan data kunci atau induk data yang membuat data dapat dikelompokkan dan masuk pada kategori kelompok tertentu. Yang mana lingkaran besar merupakan pusat data agar data yang memiliki kemiripan akan mendekati atau masuk pada kelompok tersebut. Dimisalkan dalam kehidupan jika kita melakukan pencarian pada sebuah aplikasi *start-up* atau *e-commerce*, yaitu ketika mencari sebuah produk makanan pada aplikasi Grab yaitu dengan memasukkan kata kunci berupa "ayam" maka akan tampil makanan – makanan yang termasuk olahan ayam, kata kunci ayam tersebut merupakan pusat atau induk data yang membuat kategori kelompok tertentu, dan olahan – olahan ayam tersebut merupakan data yang memiliki kemiripan atau sesuai dengan kata kunci "ayam" yang telah diinputkan. Kemudian pada sebuah data kelompok terdapat data yang jauh dari pusatnya yang mana data tersebut sebenarnya tidak sesuai dengan induk atau data pusat tetapi data yang jauh tersebut memiliki kemiripan dengan data induknya. Misalkan kita mencari handphone pada sebuah *e – commerce* namun akan muncul casing handphone, maka casing handphone tersebut lah yang disebut dengan data yang memiliki sedikit kemiripan atau hubungan pada data induknya.

4. Analisa algoritma untuk *nearest_neighbors_classifier.py*. Dan analisa algoritmanya dan jalankan di komputer anda (untuk Chapter 5)

   Jawab :

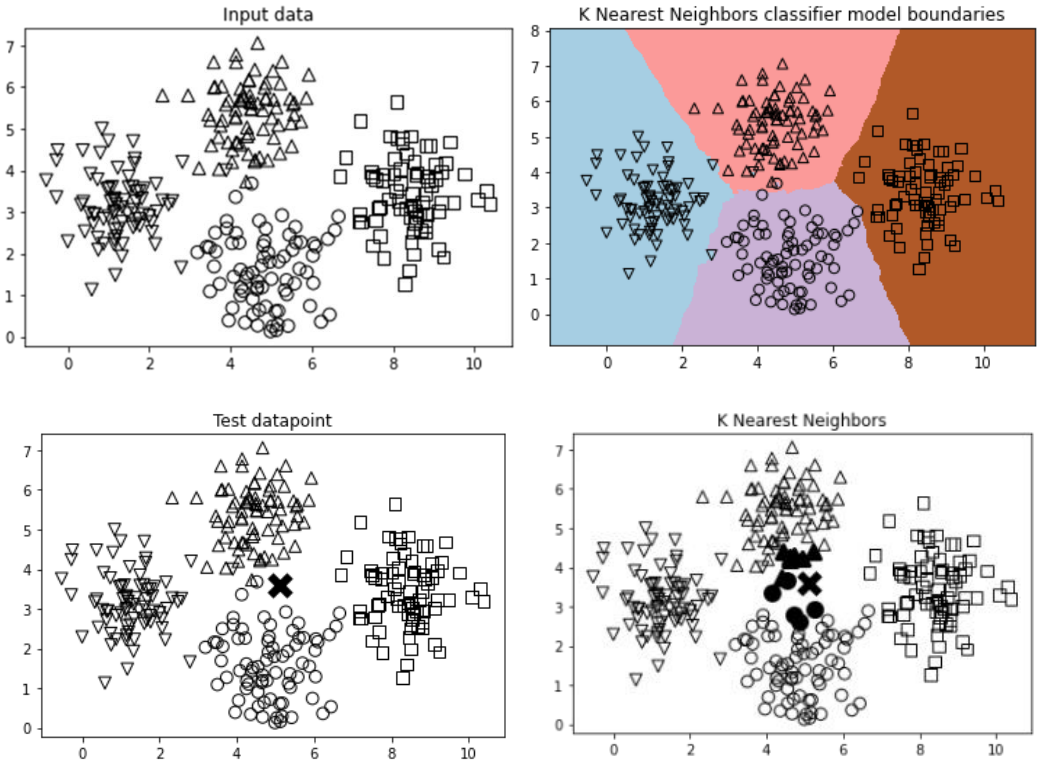
   Berikut ini kode program Chapter 5 mengenai *nearest_neighbors_classifier* :

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

```
from sklearn import neighbors, datasets

# Load input data
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(np.int)

# Plot input data
plt.figure()
plt.title('Input data')
marker_shapes = 'v^os'
mapper = [marker_shapes[i] for i in y]
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')

# Number of nearest neighbors
num_neighbors = 12

# Step size of the visualization grid
step_size = 0.01

# Create a K Nearest Neighbours classifier model
classifier = neighbors.KNeighborsClassifier(num_neighbors, we
ights='distance')

# Train the K Nearest Neighbours model
classifier.fit(X, y)

# Create the mesh to plot the boundaries
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
x_values, y_values = np.meshgrid(np.arange(x_min, x_max, step
_size),
        np.arange(y_min, y_max, step_size))

# Evaluate the classifier on all the points on the grid
output = classifier.predict(np.c_[x_values.ravel(), y_values.
ravel()])

# Visualize the predicted output
output = output.reshape(x_values.shape)
plt.figure()
plt.pcolormesh(x_values, y_values, output, cmap=cm.Paired)

# Overlay the training points on the map
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=50, edgecolors='black', facecolors='none')

plt.xlim(x_values.min(), x_values.max())
```

```
plt.ylim(y_values.min(), y_values.max())
plt.title('K Nearest Neighbors classifier model boundaries')

# Test input datapoint
test_datapoint = [5.1, 3.6]
plt.figure()
plt.title('Test datapoint')
for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',

        linewidth=6, s=200, facecolors='black')

# Extract the K nearest neighbors
_, indices = classifier.kneighbors([test_datapoint])
indices = indices.astype(np.int)[0]

# Plot k nearest neighbors
plt.figure()
plt.title('K Nearest Neighbors')

for i in indices:
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[y[i]],
            linewidth=3, s=100, facecolors='black')

plt.scatter(test_datapoint[0], test_datapoint[1], marker='x',

        linewidth=6, s=200, facecolors='black')

for i in range(X.shape[0]):
    plt.scatter(X[i, 0], X[i, 1], marker=mapper[i],
            s=75, edgecolors='black', facecolors='none')

print("Predicted output:", classifier.predict([test_datapoint
])[0])

plt.show()
```

Hasil yang diperoleh :



Pada program kali ini yaitu mengenai *K-Nearest Neighbor* (KNN), *K-Nearest Neighbor* bekerja dengan mencari jarak terdekat dari data yang ada untuk dievaluasi dengan K-tetangga terdekatnya. KNN ini mengkasifikasikan sebuah data berdasarkan mayoritas dari kategori KNN tersebut, yang mana *K-Nearest Neighbor* menggunakan algoritma *supervised*. KNN ini membuat kelompok data berdasarkan tetangga terdekat dimana klasifikasinya berdasarkan mayoritas yang ada. Dari percobaan yang ada tujuan dari KNN ini yaitu mendata atau mengklasifikasikan sebuah objek yang baru berdasarkan *training sample* yang ada. Pengklasifikasian ini menggunakan memori dari data yang sebelumya sebagai klasifikasinya.

Pada hasil percobaan jika dilihat pada *figure* yeng memiliki warna dimana warna tersebut sebaga bagian – bagian kelompok yang berbeda. Pada umumnya KNN ini memiliki kemiripan pada program *data clustering*, hanya saja pada KNN ini berbeda karena tidak memiliki induk atau data pusat sebagai acuan untuk data lainnya. Pada KNN ini data yang ada dikelompokkan secara otomatis dan random. Dimisalkan pada kehidupan sehari – hari dimana kita

menggunakan sebuah gadget, gadget tersebut merekam kegiatan kita selama sehari – hari, dari rekaman kegiatan tersebut dapat menjadi acuan medsos akan menampilkan data random yang berkaitan dengan apa yang sebelumnya kita lihat pada medsos tersebut. Data random tersebut dikelompokkan secara otomatis oleh mesin yang bekerja pada sebuah medsos. Dimisalkan kembali ketika masuk pada halaman beranda sebuah *e – commerce* maka akan ditampilkan beberapa barang – barang rekomendasi sesuai dari apa yang kita cari atau sering lihat sebelumnya pada *e – commerce* tersebut. Hal itu karena KNN ini bekerja dengan mengklasifikasikan secara memori. Maka dapat disimpulkan jika *K-Nearest Neighbor* ini proses pengelompokan ditentukan dengan mencari kemiripan satu sama lain antar tetangga secara random dan otomatis tetapi tetap tergantung pada *rule base* yang ada. *K-Nearest Neighbor* ini juga menggunakan teknik klasifikasi berdasarkan ketetanggaan untuk memprediksi nilai dari query instance yang baru.

5. Analisa algoritma untuk states.py. Dan analisa algoritmanya dan jalankan di komputer anda (untuk Chapter 6)

   Jawab :

   Beriku ini kode untuk chapter 6 mengenai *States* :

```
from logpy import run, fact, eq, Relation, var

adjacent = Relation()
coastal = Relation()

file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'

# Read the file containing the coastal states
with open(file_coastal, 'r') as f:
    line = f.read()
    coastal_states = line.split(',')

# Add the info to the fact base
for state in coastal_states:
    fact(coastal, state)

# Read the file containing the coastal states
with open(file_adjacent, 'r') as f:
    adjlist = [line.strip().split(',') for line in f if line
and line[0].isalpha()]
```

```
# Add the info to the fact base
for L in adjlist:
    head, tail = L[0], L[1:]
    for state in tail:
        fact(adjacent, head, state)

# Initialize the variables
x = var()
y = var()

# Is Nevada adjacent to Louisiana?
output = run(0, x, adjacent('Nevada', 'Louisiana'))
print('\nIs Nevada adjacent to Louisiana?:')
print('Yes' if len(output) else 'No')

# States adjacent to Oregon
output = run(0, x, adjacent('Oregon', x))
print('\nList of states adjacent to Oregon:')
for item in output:
    print(item)

# States adjacent to Mississippi that are coastal
output = run(0, x, adjacent('Mississippi', x), coastal(x))
print('\nList of coastal states adjacent to Mississippi:')
for item in output:
    print(item)

# List of 'n' states that border a coastal state
n = 7
output = run(n, x, coastal(y), adjacent(x, y))
print('\nList of ' + str(n) + ' states that border a coastal
state:')
for item in output:
    print(item)

# List of states that adjacent to the two given states
output = run(0, x, adjacent('Arkansas', x), adjacent('Kentuck
y', x))
print('\nList of states that are adjacent to Arkansas and Ken
tucky:')
for item in output:
    print(item)
```

Hasil yang diperoleh :

```
Is Nevada adjacent to Louisiana?:
No


List of states adjacent to Oregon:
```

```
Nevada
Washington
California
Idaho

List of coastal states adjacent to Mississippi:
Alabama
Louisiana

List of 7 states that border a coastal state:
Pennsylvania
Ohio
New Hampshire
Georgia
Tennessee
Delaware
New Jersey

List of states that are adjacent to Arkansas and Kentucky:
Tennessee
Missouri
```

Pada program *states* yang mana metode algoritma ini merujuk pada suatu tempat yang berfungsi untuk menyimpan suatu keadaan tertentu pada sebuah program. *State* ini bisa diartikan sebagai keadaan, suasana atau situasi yang sedang dihadapkan pada sebuah program. Program pada chapter 6 ini menggunakan hubungan relasi atau kedekatan antar data pada program yang ada. Pada program ini diinput library dengan data *coastal states* dan *adjacent states.*

Jika dilihat pada hasil yang pertama jika diartikan apakah Nevada berdekatan dengan Louisiana maka pada program akan menjawab tidak karena pada keadaannya dilihat dari *adjacent_states.txt* dimana kota Nevada tidak berdekatan dengan kota Lousiana. Selanjutnya yaitu daftar kota yang berdekatan dengan Oregon, maka program akan membaca kota Nevada, California, Washington dan Idaho. Kota – kota tersebut dapat dilihat pada file data pada library *adjacent_states.txt* dimana program atau sistem akan

mengambil data yang terdapat kata atau Kota Oregon yang paling depan sehingga didapati kota – kota tersebutlah yang berdekatan dengan Oregon. Kemudian selanjutnya pada list kota yang merupakan *coastel state* yang berdekatan pada Missisipi, maka data diambil pada *adjacent_states.txt* terlebih dahulu untuk mencari kota yang berdekatan dengan Missisipi, kemudian kota yang berdekatan tersebut dilakukan pengecekan apakah termasuk kedalam *list of coastel_state.txt* dari situ dapat ditemukan bahwa kota yang berdekatan dengan Missisipi dan termasuk kedalam *Coastel_state* adalah Alabama dan Lousiana. Selanjutnya list 7 negara yang membatasi atau mengelilingi *coastel state*, maka berdasarkan data *library* maka negara – negara yang mengelilingi *coastel state* yaitu Georgia, Massachusetts, Arizona, Nevada, Vermont, Pennyslvania dan Idaho. Kemudian *list state* yang juga berdekatan dengan kota Arkansas dan Kentucky. Maka program akan membaca data library adjacent_states.txt untuk melihat kota mana yang berdekatan pada Arkansas terlebih dahulu, data dilihat dari awalan kota Arkansas pada list data, kemudian dilihat pula kota yang bersebelahan dengan Kentucky, kemudian dari kedua kota tersebut dicari kota yang sama – sama berdekatan dengan Arkansas dan Kentucky, maka dapat dikatakan yaitu irisan (*intersection*) dari kota yang berdekatan dengan Arkansas dan Kentucky, sehingga didapat kotanya adalah Tennessee dan Missouri. Maka dapat disimpulkan bahwa *state* ini adalah sebuah metode dengan logika algoritma dimana data yang ada akan memiliki relasi atau hubungan dengan program yang ada dimana program akan menjalankan suatu kondisi berdasarkan data *library* yang ada.