UTS_Kecerdasan_Buatan_B_Hernando_Dio_Palma_Analisa

April 8, 2022

1 Ujian Tengah Semester Kecerdasan Buatan

- Nama : Hernando Dio Palma
- NIM : 3332190049
- Kelas : B

Download data-data yang diperlukan.

[ ]:
```
!wget --no-check-certificate 'https://docs.google.com/uc?
↪export=download&id=1zlJwPKAsKhlGlNu6Cn7XhKt_IUFH87YF' -O data_decision_trees.
↪txt
!wget --no-check-certificate 'https://docs.google.com/uc?
↪export=download&id=1oNqpwqQarcuIavyZFHYRAtxBnShDOPIo' -O data_clustering.txt
!wget --no-check-certificate 'https://docs.google.com/uc?
↪export=download&id=13FHXPHZK3yRHhr0OMqz6YTM70HGxEeMa' -O data.txt
!wget --no-check-certificate 'https://docs.google.com/uc?
↪export=download&id=13iqaBwvhTtyD7IX-5v5pSnAJQOouxGFT' -O coastal_states.txt
!wget --no-check-certificate 'https://docs.google.com/uc?
↪export=download&id=1AnbKOMXh_3GB2Y9jTGkmF4bDXfpbgRAr' -O adjacent_states.txt
```

1.0.1 Classifier Visualizer

[2]:
```
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='black', 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()
```

1.1 Logistic Regression

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

- Algoritma logistic regression merupakan algoritma yang digunakan untuk melakukan proses klasifikasi antara variabel independent dan variabel dependent. Algoritma ini memanfaatkan fungsi logistik yang menghasilkan kurva sigmoid, dengan menggunakan algoritma ini dapat dianalisa hubungan yang dimiliki antara variabel independent dan dependent dengan memperkirakan probabilitas yang dimilikinya.

1. Dilakukan pemanggilan library, digunakan library sklearn untuk memanggil fungsi logistic regression.

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

2. Didefinisikan variabel dependent dan independent kedalam bentuk array numpy.Variabel independent didefinisikan sebagai X sedangkan variabel independentnya didefinisikan sebagai y.

[4]:
```
# 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])
```

3. Pembuatan model untuk melatih data dapat dilakukan dengan memanggil fungsi LogisticRegression()

[5]:
```
# Create the logistic regression classifier
classifier1 = linear_model.LogisticRegression(solver='liblinear', C=1)
classifier2 = linear_model.LogisticRegression(solver='liblinear', C=100)
```

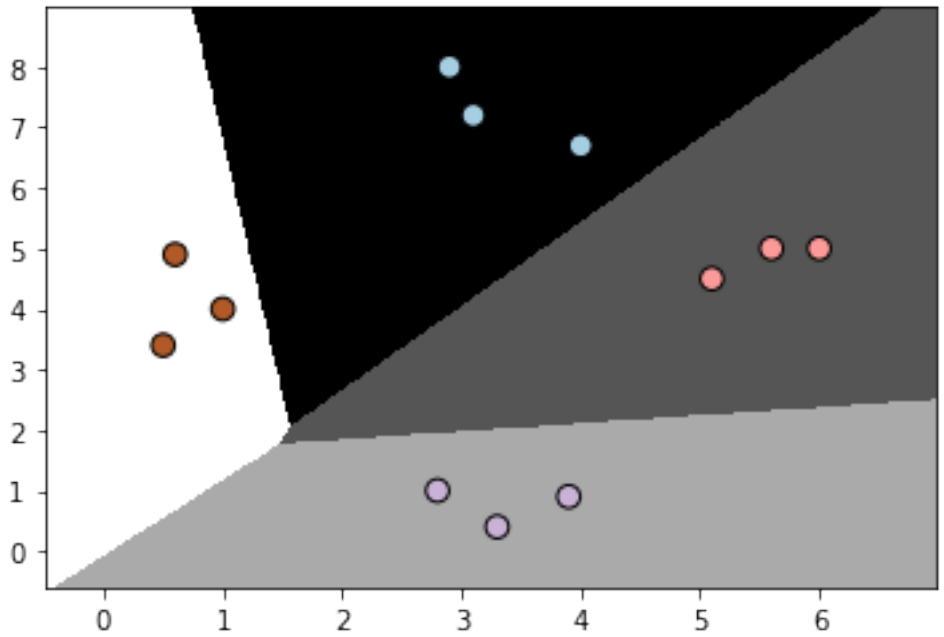
4. Melatih data dengan menggunakan fungsi fit() menggunakan data yang sudah didefinisikan sebelumnya.

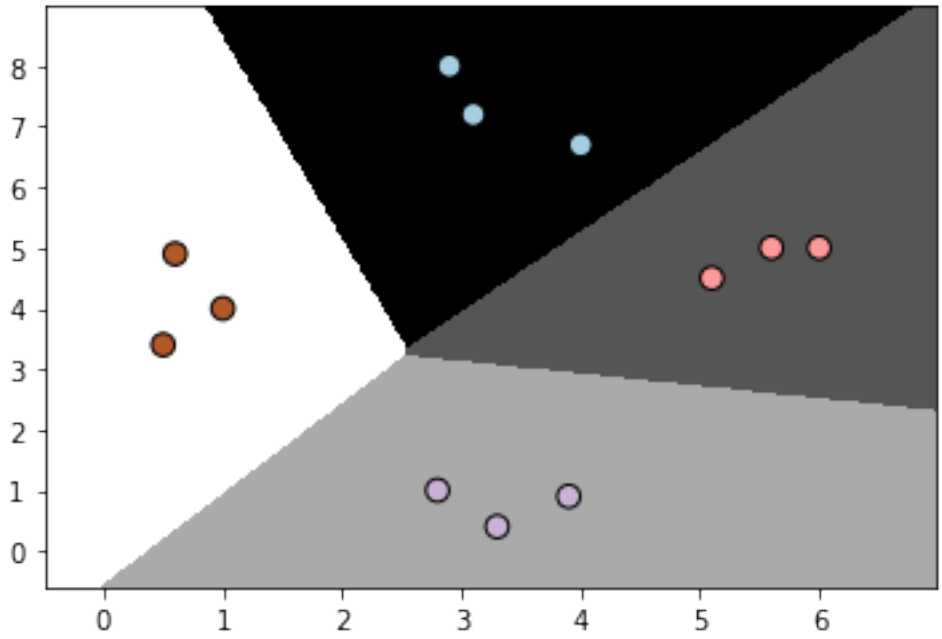
[6]:
```
# Train the classifier
classifier1.fit(X, y)
classifier2.fit(X, y)
```

[6]:
```
LogisticRegression(C=100, solver='liblinear')
```

5. Melakukan visualisasi dari proses training data sebelumnya.

[7]:
```
# Visualize the performance of the classifier
visualize_classifier(classifier1, X, y)
visualize_classifier(classifier2, X, y)
```





- Seperti yang terlihat pada hasil visualisasi, gambar tersebut merupakan grafik dari proses klasifikasi. Seperti yang sudah dijelaskan sebelumnya, logistic regression merupakan proses klasifikasi. Titik-titik tersebut merupakan variabel dependent dan area yang mengitari titik tersebut merupakan klasifikasi yang membedakan data-data tersebut. Nilai c pada fungsi LogisticRegression() merupakan nilai pinalti apa bila terjadi misklasifikasi sehingga algoritma akan menyesuaikan diri dengan data. Akan tetapi penggunaan nilai tersebut harus dipatasi, karena apabila data terlalu akurat maka akan terjadi overfitting sehingga data tidak tersebar dengan baik.

1.2 Decision trees

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

- Decission Tree adalah algoritma yang membati data set kedalam beberapa cabang lalu membuat keputusan sederhana pada tiap level cabangnya. Proses ini dicapai dengan cara melatih model agar mengetahui bagaimana cara membagi data dengan tepat. Algoritma ini akan membangun aturan berdasarkan data masukin dengan label targetnya.

1. Algoritma Decission Tree dapat digunakan dengan memanggil library sklearn.tree. Dipanggil juga library sklearn.model_selection untuk memanggil fungsi train_test_split

[9]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
```

2. Didefinisikan data yang akan digunakan untuk proses klasifikasi, data yang dipakai berasal dari file data_decision_trees.txt yang sudah didownload sebelumnya.

[10]:
```
# Load input data
input_file = 'data_decision_trees.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

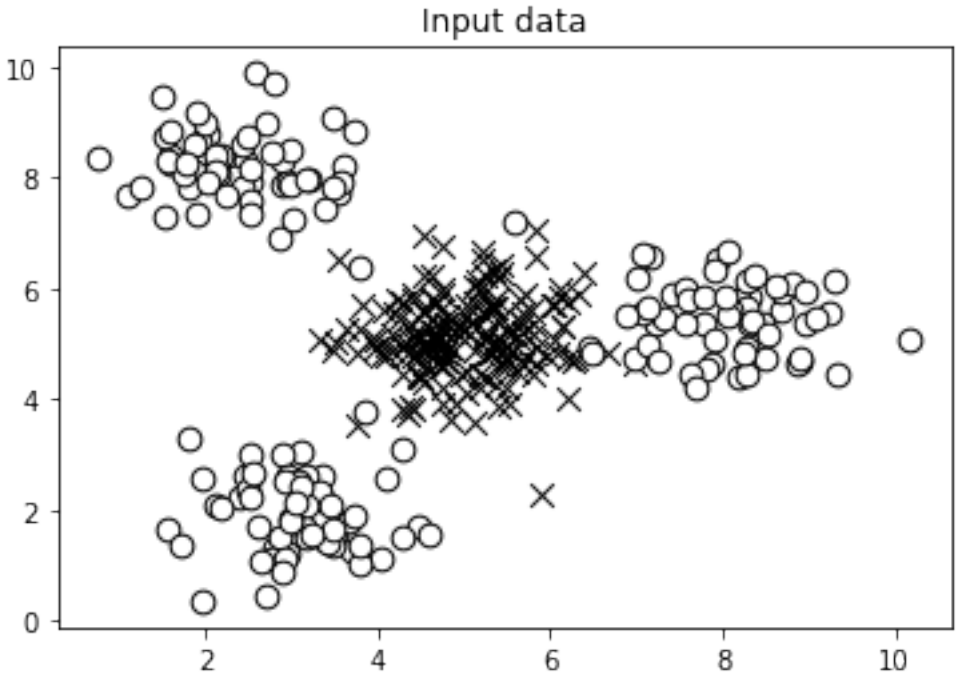
3. Membagi kelas pada data sesuai dengan label dari masing-masing data.

[12]:
```
# Separate input data into two classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

4. Melakukan visualisasi pada data input dengan menggualan matplotlib. Data ditampilkan dalam bentuk scatter plot.

[13]:
```
# Visualize input data
plt.figure()
plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
        edgecolors='black', linewidth=1, marker='x')
plt.scatter(class_1[:, 0], class_1[:, 1], s=75, facecolors='white',
        edgecolors='black', linewidth=1, marker='o')
plt.title('Input data')
```

[13]:
```
Text(0.5, 1.0, 'Input data')
```



- Grafik tersebut menunjukkan persebaran dari data masukan yang sudah didefinisikan tadi, dapat terlihat bahwa terdapat pola dimana beberapa data cenderung untuk saling berdekatan satu-sama lain.

5. Data dibagi menjadi data training dan data testing dengan menggunakan fungsi train_test_split(). Data training merupakan data yang digunakan untuk melatih model, sepentara testing merupakan data yang akan dipakai untuk menguji model yang telah dibuat. test_size menunjukkan rasio dari pembagian data, dimana 25% data akan digunakan sebagai test sedangkan sisanya akan digunakan untuk training.

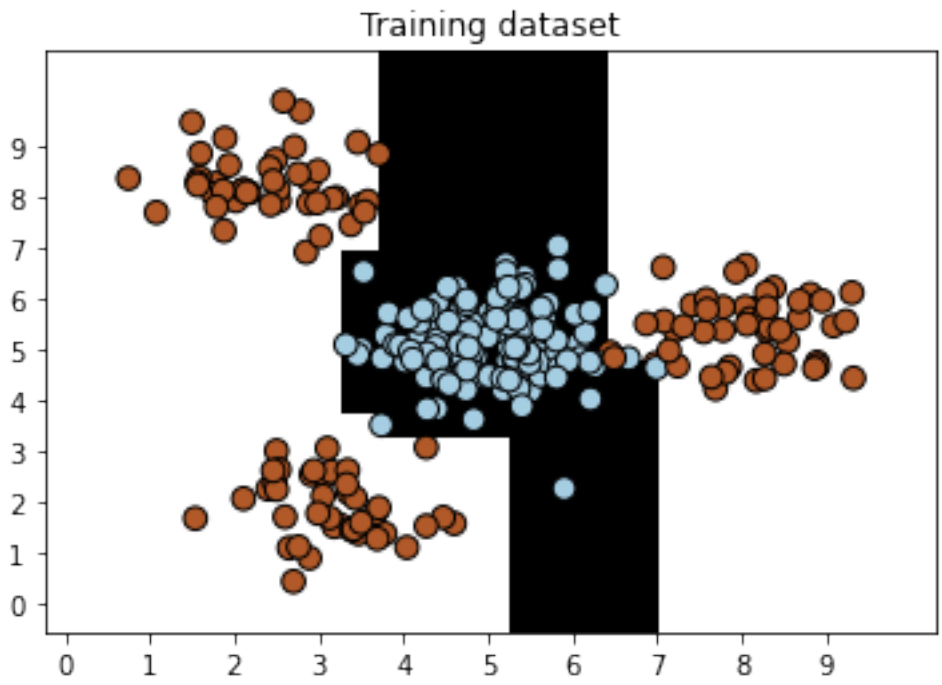
[14]:
```
# Split data into training and testing datasets
X_train, X_test, y_train, y_test = train_test_split(
        X, y, test_size=0.25, random_state=5)
```

6. Mendefinisikan parameter dan model yang digunakan, max_depth menunjukkan seberapa dalamnya level percabangan yang akan dimiliki oleh trees, sementara random_state menentukan nilai acak yang dimiliki oleh algoritma. Pemanggilan algoritma dilakukan dengan memanggil fungsi DecisionTreeClassifier() yang sebelumnya telah diimport.

[15]:
```
# Decision Trees classifier
params = {'random_state': 0, 'max_depth': 4}
classifier = DecisionTreeClassifier(**params)
```

7. Melakukan proses training menggunakan fungsi fit() menggunakan data X_train dan y_train yang sudah di split sebelumnya. Kemudian dilakukan proses visualisasi dengan menggunakan fungsi visualize_classifier().

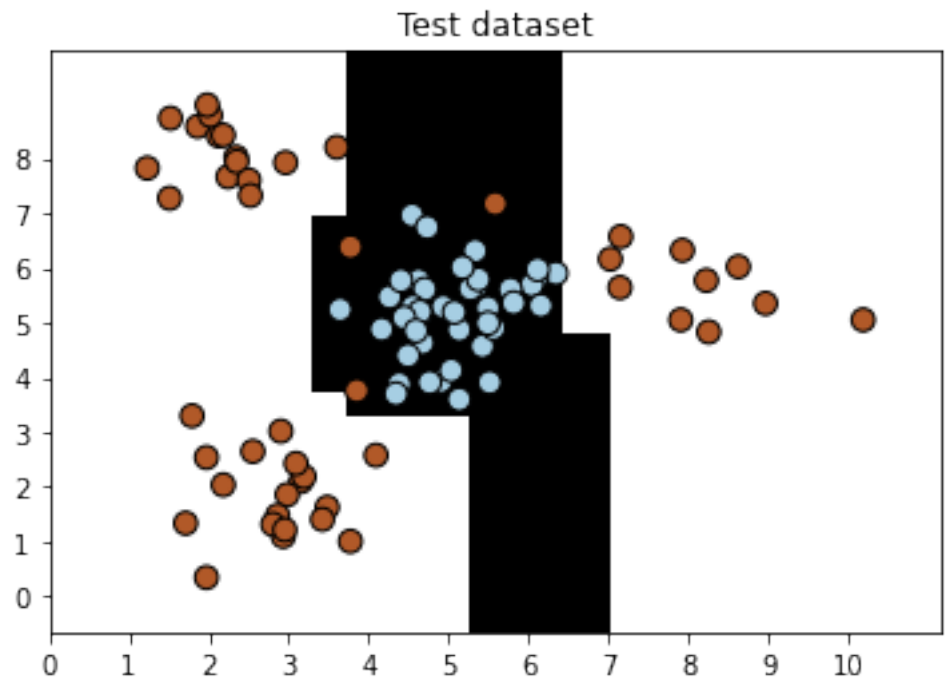
[16]:
```
classifier.fit(X_train, y_train)
visualize_classifier(classifier, X_train, y_train, 'Training dataset')
```



- Sama seperti pada logistic regression, grafik ini menujukkan proses klasifikasi yang dilakukan oleh decission tree. Apabila melihat grafik sebelumnya, dapat terlihat bahwa terdapat perbedaan pada data ditengah, setelah dilakukan proses klasifikasi terlihat bahwa data telah terbagi dan terpisah dari data yang lain.

8. Dilakukan proses prediksi menggunakan data testing, proses ini dilakukan untuk menguji hasil prediksi yang dimiliki oleh model untuk dilakukan evaluasi.

[17]:
```
y_test_pred = classifier.predict(X_test)
visualize_classifier(classifier, X_test, y_test, 'Test dataset')
```



- Seperti yang terlihat pada grafik, data yang hasil testing berhasil diklasifikas. Akan tetapi tidak semua data berhasil diprediksi karena terdapat beberapa data yang ikut tercampur.

9. Program ini digunakan untuk mengevaluasi performa dari proses prediksi yang dilakukan oleh model, untuk mendapatkan data-data evaluasi dapat digunakan fungsi classification_report()

[18]:
```
# 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_train),␣
 ↪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()
```

```
########################################

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

########################################
```

- Berdasarkan data tersebut terlihat bahwa algoritma prediksi memberikan persentase sekitar 97% pada presisi, recall dan akurasi.

1.3 Mean Shift

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

- Mean Shift adalah algoritma yang digunakan dalam unsupervised. Mean shift merupakan algoritma non-parametrik yang sering digunakan untuk klasifikasi. Tak seperti algoritma parametrik, pada algoritma ini tidak dilakukan proses amsumsi pada data-datanya.
- Tujuan utama dari algoritma ini adalah untuk mengidentifikasi lokasi dari centroids. Pada setiap titik didalam data training, dilakukan windowing untuk menghitung centroid. Proses ini terus dilakukan pada tiap titik didalam data dengan menggeser area untuk windowing.

1. Dilakukan pemanggilan libary yang dibutuhkan, fungsi untuk menjalankan algoritma Mean Shift dapat ditemukan dalam libary sklearn.cluster

[21]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle
```

2. Mendefinisikan data yang akan digunakan sebagai input.

[19]:
```
# Load data from input file
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

3. Menghitung estimasi bandwith untuk data independent. Bandwidth merupakan parameter yang menunjukkan estimasi kepadatan pada proses mean shift.

[22]:
```
# Estimate the bandwidth of X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))
```

4. Program ini berfungsi untuk melatih model clustering menggunakan fungsi MeanShift() memakai data yang sudah diestimasikan sebelumnya.

[23]:
```
# Cluster data with MeanShift
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seeding=True)
meanshift_model.fit(X)
```

[23]:
```
MeanShift(bandwidth=1.3044799765090382, bin_seeding=True)
```

5. Melakukan ekstraksi nilai tengah dari klaster yang sudah terbentuk sebelumynya.

[24]:
```
# Extract the centers of clusters
cluster_centers = meanshift_model.cluster_centers_
print('\nCenters of clusters:\n', cluster_centers)
```

```
Centers of clusters:
 [[2.95568966 1.95775862]
 [7.20690909 2.20836364]
 [2.17603774 8.03283019]
 [5.97960784 8.39078431]
 [4.99466667 4.65844444]]
```

6. Melakukan estimasi untuk mengetahui jumlah klaster yang sudah terbentuk sebelumnya.

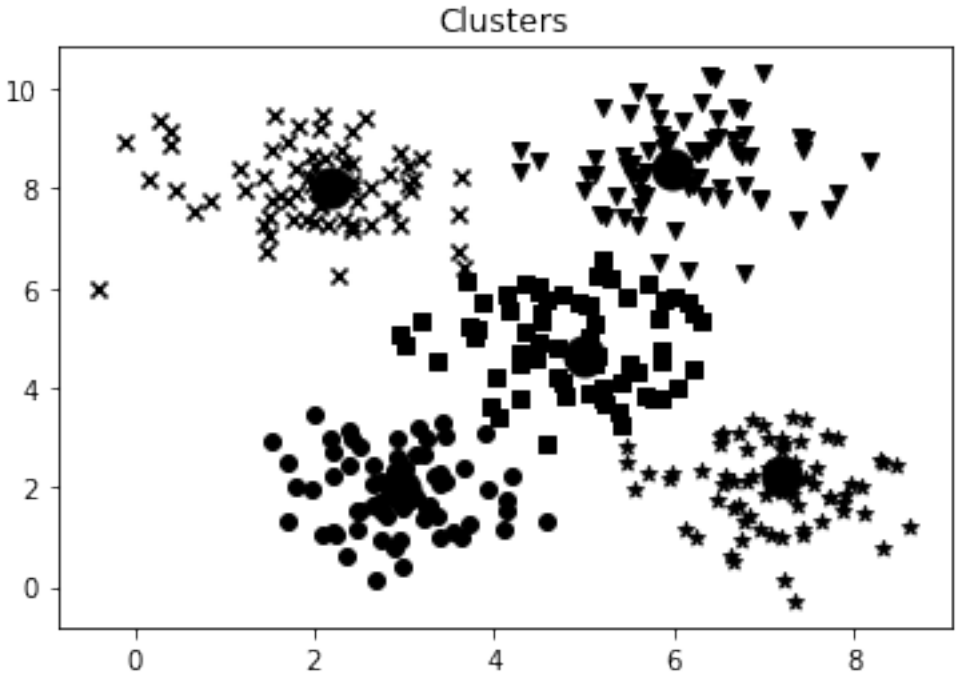
[25]:
```
# Estimate the number of clusters
labels = meanshift_model.labels_
num_clusters = len(np.unique(labels))
print("\nNumber of clusters in input data =", num_clusters)
```

```
Number of clusters in input data = 5
```

7. Melakukan visualisasi dari titik-titik data dan mengamber titik tengah dari klaster yang terbentuk.

[26]:
```
# 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=marker, 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 algoritma mean-shift telah melakukan klasifikasi berdasarkan data yang sudah diberikan, seperti yang dapat terlihat setiap objek yang saling berdekatan memiliki bentuk yang berbeda.

1.4 Nearest Neighbors Classifier

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

- K nearest neighbors classifier merupakan model algoritma klasifikasi yang menggunakan nilai dari tetangga terdekat untuk melakukan klasifikasi. Algoritma akan menentukan nilai K terdekat didalam dataset untuk mengidentifikasi kategori dari data masukan yang diberikan

dimana data akan dikaslifikasi berdasarkan nilai tertinggi yang diberikan oleh data-data terdekatnya.

1. Melakukan pemanggilan library yang dibutuhkan, fungsi untuk menjalankan algoritma KNN terdapat didalam library sklearn.neighbors.

[27]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
```

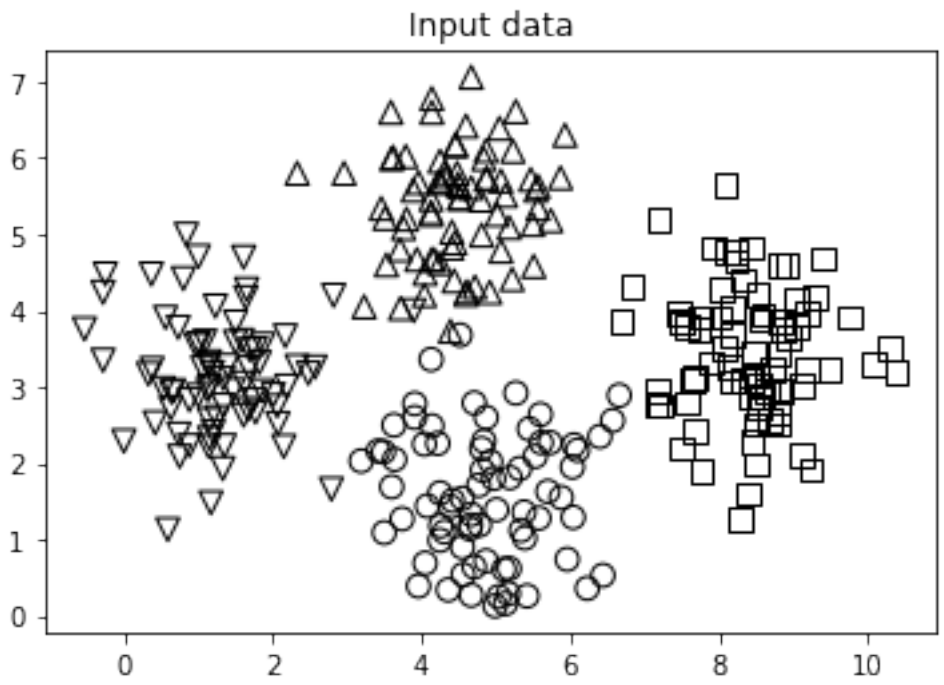
2. Mendefinisikan data yang akan digunakan sebagai input, data yang digunakan berasal dari file data.txt yang telah didownload sebelumnya. Berdasarkan data tersebut, dilakukan pembagian untuk variabel independent dan variabel dependentnya.

[28]:
```
# Load input data
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(np.int)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:4:
DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To
silence this warning, use `int` by itself. Doing this will not modify any
behavior and is safe. When replacing `np.int`, you may wish to use e.g.
`np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  after removing the cwd from sys.path.
```

3. Melakukan visualisasi dari data yang diberikan sebelumnya.

[29]:
```
# 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')
```



- Seperti yang diperlihatkan oleh grafik, terdapat 4 kelas yang dimiliki oleh data. Setiap data yang berbeda ditandai oleh label yang direpresentasikan oleh simbol yang berbeda.

4. Mendefinisikan banyaknya tetangga terdekat yang perlu dipertimbangkan oleh model.

[30]:
```
# Number of nearest neighbors
num_neighbors = 12
```

5. Membentuk model klasifikasi KNN dengan menggunakan fungsi KNeighborsClassifier()

[31]:
```
# Step size of the visualization grid
step_size = 0.01

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

6. Melatih model menggunakan training data yang sudah didefiniskan sebelumnya.

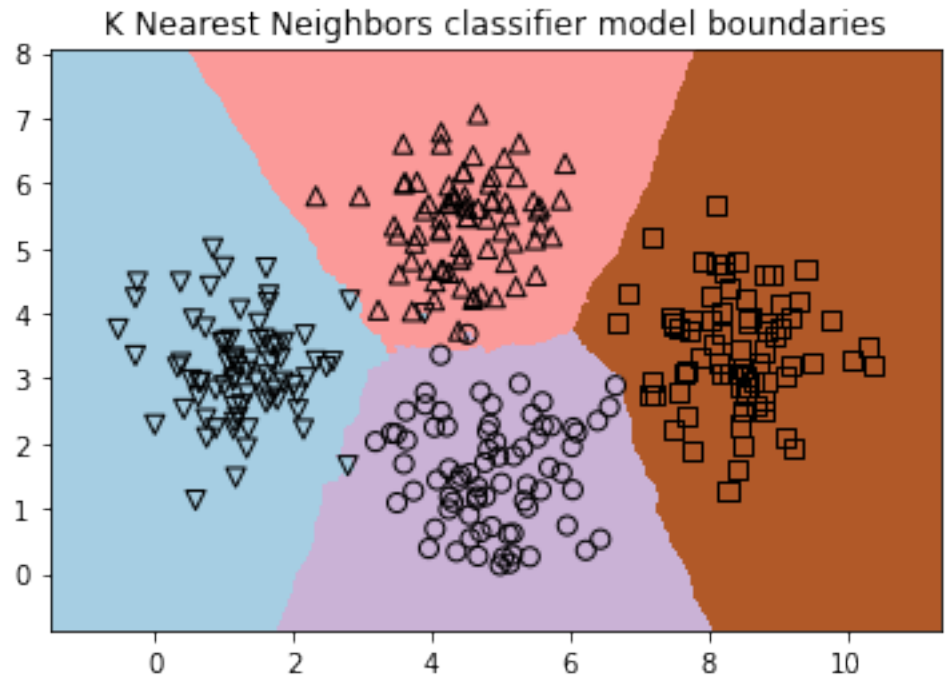
[32]:
```
# Train the K Nearest Neighbours model
classifier.fit(X, y)
```

[32]:
```
KNeighborsClassifier(n_neighbors=12, weights='distance')
```

7. Melakukan visualisasi data untuk menggambarkan perbatasan yang dimiliki oleh setiap kelas pada data.

[33]:
```
# 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')
```

[33]:
```
Text(0.5, 1.0, 'K Nearest Neighbors classifier model boundaries')
```

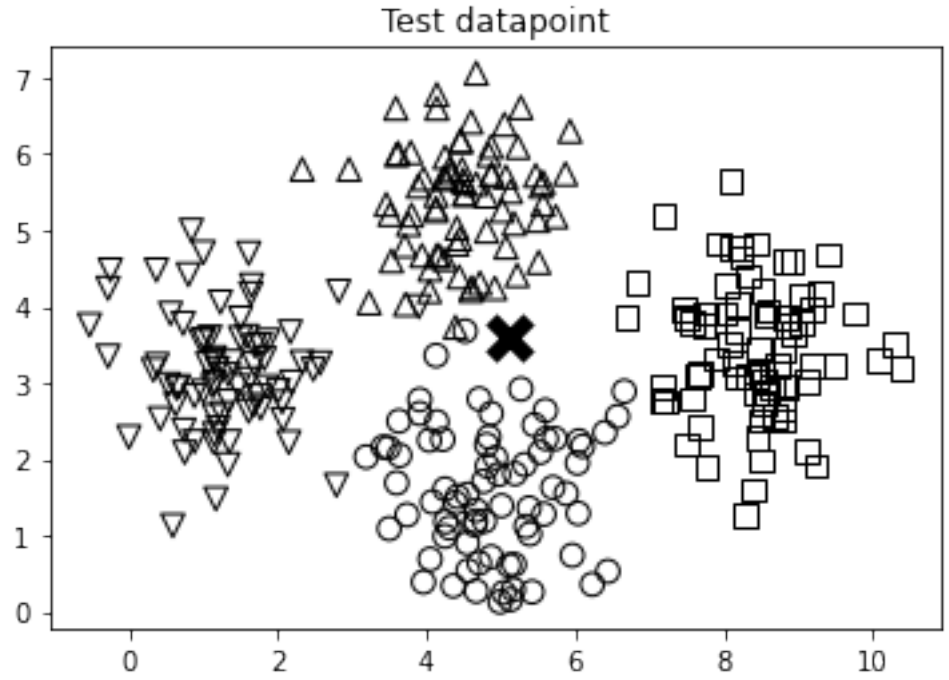


- Seperti yang terlihat pada grafik, setiap label yang berbeda memiliki kawasan-kawasan tersendiri. Area tersebut merupakan visualisasi dari dari hasil proses klasifikasi dari kelas-kelas yang berbeda dari tiap data.

8. Melakukan visualisasi dari lokasi data yang digunakan sebagai data masukkan.

[34]:
```
# 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')
```

[34]:
```
<matplotlib.collections.PathCollection at 0x7f0ff2a25c50>
```



- Seperti yang diperlihatkan pada grafik, titik X merupakan posisi dari data yang baru saja dimasukan.

9. Berdasarkan nilai tersebut, ditentukkan nilai K dari data terdekat.

[35]:
```
# Extract the K nearest neighbors
_, indices = classifier.kneighbors([test_datapoint])
```

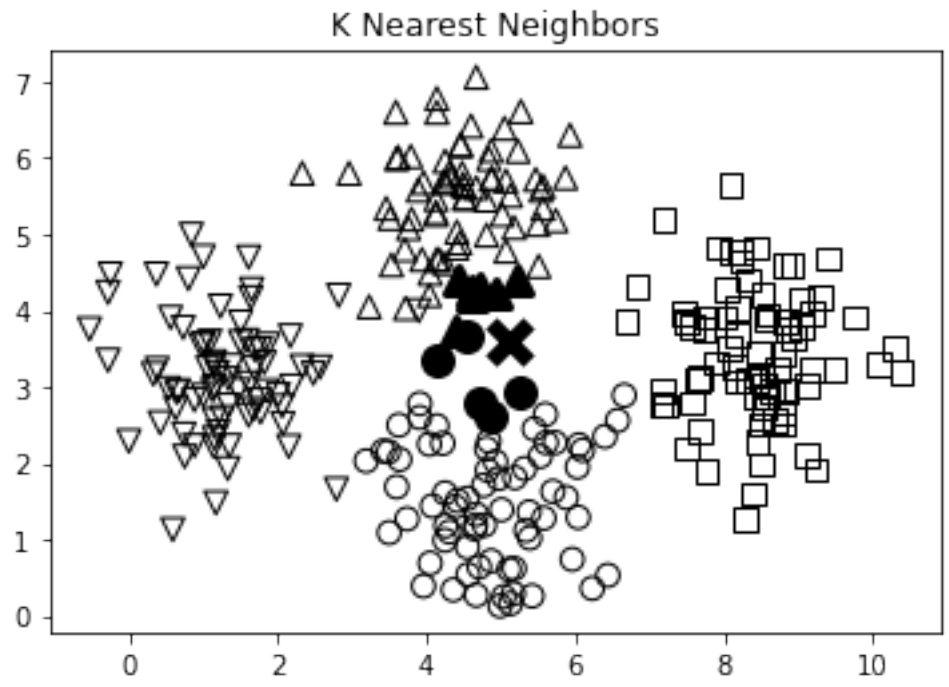
```
indices = indices.astype(np.int)[0]
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:3:
DeprecationWarning: `np.int` is a deprecated alias for the builtin `int`. To
silence this warning, use `int` by itself. Doing this will not modify any
behavior and is safe. When replacing `np.int`, you may wish to use e.g.
`np.int64` or `np.int32` to specify the precision. If you wish to review your
current use, check the release note link for additional information.
Deprecated in NumPy 1.20; for more details and guidance:
https://numpy.org/devdocs/release/1.20.0-notes.html#deprecations
  This is separate from the ipykernel package so we can avoid doing imports
until
```

10. Melakukan visualisasi dari data yang memiliki nilai terdekat dengan data masukkan sebelumnya.

[36]:
```
# 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()
```

```
Predicted output: 1
```



- Seperti yang terlihat, data dengan nilai K terdekat ditunjukkan oleh simbol yang ditebalkan. Bisa dilihat terdapat sekitar 12 objek yang dihitamkan, itu karena sebelumnya sudah ditentukan banyaknya tetangga yang diperhitungkan adalah sebanyak 12. Berdasarkan hasil visualisasi tersebut, dapat terlihat jelas bahwa algoritma K Nearest Neighboiur melakukan klasifikasi dengan memperhitungkan nilai yang dimiliki oleh data yang paling dekat dengan data masukkannya.

1.5 States

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

- States disini bukan merupakan nama sebuah algoritma, melainkan nama dari * program yang berfungsi untuk mengetahui hubungan antara propinsi di suatu negara.
- Dalam perancangan program tersebut digunakan sebuah pendekatan yang dinamakan logic programming. Pendekatan tersebut didasarkan pada penggunaan logika formal untuk mengungkapkan fakta atau aturan dari suatu permasalahan. Berikut ini merupakan penerapan dari pendekatan tersebut.
- Program states.py memiliki tujuan untuk menganalisa geografi dari berbagai propinsi pada amerika serikat, dimana program akan dapat memberikan jawaban berdasarkan fakta dan aturan yang sudah didefinisikan melalui pendekatan logic programming.

1. Melakukan import library logpy, di dalam library tersebut berisi fungsi-fungsi yang berguna dalam melakukan logic programming.

[40]:
```
from logpy import run, fact, eq, Relation, var
```

2. Melakukan inisiasi hubungan dan mendefinisikan file-file yang berisikan data-data yang diperlukan.

[41]:
```
adjacent = Relation()
coastal = Relation()
file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'
```

3. Membaca file yang diberikan sebelumnya, lalu ditambahkan fakta-fakta kedalam base berdasarkan data-data yang ada didalamnya.

[42]:
```
# 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 adjacent 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)
```

4. Inisiasi variabel x dan y

[44]:
```
# Initialize the variables
x = var()
y = var()
```

5. Program ini berfungsi untuk memberikan pertanyaan-pertanyaan pada algoritma yang telah dibuat. Pada bagian ini kita menanyakan apakah Nevada Bersebelahan dengan Loisiana

[45]:
```
# 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')
```

```
Is Nevada adjacent to Louisiana?:
```

```
No
```

6. Pada bagian ini program akan menampilkan propinsi mana saja yang bersebalahan dengan oregon.

[50]:
```
# States adjacent to Oregon
output = run(0, x, adjacent('Oregon', x))
print('\nList of states adjacent to Oregon:')
for item in output:
    print(item)
```

```
List of states adjacent to Oregon:
Nevada
California
Idaho
Washington
```

7. Program berikut akan menampilkan propinsi mana saja yang bersebalahn dengan Mississippi dan berada dipesisir

[51]:
```
# 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 coastal states adjacent to Mississippi:
Alabama
Louisiana
```

8. Menghitung ada berapa propinsi yang bersebelahan dengan propinsi yang berada dipesisir

[48]:
```
# 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 7 states that border a coastal state:
North Dakota
Massachusetts
Indiana
Delaware
Florida
New Hampshire
Arkansas
```

9. Melihat daftar propinsi yang bersebalahan berdasarkan 2 propinsi yang diberikan. Dalam program ini diberikan masukan Arkansas dan Kentucky

[49]:
```
# List of states that adjacent to the two given states
output = run(0, x, adjacent('Arkansas', x), adjacent('Kentucky', x))
print('\nList of states that are adjacent to Arkansas and Kentucky:')
for item in output:
    print(item)
```

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