Aldi Riza Rahnawan
3332190048
Kecerdasan Buatan A

Ulangan Tengah Semester

1. Logistic Regression

In [1]:
```
import numpy as np
from sklearn import linear_model
import matplotlib.pyplot as plt
```

- **import numpy** digunakan untuk memanggil library numpy yang berfungsi untuk mengolah array, matriks, dan linear aljabar.
- **sklearn** atau scikit-learn adalah library pada python yang digunakan dalam machine learning. Dalam kasus kali ini digunakan *linear_model* untuk melakukan prediksi dalam kasus-kasus linear.
- **matplotlib** adalah library pada python yang digunakan untuk memvisualisasikan data

In [2]:
```
from utilities import visualize_classifier
```

- Perintah diatas digunakan untuk memvisualisasikan model classifier yang telah dibuat.

In [3]:
```
# 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])
```

- Listing ini digunakan untuk mendeskripsikan data vektor 2 dimensi pada variabel X dan y, variabel X akan digunakan sebagai **predictor(variabel independen)** dan variabel y akan digunakan sebagai **target(variabel dependen)**.
- Data yang digunakan adalah jenis numpy array.

In [4]:
```
# Create the logistic regression classifier
classifier = linear_model.LogisticRegression(solver='liblinear', C=1)
classifier_2 = linear_model.LogisticRegression(solver='liblinear', C=100)
```

- **linear_model.LogisticRegression** adalah perintah yang digunakan untuk membuat suatu model klasifikasi Logistic Regression. Logistic Regression sendiri adalah metode yang digunakan untuk menjelaskan hubungan antara variabel input (independen variabel) dan variabel output (dependen variabel). Logistic Regression mengestimasi hubungan antara variabel independen dan dependen menggunakan probababilitas dengan fungsi linear. Fungsi logistic berbentuk kurva sigmoid yang mengklasifikasikan sebuah data kedalam kondisi True atau False.
- Pada kode diatas digunakan **liblinear** yang meruoakan library untuk melakukan klasifikasi secara linear dengan menggunakan metode Coordinat Descendant.

- Kemudian ada nilai **C** yang digunakan untuk memberikan pinalty atas kesalahan klasifikasi, sehingga model akan lebih menyesuaikan terhadap data dan akan mendapatkan garis pembatas antara class yang semakin akurat. Semakin besar nilai **C** maka akan semakin akurat, namun perlu diperhatikan bahwa jika nilai **C** terlalu besar dapat menyebabkan overfit pada model. Kondisi overfit ini adalah kondisi dimana model memiliki akurasi tinggi terhadap data training, namun memiliki performa yang rendah pada saat dilakukan test.
- Pada liting code diatas saya membuat dua buah model classifier dengan masing-masing **C=1** dan **C=100**. Hal ini akan digunakan sebagai perbandingan.

In [5]:
```
# Train the classifier C=1
classifier.fit(X, y)
```

Out[5]:
```
LogisticRegression(C=1, solver='liblinear')
```

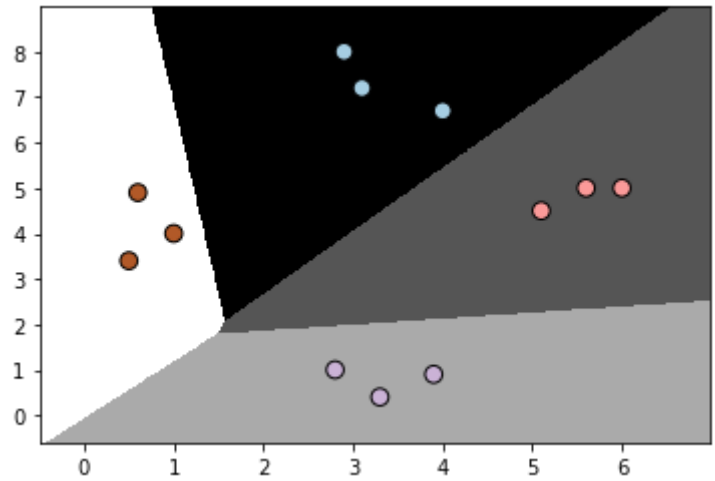
- **classifier.fit(X, y)** adalah perintah yang digunakan untuk melakukan training model menggunakan data yang telah di definisikan sebelumnya, yaitu **X** sebagai variabel independen dan **y** sebagai variabel dependen.

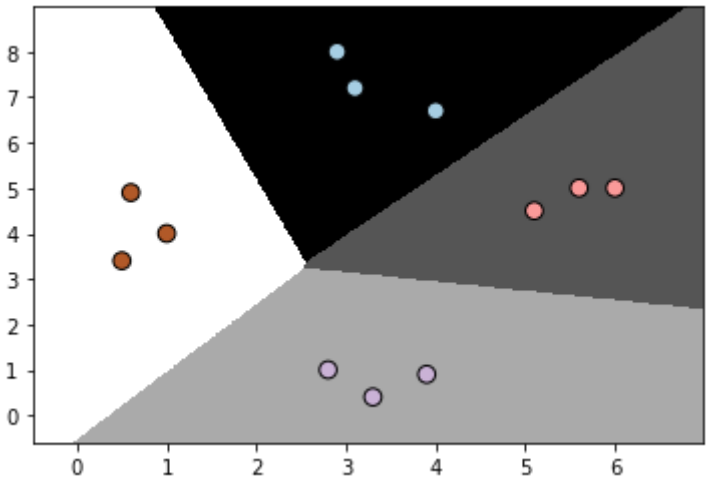
In [6]:
```
# Train The Classifier C=2
classifier_2.fit(X, y)
```

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

- sama seperti perintah sebelumnya **classifier_2.fit** juga digunakan untuk melakukan training pada model.

In [7]:
```
# Visualize the performance of the classifier
visualize_classifier(classifier, X, y)
visualize_classifier(classifier_2, X, y)
```





- Visualize Classifier digunakan untuk memvisualkan klasifikasi yang telah dibuat oleh model.
- Grafik diatas membagi klasifikasi menjadi 4 bagian, hal ini sesuai dengan parameter yang ada pada variabel y yaitu 0,1,2,3. Parameter inilah yang digunakan sebagai terget klasfikasi. Garis-garis perbedaan warna yang ada pada grafik merupakan batas antar kelas dari targetnya. Disini terlihat bahwa tiap kelas target memiliki 3 buah data.
- Pada model klasifikasi ini juga dibandingkan antara model logistic regression yang menggunakan C=1 dan C=100. Dapat dilihat bahwa grafik dari model yang menggunakan nilai C=100 memiliki garis pembatas yang lebih akurat dan presisi satu sama lain.

2. Decision Tree Classifier

Decision Tree merupakan salah satu algoritma yang masuk kedalam Supervised Learning. Algoritma Decision Tree adalah algoritma klasifikasi yang membagi dataset menjadi beberapa cabang untuk mempermudah pengambilan keputusan. Bisa dikatakan algoritma ini hampir mirip dengan flow chart dengan struktur if-else condition. Pada Algoritma ini terdapat beberapa bagian antara lain yaitu:
-Root Node
-Internal Node
-Edge
-Leaf Node

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

from utilities import visualize_classifier
```

- **import numpy** digunakan untuk memanggil library numpy yang berfungsi untuk mengolah array, matriks, dan linear aljabar.
- **matplotlib** adalah library pada python yang digunakan untuk memvisualisasikan data
- **sklearn** atau scikit-learn adalah library pada python yang digunakan dalam machine learning. Library ini banyak memuat model-model machine learning. Dalam kasus kali ini digunakan model **DecisionTreeClassifier**.
- Pada program ini juga digunakan **train_test_split** untuk memecah dataset menjadi dua bagian, yaitu data yang digunakan untuk training dan data yang digunakan untuk melakukan test. Kedua data ini perlu dibedakan untuk melakukan pengujian pada model. Dengan menggunakan data training model akan mempelajari pola klasfikasinya dan dengan menggunakan data test model akan mencoba menerapkan pola yang telah dipelajari untuk mengklasifikasikan data.
- Pemilihan Node pada Decision Tree didasarkan pada data yang memiliki Entropy atau Gini Index dengan nilai terbaik.
- **classification_report** digunakan untuk melihat performa dari model yang telah kita buat. Mulai dari Precission, Recall, f1-score dan Accuracy.

In [2]:
```
# Load input data
input_file = 'data_decision_trees.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1]
```

- Perintah diatas merupakan perintah untuk mendefinisikan data yang akan digunakan sebagai masukan bagi model.
- Data yang digunakan adalah **data_decision_trees.txt**
- Data ini kemudian dipecah menjadi dua bagian yaitu **X** dan **y**, pemecahan ini menggunakan teknik **slicing array**. **X** mengambil data dari index 0 - dengan -2 yang akan digunakan

sebagai predictor(variabel independent). Sedangkan **y** akan mengambil data dengan index -1 yang digunakan sebagai target(variabel dependen).

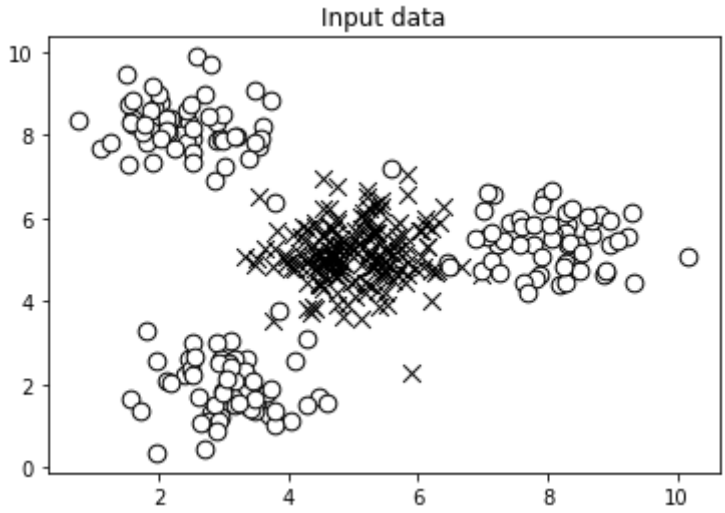
In [3]:
```
# Separate input data into two classes based on labels
class_0 = np.array(X[y==0])
class_1 = np.array(X[y==1])
```

- Data **X** kemudian dipecah lagi menjadi 2 bagian berdasarkan labelnya. Yaitu data **class_0** dengan nilai label **y = 0** dan data **class_1** dengan nilai label **y = 1**.
- Pemecahan ini dilakukan agar mempermudah pengambilan keputusan untuk tiap-tiap percabangan.

In [4]:
```
# 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')
```

```
C:\Users\Aldi Riza\AppData\Local\Temp\ipykernel_15184\3606243706.py:3: UserWarning:
You passed a edgecolor/edgecolors ('black') for an unfilled marker ('x').  Matplotli
b is ignoring the edgecolor in favor of the facecolor.  This behavior may change in
the future.
  plt.scatter(class_0[:, 0], class_0[:, 1], s=75, facecolors='black',
```

Out[4]:
```
Text(0.5, 1.0, 'Input data')
```



- Visualisasi menggunakan scatter plot diatas menggambarkan sebaran/distribusi data dengan label 0 dan 1. Label 0 digambarkan dengan **X** dan label 1 digambarkan dengan **o**.
- Dapat dilihat bahwa persebaran data dengan label 0 lebih mengumpul ditengah, sedangkan data dengan label 1 lebih banyak menyebar disekitar data berlabel 0.

In [5]:
```
# 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)
```

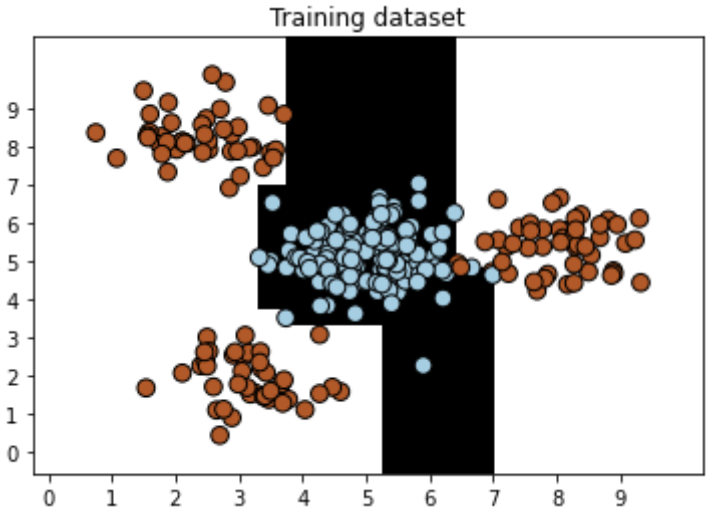
- Pada perintah **train_test_split** diatas data **X** dan **y** akan displit/dibagi menjadi **data train** dan **data test**. Data test memiliki ratio 25% dari keseluruhan data, sedangkan 75% sisanya

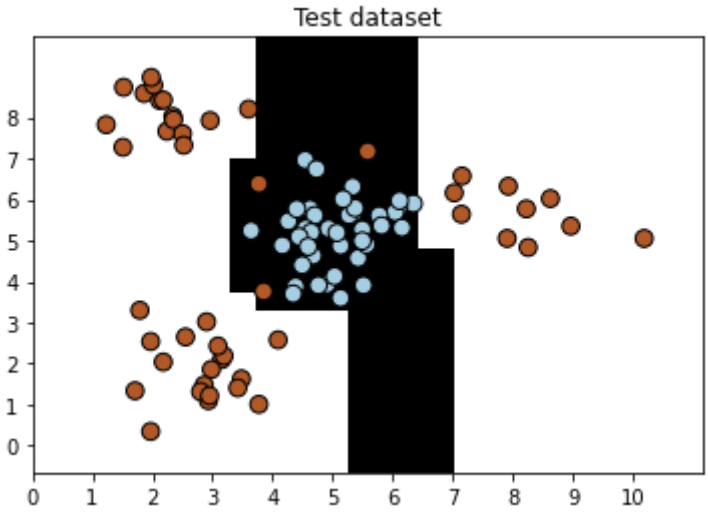
akan digunakan sebagai data training.

- Pada listing juga digunakan **random_state = 5**, hal ini dimaksudkan agar data dipecah secara random dan tidak berurutan yang bertujuan untuk mengindari pola data yang monoton dan dapat menyebabkan terjadinya overfit.

In [6]:

```
# 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 dataset')
```





- Pada program diatas dilakukan training terhadap model dengan menggunakan data training **X_train** dan **y_train**. Data **X_train** digunakan sebagai prediktor dan **y_train** digunakan sebagai targetnya.
- Setelah dilakukan training, kemudian dilakukan testing model menggunakan data set **X_test** dan **y_test**.
- Hasil Training dan test ini kemudian dilakukan visualisasi untuk melihat batasan antara label 0 dan label 1. Dapat telihat disini, daerah yang hitam merupakan daerah klasifikasi dari label bernilai 0 dan daerah berwarna putih adalah daerah klasifikasi dari label bernilai 1.
- Jika melihat perbandingan hasil Training dan hasil Test diatas, bisa dikatakan model sudah mengklasifikasikan dengan baik.

In [7]:

```
# 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
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

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

- Setelah model selesai melewati tahapan Training dan Testing, selanjutnya dilakukan pengecekan performa dari model dengan melihat nilai Precission, Recall, F1-Score dan Accuracynya.
- Precision menggambarkan seberapa besar accuracy dari sebuah model dalam melakukan klasifikasi secara benar dan recall mengacu pada jumlah items yang diklasifikasikan dengan benar dari keseluruhan items. F1-Score adalah rata-rata harmonik dai Precission dan Recall yang menggambarkan keseimbangan performa dari precission dan recall itu sendiri.
- Pada model decision tree yang dibuat ini, nilai Precission dan Recall nya sudah sangat bagus bahkan mendakati nilai 1. Dengan nilai tersebut bisa dikatakan bahwa model ini dapat mengklasifikasikan data dengan sempurna.

3. Mean Shift

Mean Shift adalah salah satu algoritma Unsupervised Learning (Pembelajaran Tanpa Pengawasan). Mean Shift memiliki cara kerja dengan menghitung semua titik yang ada pada data untuk mencari nilai mean(rata-rata) yang akan digunakan sebagai pusat(center) kluster nya. Perhitungan mean dilakukan untuk tiap data yang berdekatan dan masih memenuhi batas yang telah ditentukan. Nilai mean ini akan menjadi label dan digunakan sebagai pusat cluster. Proses seperti ini terus diulang sampai semua data konvergen.

In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import MeanShift, estimate_bandwidth
from itertools import cycle
```

- **import numpy** digunakan untuk memanggil library numpy yang berfungsi untuk mengolah array, matriks, dan linear aljabar.
- **matplotlib** adalah library pada python yang digunakan untuk memvisualisasikan data
- **sklearn** atau scikit-learn adalah library pada python yang digunakan dalam machine learning. Library ini banyak memuat model-model machine learning. Dalam kasus kali ini digunakan model **Mean Shift** yang digunakan untuk melakukan clustering.
- Pada program ini juga digunakan **cycle** yang merupakan bagian dari module **itertools**. **cycle** digunakan untuk melakukan perulangan sampai semua data mencapai konvergen.

In [2]:
```
# Load data from input file
X = np.loadtxt('data_clustering.txt', delimiter=',')
```

- Pada perintah diatas didefinisikan data **X** yang akan digunakan sebagai masukan untuk model Mean Shift ini.
- Data **X** akan dilakukan clustering untuk memetakan data ke label-label tertentu.

In [3]:
```
# Estimate the bandwidth of X
bandwidth_X = estimate_bandwidth(X, quantile=0.1, n_samples=len(X))
```

- **estimate_bandwidth** digunakan untuk menentukan batasan atau jumlah data yang akan dihitung pada setiap cycle nya. Dalam hal ini quantile yang digunakan adalah 0.1.

In [4]:
```
# Cluster data with MeanShift
meanshift_model = MeanShift(bandwidth=bandwidth_X, bin_seeding=True)
meanshift_model.fit(X)
```

Out[4]:
```
MeanShift(bandwidth=1.3044799765090382, bin_seeding=True)
```

- Pada listing diatas, dilakukan pemodelan Mean Shift untuk melakukan clustering pada data X
- Selanjutnya dilakukan juga Training model menggunakan perintah **.fit** dengan data masukan **X**

In [5]:
```
# 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]]
```

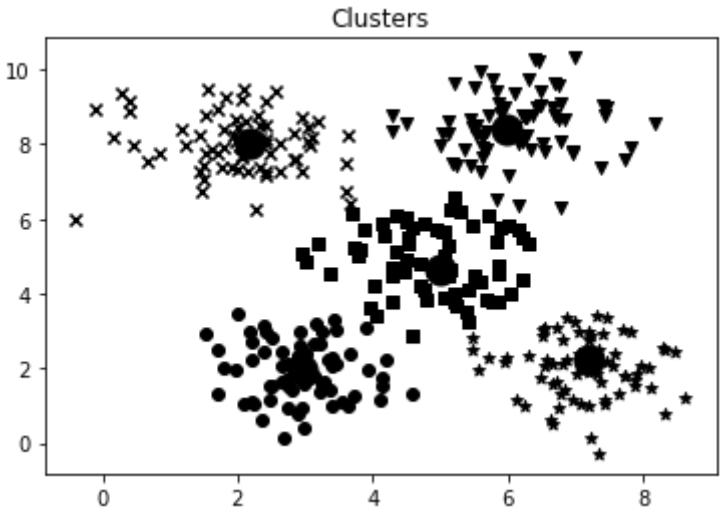
- Dari hasill Pemodelan dan Training diatas didapatkan bahwa data X memiliki 5 label atau pusat cluster dengan titik koordinat(x,y) ditampilkan diatas.

In [6]:
```
# 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
```

- Jumlah cluster diestimasikan berjumlah 5, artinya terdapat 5 titik pusat cluster pada data **X**.

In [7]:
```
# 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()
```



- Berikut adalah hasil clustering yang didapatkan, terdapat 5 titik pusat cluster yang memiliki titik-titik data sebagai anggota class nya.
- Beberapa cluster ini digambarkan dengan simbol **x**,**segitiga**,**kotak**,**lingkaran** dan **bintang**.

- Setiap anggota cluster akan terpusat menuju titik pusat clusternya. Kondisi inilah yang dinamakan **Konvergen**

4. K-Nearest Neighbor Classification

Nearest Neighbor Classifier belajar berdasarkan analogi dengan cara mengkomparasikan tuple data test yang diberikan dengan tuple data training yang memiliki kemiripan. Komparasi ini diatur dengan variabel **k**, jika **k = 1** maka tuple data yang tidak diketahui akan ditaruh pada kelas tuple training yang paling dekat dan memiliki kesamaan pola. Semakin besar nilai **k** maka akan semakin baik klasifikasinya, namun hal ini berpengaruh juga terhadap komputasi yang semakin berat. Untuk itu guna menemukan nilai **k** yang paling tepat biasanya digunaka **cross validation**. Untuk mengitung jarak terdekat dan kesamaan pola ini umumnya digunakan persamaan **Euclidean Distance**.

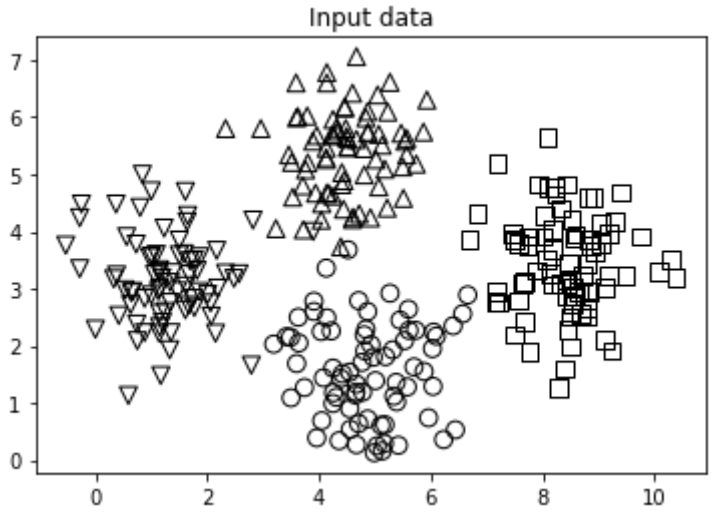
In [1]:
```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from sklearn import neighbors, datasets
```

- **import numpy** digunakan untuk memanggil library numpy yang berfungsi untuk mengolah array, matriks, dan linear aljabar.
- **matplotlib** adalah library pada python yang digunakan untuk memvisualisasikan data.
- **sklearn** atau scikit-learn adalah library pada python yang digunakan dalam machine learning. Library ini banyak memuat model-model machine learning. Dalam kasus kali ini digunakan model **neighbors**.

In [2]:
```
# Load input data
input_file = 'data.txt'
data = np.loadtxt(input_file, delimiter=',')
X, y = data[:, :-1], data[:, -1].astype(int)
```

- Perintah diatas merupakan perintah untuk mendefinisikan data yang akan digunakan sebagai masukan bagi model.
- Data yang digunakan adalah **data.txt**
- Data ini kemudian dipecah menjadi dua bagian yaitu **X** dan **y**, pemecahan ini menggunakan teknik **slicing array**. **X** mengambil data dari index 0 - dengan -2 yang akan digunakan sebagai predictor(variabel independent). Sedangkan **y** akan mengambil data dengan index -1 yang digunakan sebagai target(variabel dependen).
- Semua data ini diubah menjadi bentuk **integer**.

In [3]:
```
# 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')
```



- Data yang telah didefinisikan tadi, kemudian divisualisasikan enagn scatter plot untuk melihat distribusi datanya.
- Dari grafik diatas terlihat bahwa data yang digunakan memiliki 4 buah label

In [4]:
```
# Number of nearest neighbors
num_neighbors = 12
```

- Nilai **k (neigbors)** didefinisikan sebesar 12, artinya kita akan mengambil 12 titik terdekat dari data inputan yang kita masukkan.

In [5]:
```
# Step size of the visualization grid
step_size = 0.01
```

- **step_size** digunakan untuk menentukan step dalam proses komparasi untuk mencari nilai data terdekat yang memiliki kemiripan.

In [6]:
```
# Create a K Nearest Neighbours classifier model
classifier = neighbors.KNeighborsClassifier(num_neighbors, weights='distance')
```

- Pada tahap ini dibuat sebuah model **K-Nearest Neighbor (KNN)** dengan parameter nilai **k** dan pembobotan berupa jarak.

In [7]:
```
# Train the K Nearest Neighbours model
classifier.fit(X, y)
```

Out[7]:
```
KNeighborsClassifier(n_neighbors=12, weights='distance')
```

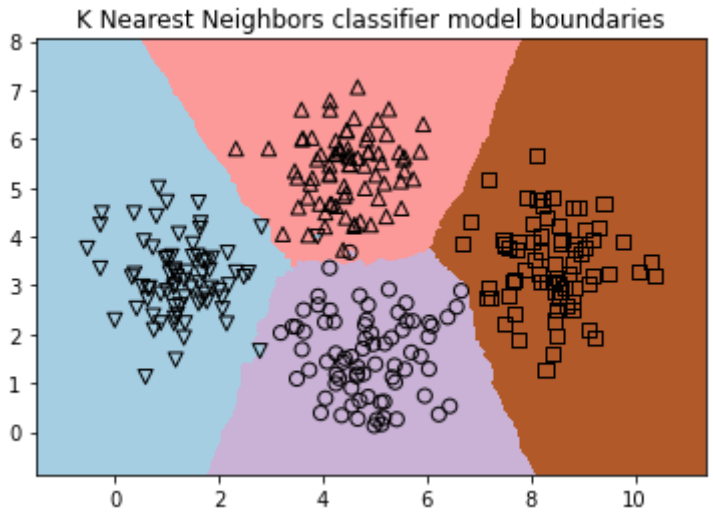
- Dengan Model yang telah dibuat tadi kemudian dilakukan training menggunakan data **X** sebagai predictor dan data **y** sebagai labelnya.

In [8]:
```
# 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))
```

- Untuk melihat hasil training model, dibuat sebuah mesh untuk visualisasi agar batasan antar label(target) dapat terlihat, sehingga memudahkan proses analisa.

In [9]:
```
# 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')
```

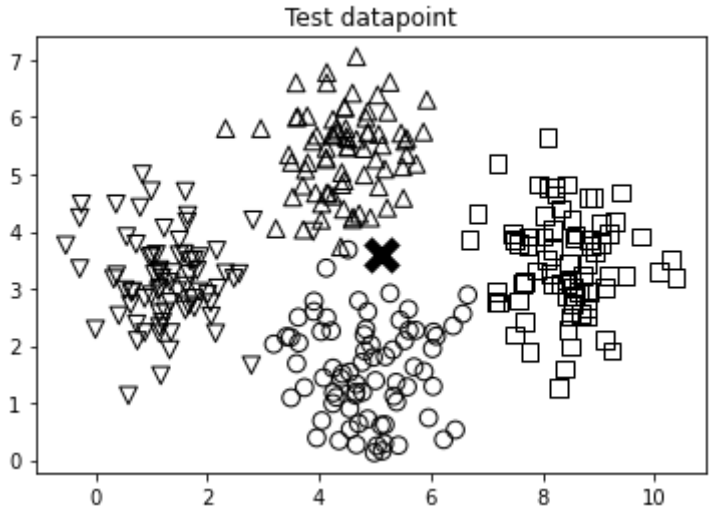
Out[9]:
```
Text(0.5, 1.0, 'K Nearest Neighbors classifier model boundaries')
```



- Grafik diatas merupakan batasan antar label yang sudah diklasifikasikan dengan menggunakan model KNN.
- Terlihat bahwa ada 4 area dengan warna yang berbeda-beda, setiap warna merepresentasikan class dari label yang berbeda.

In [10]:
```
# 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')
```

Out[10]:
```
<matplotlib.collections.PathCollection at 0x20b811cb9a0>
```

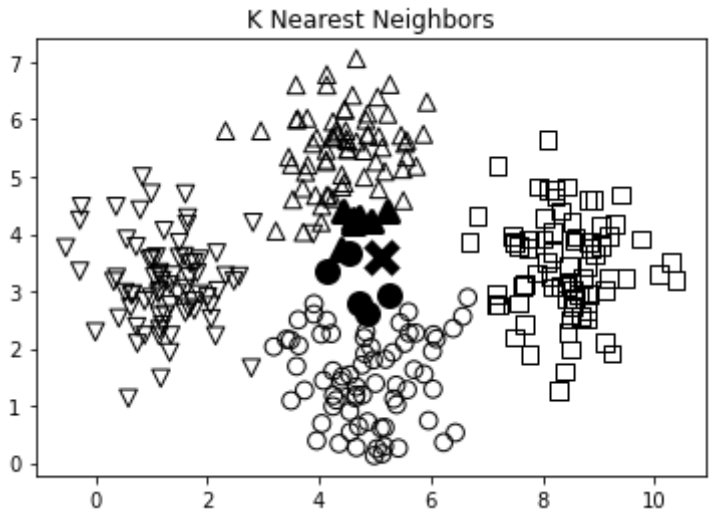


- Dengan menggunakan sebuah input baru, akan dilakukan ujicoba terhadap model KNN ini.
- Titik data tersebut digambarkan dengan tanda **x**, titik data ini akan mencari 12 titik data lainnya yang paling berdekatan dan memiliki kemiripan.

In [11]:

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

```
Predicted output: 1
```



- Disini terlihat bahwa ada 12 titik data disekitar **x** yang di **bold**. Titik data inilah yang dianggap paling dekat dan memiliki kemiripan dengn titik **x**.
- Kemudian dilakukan perhitungan untuk melihat label mana yang memiliki titik terseleksi lebih banyak. Dalam kasus ini yang dibandingkan adalah label **segitiga** dan label **lingkaran**.
- **Label lingkaran** hanya memiliki **5** titik data yang terseleksi, sedangkan **label segitiga** memiliki **7** titik data yang terseleksi. Sehingga dapat disimpulkan bahwa titik data **x** masuk kedalam class **label segitiga** atau dalam hal ini **y = 1**.

5. States (Logic Programming)

States adalah salah satu bentuk penerapan dari Logic Programming, dimana dalam kasus ini digunakan data tentang negara bagian yang saling bersebelahan dan negara bagian yang ada di pesisir pantai Amerika Serikat.

Logic Programming sendiri adalah sebuah pemrograman dimana komputasi yang dilakukan terjadi secara otomatis didasarkan pada database yang memuat fakta dan aturan tertentu. Logic Programming menggunakan metode pencocokan variabel dengan item yang berbeda berdasarkan fakta dan aturan tertentu yang sudah didefinisikan sebelumnya, proses ini dinamakan dengan unifikasi. Hubungan antar variabel dinamakan sebagai relasi, jika semua relasinya sesuai dengan aturan dan fakta, maka relasinya dinyatakan dengan benar.

In [1]:
```
from logpy import run, fact, eq, Relation, var
```

- **logpy** adalah library yang digunakan untuk melakukan pemrograman logic dan relasional pada python.
- Pada listing diatas juga terdapat beberapa fungsi seperti run, fact, eq, Relation dan var yang digunakan untuk melakukan operasi logic dan relasional.

In [2]:
```
adjacent = Relation()
coastal = Relation()
```

- Listing diatas mendefinisikan function **Relation()** pada variabel **adjacent** dan **coastal**.
- Function ini akan digunakan untuk melihat relasi antar kedua variabel tersebut.

In [3]:
```
file_coastal = 'coastal_states.txt'
file_adjacent = 'adjacent_states.txt'
```

- Data **file_coastal** didefinisikan dari **coastal_states.txt** dan **adjacent** didefinisikan dari **adjacent_states.text**.
- Kedua data inilah yang akan dicari hubungan relasinya berdasarkan aturan dan fakta yang terdapat pada data tersebut.

In [4]:
```
# Read the file containing the coastal states
with open(file_coastal, 'r') as f:
    line = f.read()
    coastal_states = line.split(',')
```

- Data negara bagian yang merupakan **pesisir(coastal)** dibaca, untuk mendapatkan data, aturan dan fakta yang ada didalamnya.

In [5]:
```
# Add the info to the fact base
for state in coastal_states:
    fact(coastal, state)
```

- Data **coastal_state** yang telah dibaca tadi, kemudian dimasukan kedalam kumpulan fakta. Menggunakan perulangan untuk mendapatkan nilai dari setiap indexnya.

In [6]:
```
# 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()]
```

- Data negara bagian yang saling **berdekatan(adjacent)** dibaca, untuk mendapatkan data, aturan dan fakta yang ada didalamnya.

In [7]:
```
# 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)
```

- Data **adjlist** yang telah dibaca tadi, kemudian dimasukan kedalam kumpulan fakta. Menggunakan perulangan untuk mendapatkan nilai dari setiap indexnya.

In [8]:
```
# Initialize the variables
x = var()
y = var()
```

- Variabel **x** dan **y** didefinisikan sebagai variabel kosong untuk menampung inputan dan keluaran

Dibawah Ini adalah percobaan yang dilakukan untuk mengukur Logic Programming berdasarkan Fakta yang ada.

Terdapat beberapa inputan pertanyaan yang akan dijawab menggunakan Logic Programming ini. Berikut dilampirkan foto negara bagian di Amerika Serikat untuk mempermudah analisanya.

In [9]:
```
# 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
```

- Dalam percobaan ini dilakukan pembuktian algoritma Logic Programming untuk mententukan **Apakah Nevada berdekatan dengan Louisiana?**
  Jawabannya adalah **tidak**, hal ini dapat dibuktikan di peta.

In [10]:
```
# 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
Idaho
California
Washington
```

In [11]:
```
# 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
```

In [12]:
```
# 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:
Wisconsin
New York
Vermont
Georgia
Washington
Oklahoma
Connecticut
```

In [13]:
```
# 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
```

Berdasarkan keseluruhan percobaan yang telah dilakukan, terbukti bahwa Logic Programming menghasilkan sebuah output berdasarkan fakta dan aturannya. Jika sebuah data saling berelasi karena fakta dan aturan terpenuhi, maka data tersebut diambil dan digunakan sebagai output.