Scikit-learn & a little bit of machine learning

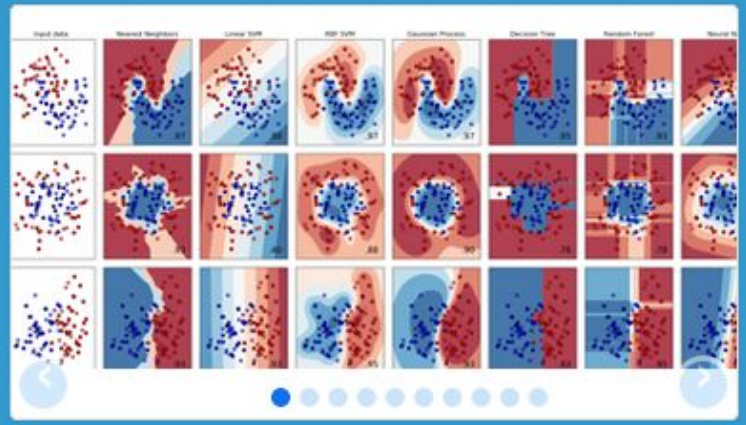
Jakramate Bootkrajang

Outline

- แนะนำโมดูล Scikit-learn
- วิธีการติดตั้ง
- Classification
- Regression
- Clustering
- Performance evaluations

Scikit-learn

- ไลบรารีที่ช่วยให้การทำ data analysis and machine learning ง่ายขึ้น
- Opensource
- Built on Numpy, Scipy and Matplotlib
- Supports
  - Data classification and regression
  - Data clustering
  - Dimensionality reduction
  - Performance evaluation
- http://scikit-learn.org/stable/index.html



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, … — Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, … — Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, … — Examples

Installation

Scikit-learn requires:

- Python (>= 2.7 or >= 3.4),
- NumPy (>= 1.8.2),
- SciPy (>= 0.13.3).

Install via pip

```
pip install -U scikit-learn
```

Data classification

Given a set of (feature, label) pairs find a function that assigns label to unseen data (feature) with high accuracy.

Data can be

- Image
- A set of attributes representing object (ผู้ป่วยใน Haberman's data)
- A time-series (for example, sound)
- etc.

Labels are usually given by experts

Representing data point and labels

Scikit-learn uses 2D-array for data points

```
X = [[30, 62, 5], [33, 70, 12], ….. , [45, 65, 9]]
```

And an array for labels

```
y = [1, 2, …., 1]
```

Example of classification models

- K-nearest neighbour
- Decision Tree
- Neural network

K-nearest neighbour (kNN)

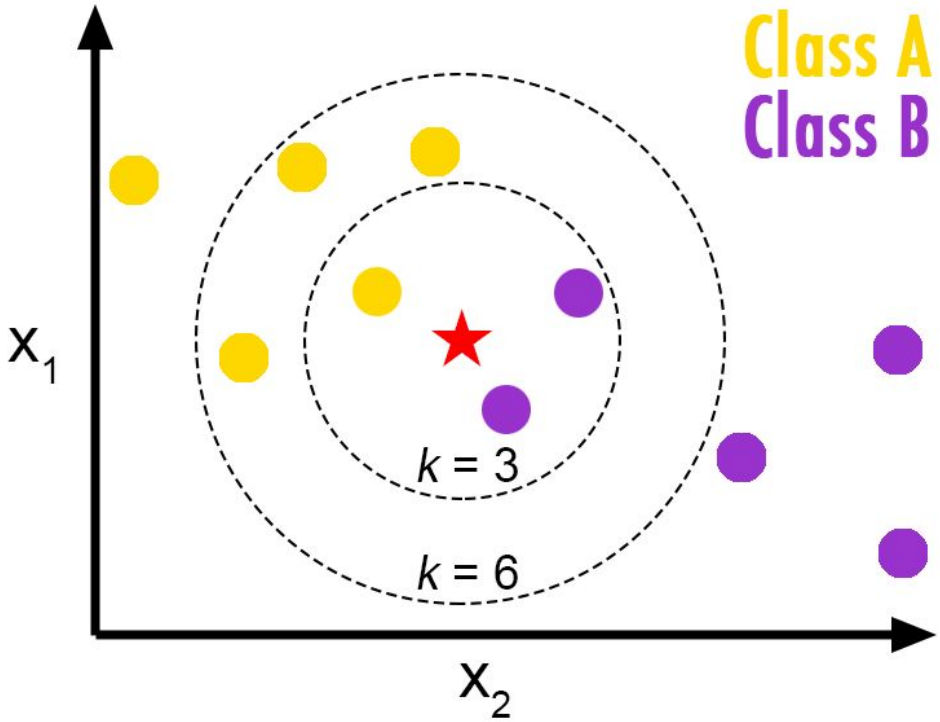
Assumption: Label of nearby data should be the same.

Need to define the `closeness’

Usually `closeness’ is defined as Euclidean distance between two data points.

To predict the label of unseen data, we find its k-closest training data points and take the majority label

Visual example

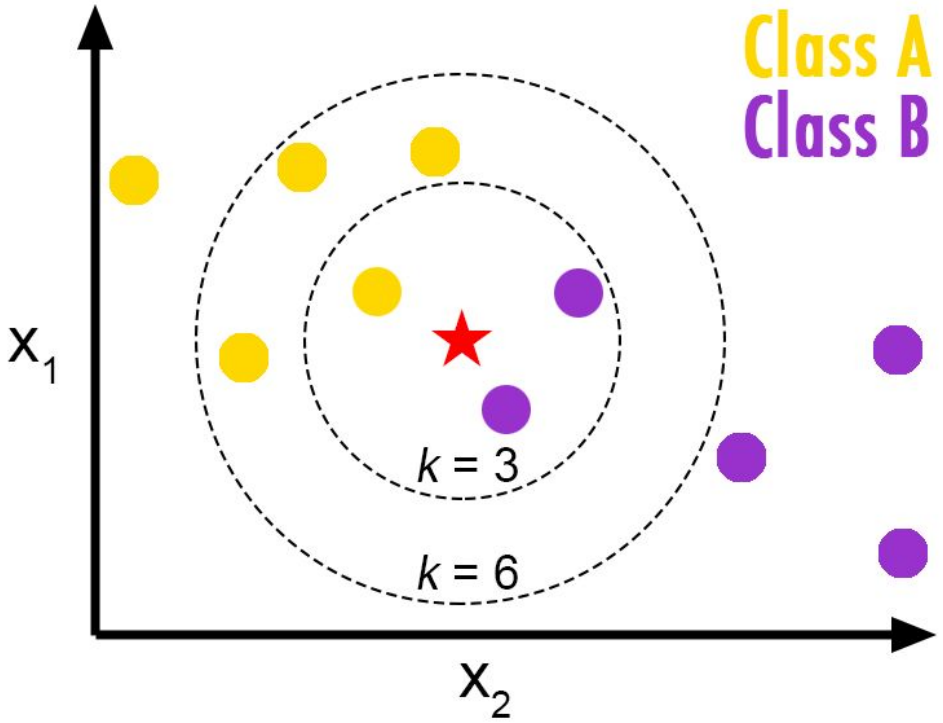


ref: https://helloacm.com/a-short-introduction-to-k-nearest-neighbors-algorithm/

Two types of label voting

Uniform voting

Distance-based voting



kNN in Scikit-learn (1/2)

The function belongs to neighbors sub-module

To construct a classifier model we use

```
model = KNeighborsClassifier(n_neighbors=5, weights='uniform' )
```

To fit the model we call

```
model.fit(data, label)
```

kNN in Scikit-learn (2/2)

To predict a new data point

```
answer = model.predict(data)
```

To obtain the probability of the prediction

```
answer_prob = model.predict_proba(data)
```

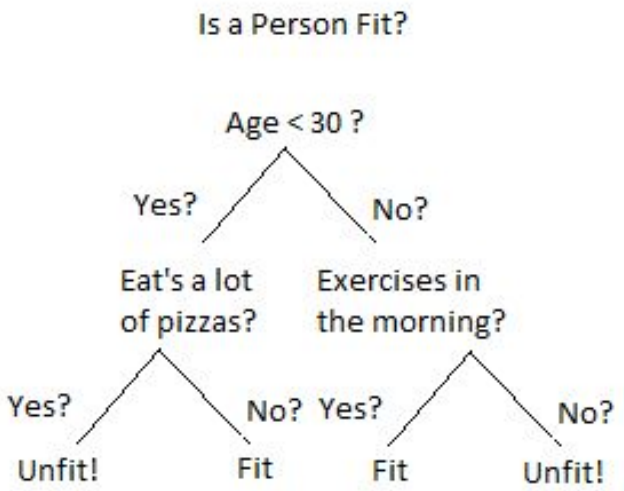
A tiny example

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
>>> from sklearn.neighbors import KNeighborsClassifier
>>> neigh = KNeighborsClassifier(n_neighbors=3)
>>> neigh.fit(X, y)
KNeighborsClassifier(...)
>>> print(neigh.predict([[1.1]]))
[0]
>>> print(neigh.predict_proba([[0.9]]))
[[0.66666667 0.33333333]]
```

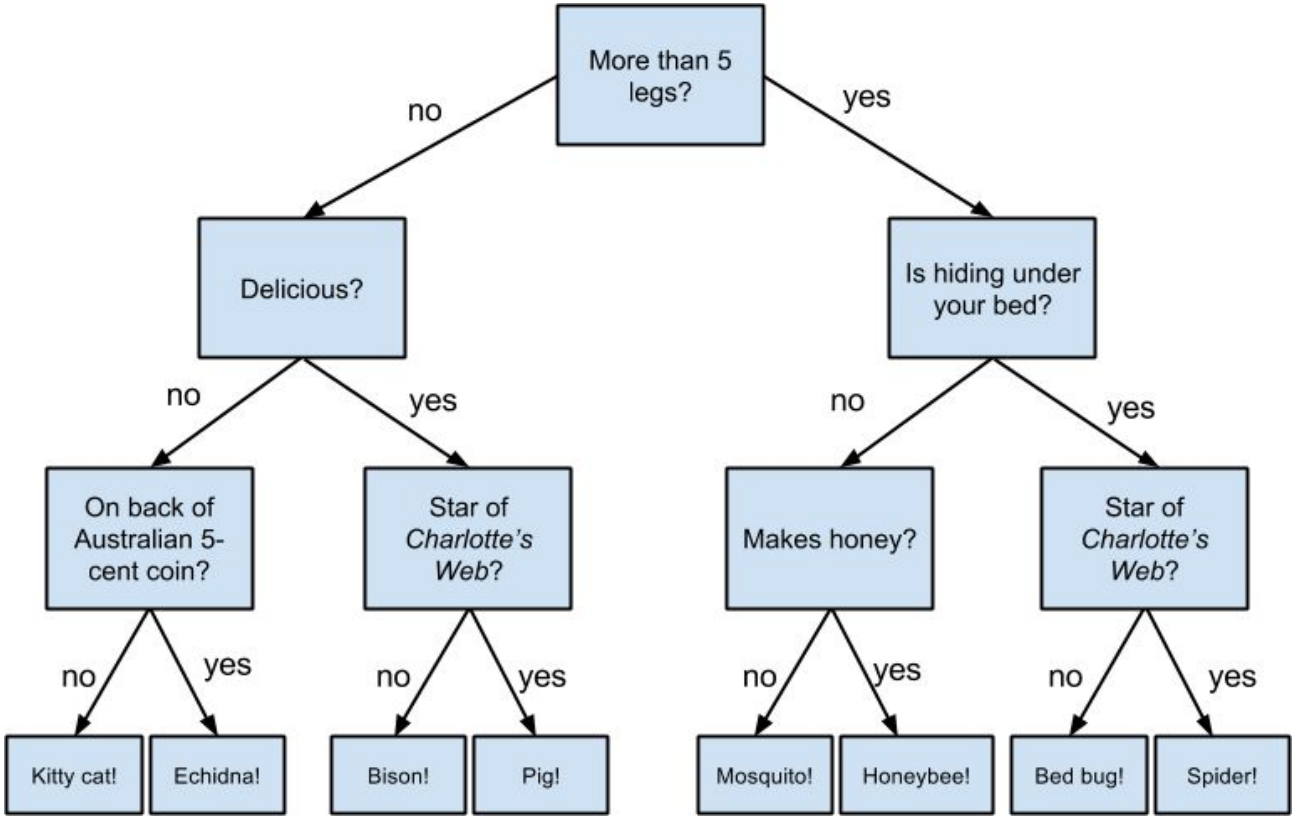
Decision Tree

One of the most natural decision making mechanisms.

Composed of several *internal* decision nodes leading to final prediction



Another example



How tree is built ?

First, build decision node based on an attribute that gives the best split

Then choose an attribute with the second best split

and so on.

Decision Tree in Scikit-learn

Using DecisionTreeClassifier function

DecisionTreeClassifier(*criterion='gini'*, *splitter='best'*, *max_depth=None*, *min_samples_split=2*, *min_samples_leaf=1*, *min_weight_fraction_leaf=0.0*, *max_features=None*, *random_state=None*, *max_leaf_nodes=None*, *min_impurity_decrease=0.0*, *min_impurity_split=None*, *class_weight=None*, *presort=False*)

```
>>> from sklearn import tree
>>> X = [[0, 0], [1, 1]]
>>> Y = [0, 1]
>>> clf = tree.DecisionTreeClassifier()
>>> clf = clf.fit(X, Y)
```

Tree visualisation

It is possible to draw a picture of the resulting decision tree using Graphviz module

Installation

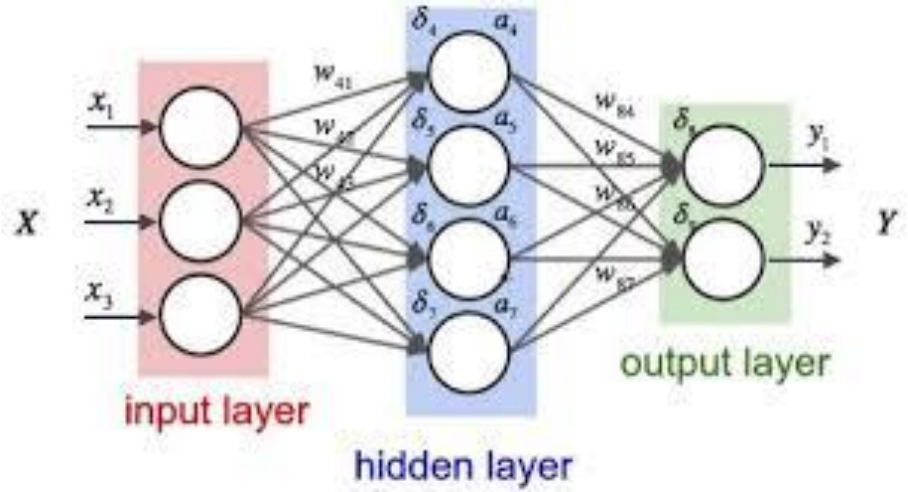
```
pip install graphviz
```

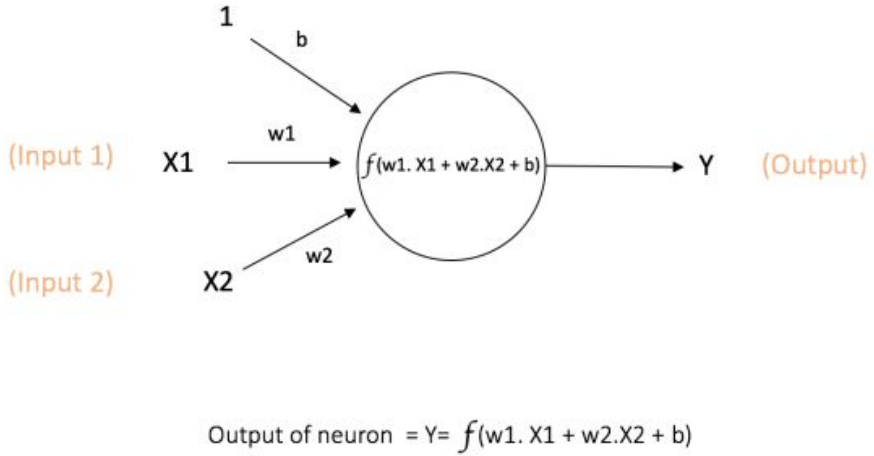
Usage

```
import graphviz
dot_data = tree.export_graphviz(clf, out_file=None)
graph = graphviz.Source(dot_data)
graph.render("iris")
```

Neural network

Inspired by the working of neurons in the brain





To train a network is to learn the best weights for the task.

Neural network in Scikit-learn

MLPClassifier (with too many optional parameters)

MLPClassifier(*hidden_layer_sizes=(100,)*, *activation='relu'*, *solver='adam'*, *alpha=0.0001*, *batch_size='auto'*, *learning_rate='constant'*, *learning_rate_init=0.001*, *power_t=0.5*, *max_iter=200*, *shuffle=True*, *random_state=None*, *tol=0.0001*, *verbose=False*, *warm_start=False*, *momentum=0.9*, *nesterovs_momentum=True*, *early_stopping=False*, *validation_fraction=0.1*, *beta_1=0.9*, *beta_2=0.999*, *epsilon=1e-08*, *n_iter_no_change=10*)

Important parameter is hidden_layer_sizes

This specifies the number of hidden layers as well as the number of nodes in each layer

Example

hidden_layer_sizes = (10,10) # two hidden layers with 10 nodes each

hidden_layer_sizes = (100,1) # one hidden layer with 100 nodes

Note: the more layers MLP has the more complex the model will be

Now for the code

```
>>> from sklearn.neural_network import MLPClassifier
>>> X = [[0., 0.], [1., 1.]]
>>> y = [0, 1]
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
...                     hidden_layer_sizes=(5, 2), random_state=1)
...
>>> clf.fit(X, y)
```

prediction

```
>>> clf.predict([[2., 2.], [-1., -2.]])
array([1, 0])
```

Toy dataset

Let’s compare the performance of the three classifiers on Haberman’s survival data

However, before training we need to perform some preprocessing

```
>>> from sklearn import preprocessing
>>> import numpy as np
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
>>> X_scaled = preprocessing.scale(X_train)

>>> X_scaled
array([[ 0.  ..., -1.22...,  1.33...],
       [ 1.22...,  0.  ..., -0.26...],
       [-1.22...,  1.22..., -1.06...]])
```

Data preprocessing

- Data cleaning
- Filling missing values
- Data normalisation
  - Normalise each attributes to have zero mean and unit variance
- Normalised data improves the learning of most models

Performance evaluation

accuracy_score() function compute correct prediction ratio

```
>>> import numpy as np
>>> from sklearn.metrics import accuracy_score
>>> y_pred = [0, 2, 1, 3]
>>> y_true = [0, 1, 2, 3]
>>> accuracy_score(y_true, y_pred)
0.5
```

Accuracy on training data: could be optimistic.

Accuracy on test data: this is what we want (test model's ability to generalise)

Regression

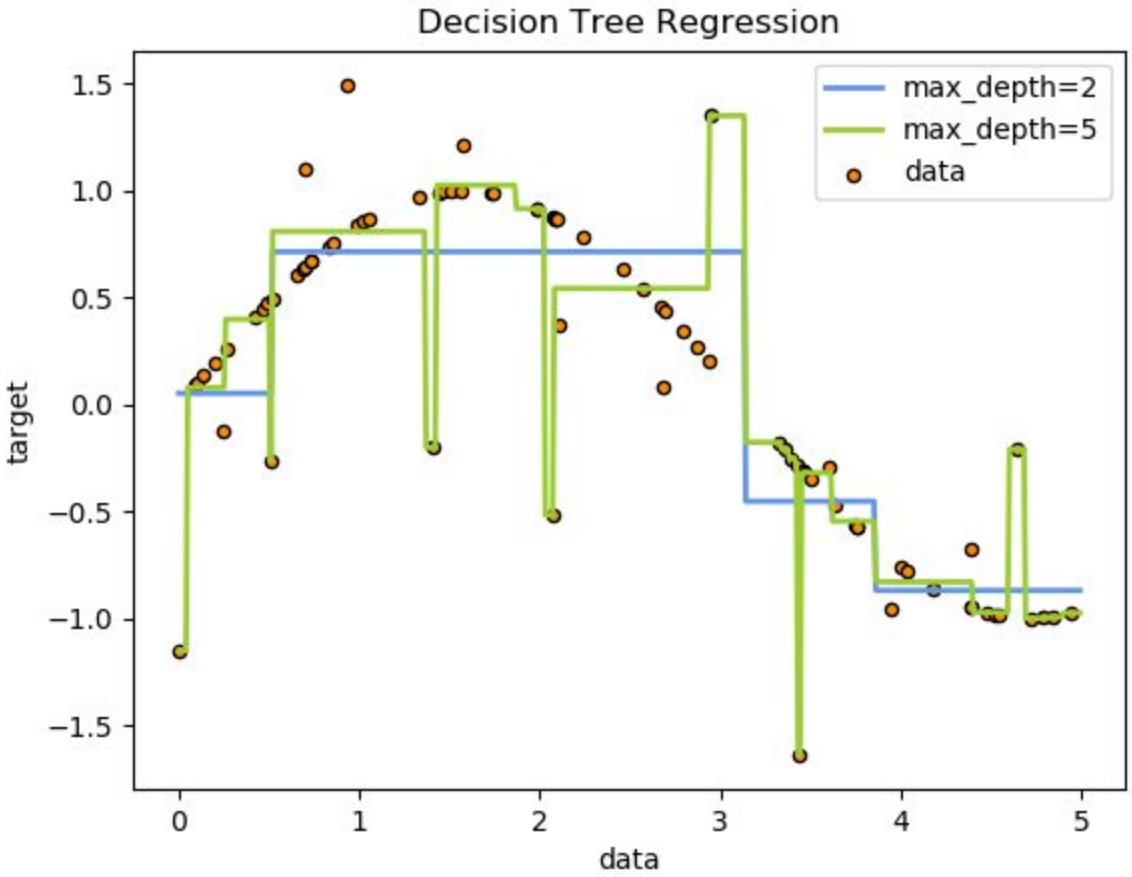
Given a set of (feature, value) pairs find a function that assigns value to unseen data (feature) with high accuracy.

Data can be

- A set of attributes representing some object (car, house)
- the value can be value of car or house

Decision Tree Regressor

```
>>> from sklearn import tree
>>> X = [[0, 0], [2, 2]]
>>> y = [0.5, 2.5]
>>> clf = tree.DecisionTreeRegressor()
>>> clf = clf.fit(X, y)
>>> clf.predict([[1, 1]])
array([0.5])
```


Decision Tree Regression
max_depth=2
max_depth=5
data
target
data
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
0
1
2
3
4
5

kNN for regression

label assigned to a query point is computed based on the mean of the labels of its nearest neighbors.

```
from sklearn import neighbors
```

```
knn = neighbors.KNeighborsRegressor(n_neighbors, weights=weights)
y_ = knn.fit(X, y).predict(T)
```

Here, T is a new data point that needs to be predicted.

Neural network for regression

There is a function called MLPRegressor

```
MLPRegressor(hidden_layer_sizes=(100, ), activation='relu', solver='adam',
alpha=0.0001, batch_size='auto', learning_rate='constant', learning_rate_init=0.001,
power_t=0.5, max_iter=200, shuffle=True, random_state=None, tol=0.0001,
verbose=False, warm_start=False, momentum=0.9, nesterovs_momentum=True,
early_stopping=False, validation_fraction=0.1, beta_1=0.9, beta_2=0.999, epsilon=1e-08,
n_iter_no_change=10)
```

The usage is similar to MLPClassifier

```
mlp = MLPRegressor(algorithm='sgd', max_iter=100, activation='relu',
                   random_state=1, learning_rate_init=0.01,
                   batch_size=X.shape[0], momentum=momentum)
mlp.fit(X,y)
mlp.predict(newX)
```

Toy data

Boston dataset

- Concerns housing values in suburbs of Boston.
- Attribute Information:
    1. CRIM per capita crime rate by town
    2. ZN proportion of residential land zoned for lots over 25,000 sq.ft.
    3. INDUS proportion of non-retail business acres per town
    4. CHAS Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    5. NOX nitric oxides concentration (parts per 10 million)

Boston data (cont.)

6. RM average number of rooms per dwelling
7. AGE proportion of owner-occupied units built prior to 1940
8. DIS weighted distances to five Boston employment centres
9. RAD index of accessibility to radial highways
10. TAX full-value property-tax rate per $10,000
11. PTRATIO pupil-teacher ratio by town
12. B 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
13. LSTAT % lower status of the population
14. MEDV Median value of owner-occupied homes in $1000's (target)

Boston task

Download the data from

https://archive.ics.uci.edu/ml/machine-learning-databases/housing/

The task is regression task for estimating new house in Boston area.

This can be applied for estimating values of second hand car.

Performance evaluation

Usually the performance is measured by Mean Square Errors (MSE)

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2.$$

Or in scikit-learn

`metrics.explained_variance_score` (y_true, y_pred)	Explained variance regression score function
`metrics.mean_absolute_error` (y_true, y_pred)	Mean absolute error regression loss
`metrics.mean_squared_error` (y_true, y_pred[, …])	Mean squared error regression loss
`metrics.mean_squared_log_error` (y_true, y_pred)	Mean squared logarithmic error regression loss
`metrics.median_absolute_error` (y_true, y_pred)	Median absolute error regression loss
`metrics.r2_score` (y_true, y_pred[, …])	R^2 (coefficient of determination) regression score function.

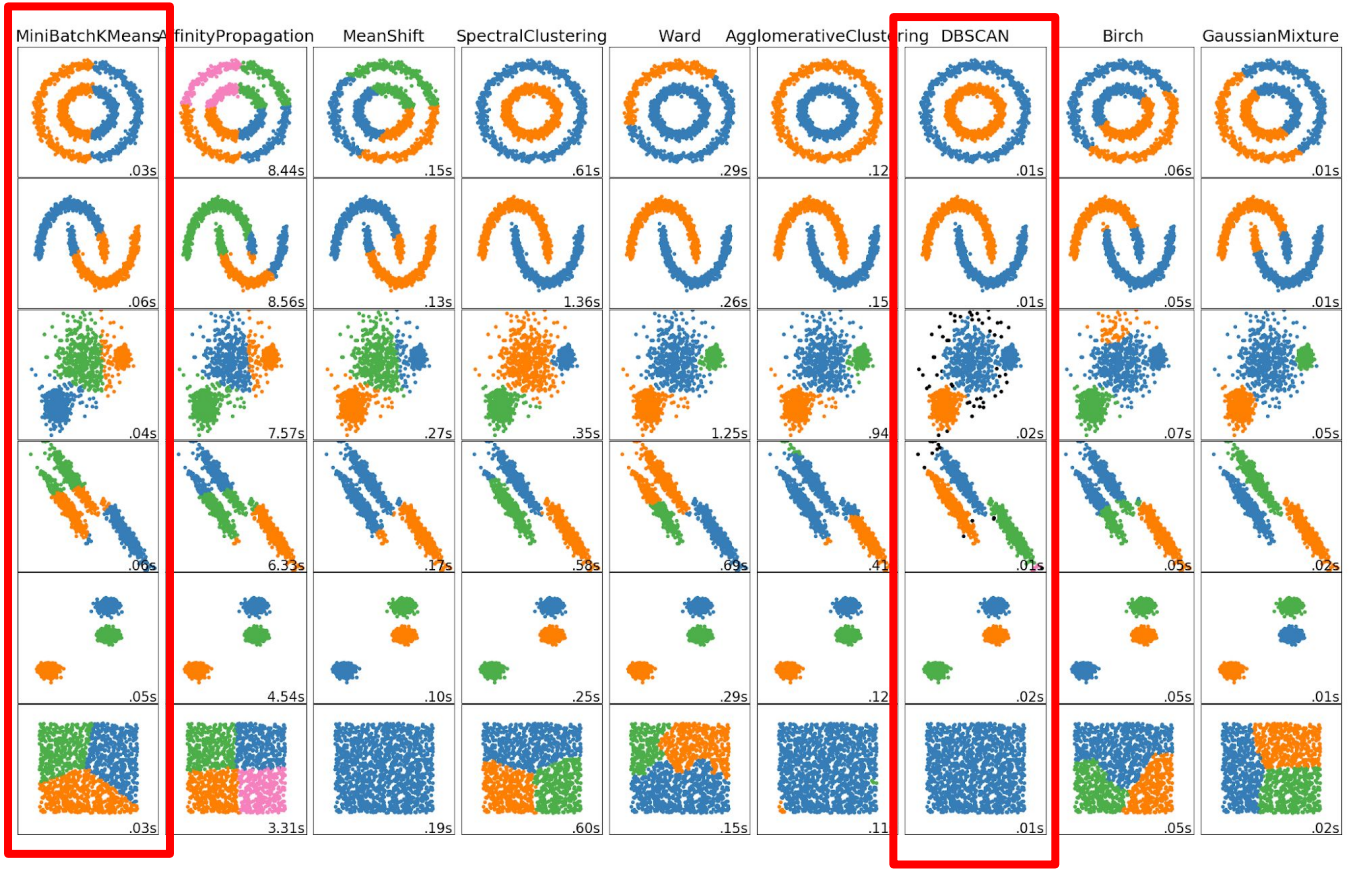
Data clustering

Given a set of data without labels, find natural grouping of the data

Use cases

- We have no idea about the data.
- See some characteristic before reaching out for help from experts.
- Usually the first analysis task before performing classification.

Data clustering algorithms

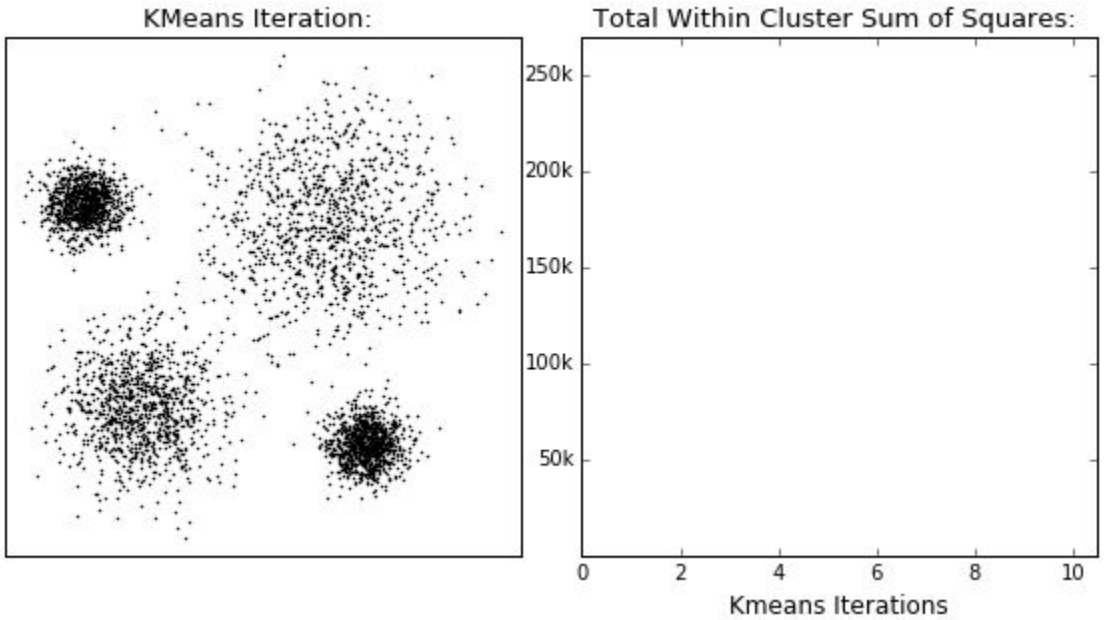


K-mean algorithm

Input: k (the number of clusters),
D (a set of lift ratios)
Output: a set of k clusters
Method:
Arbitrarily choose k objects from D as the initial cluster centers;
Repeat:
1. (re)assign each object to the cluster to which the object is the most similar, based on the mean value of the objects in the cluster;
2. Update the cluster means, i.e., calculate the mean value of the objects for each cluster

Until no change;

Some visualisation

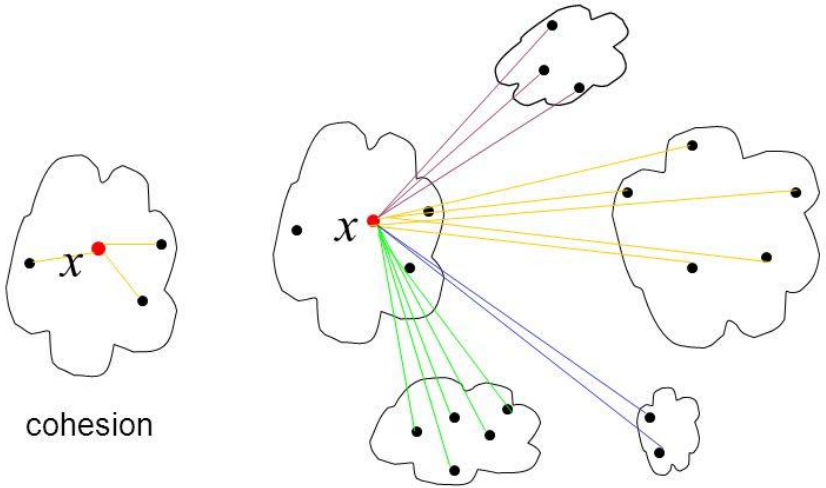


K-means in Scikit-learn

```
1  from sklearn.cluster import KMeans
2  from pylab import *
3
4  X = concatenate((randn(20,2), randn(20,2)+2, randn(20,2)-2))
5
6  model = KMeans(n_clusters=3)
7  model.fit(X)
8
9  scatter(X[model.labels_ == 0,0], X[model.labels_ == 0,1], color='k')
10 scatter(X[model.labels_ == 1,0], X[model.labels_ == 1,1], color='b')
11 scatter(X[model.labels_ == 2,0], X[model.labels_ == 2,1], color='g')
12 show()
```

Performance evaluation

Silhouette coefficient



$a(x)$: average distance in the cluster

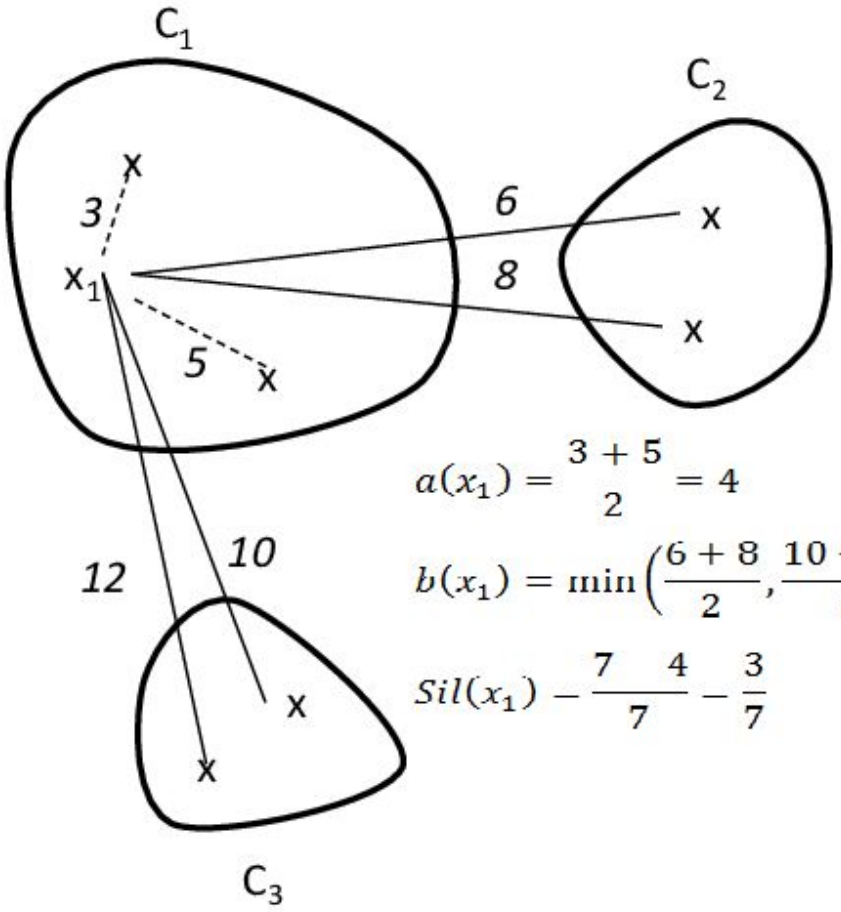
$b(x)$: average distances to others clusters, find minimal

$$s(x) = \frac{b(x) - a(x)}{\max\{a(x), b(x)\}}$$

$$S = \frac{\sum_{i=1}^{n} s(x_i)}{n}$$

Averaging all s(x) to get the Silhouette coefficient of the dataset

Score is between -1 and 1
Best score is 1
Worse score is -1



$$a(x_1) = \frac{3+5}{2} = 4$$

$$b(x_1) = \min\left(\frac{6+8}{2}, \frac{10+12}{2}\right) = 7$$

$$Sil(x_1) - \frac{7}{7} \quad \frac{4}{7} - \frac{3}{7}$$

The code

```
1 from sklearn.cluster import KMeans
2 from sklearn.metrics import silhouette_score
3 from pylab import *
4
5
6 X = concatenate((randn(20,2), randn(20,2)+2, randn(20,2)-2))
7
8 model = KMeans(n_clusters=3)
9 model.fit(X)
10
11 score = silhouette_score(X, model.labels_)
12 print(score)
```