Hands-on Tutorial of Machine Learning in Python

- 投影片: https://goo.gl/xCFSdQ
- 程式: https://goo.gl/oDaAgv
  - 有稍作修改 ☺ 若網路還行，可以考慮重抓一下
  - Github 也有更新, 重新 git clone 或 git pull 更新
- 所需套件跟程式:

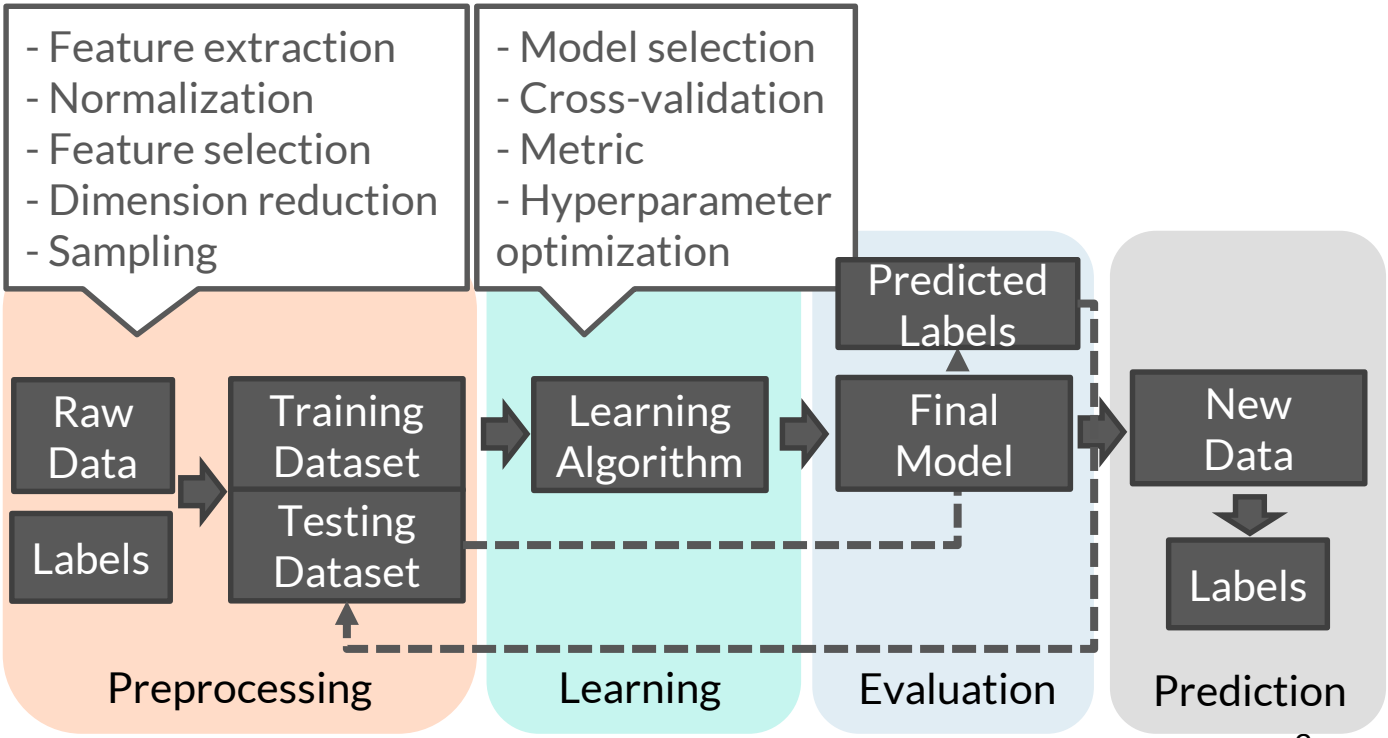
  https://github.com/tw-cmchang/hands-on-ML

Hands-on Tutorial of Machine Learning in Python

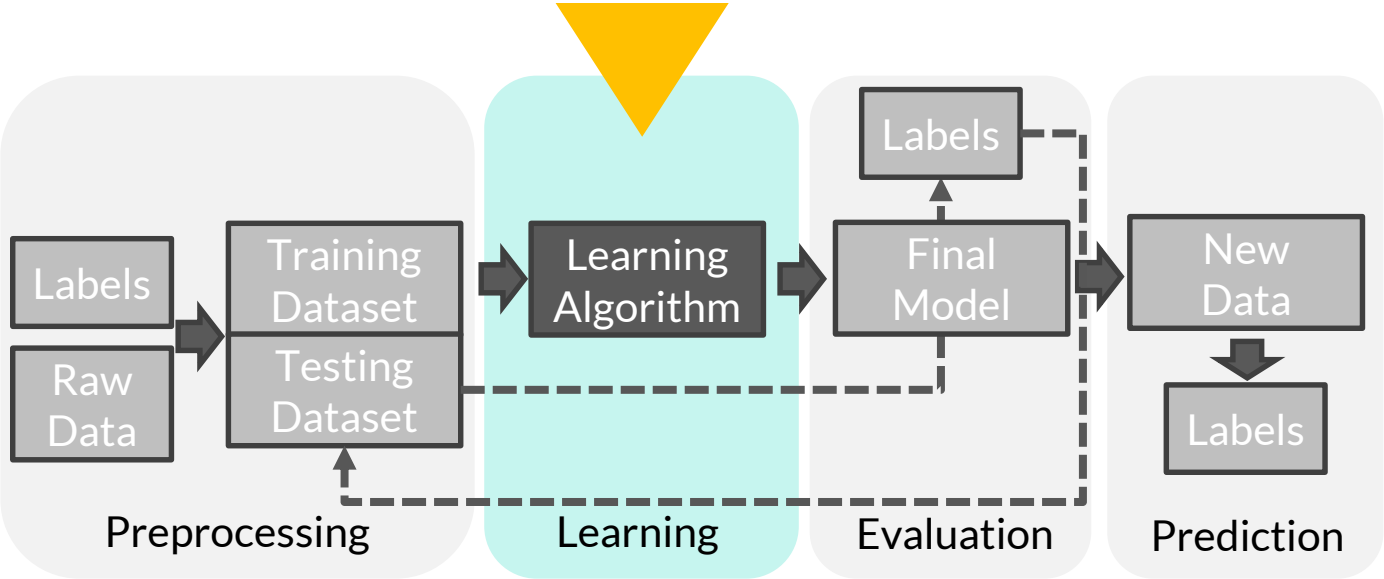
中央研究院資訊科學所資料洞察實驗室 張鈞閔

2017.09.19

Flow Chart of Predictive Modeling



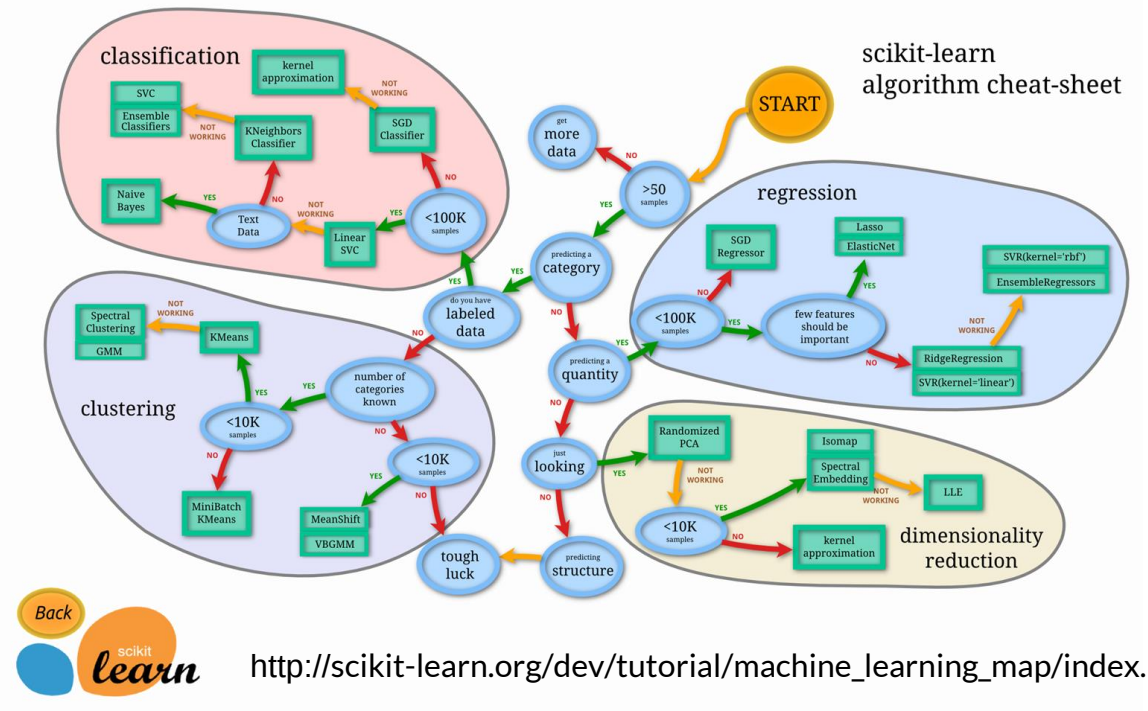
Today we will focus on …



Scikit-learn: a powerful Python library in the field of machine learning

Overview

Scikit-learn: Machine Learning in Python



http://scikit-learn.org/dev/tutorial/machine_learning_map/index.html

The content of this lecture partially comes from

(1) https://github.com/jakevdp/sklearn_tutorial

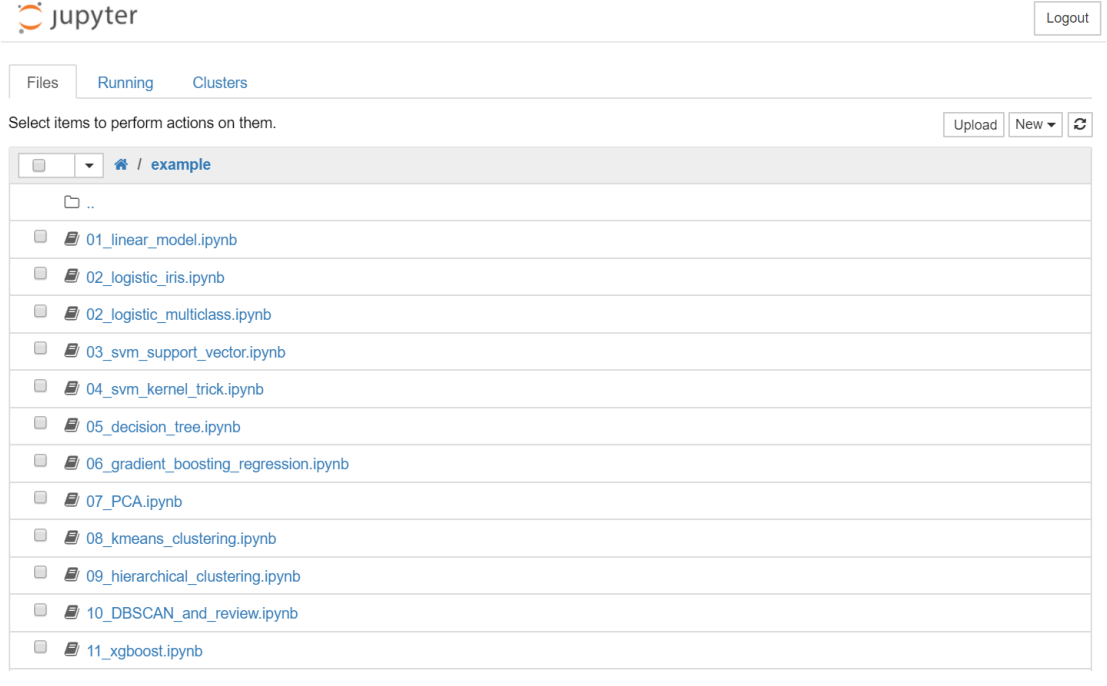
(2) http://scikit-learn.org/stable/documentation.html

(3) https://blog.keras.io/

Let's do it!

Jupyter notebook

- Interactive web application and able to execute codes



Basic Commands

Enter	: edit mode
Esc	: command mode
A	: add a cell **a**bove
B	: add a cell **b**elow
DD	: delete this cell

Shift + **Enter** : run cell and select below

Import Package

```
import numpy as np
```

Package name alias

numpy.sum() ➔ np.sum()

```
from sklearn.linear_model import LinearRegression
```

Package Package/Module Package/Module/Class/Function

- Execute “python –m site” to observe your own site-packages directory

What is Machine Learning?

About building programs with **tuneable parameters** that are adjusted automatically to improve their performance by **adapting to previously seen data**.

Framework

Define a set of functions

↓

Evaluate and Search

↓

Pick the best function

- From a machine learning method, for example (linear regression)

  $$y = w_1 x_1 + w_2 x_2 + \cdots + w_n x_n$$

- **Objective/Cost/Loss** function to evaluate the goodness of current function

  $$y = 1.5 x_1 + 0.5 x_2 + \cdots + 2 x_n$$

- W* to minimize objective function

What is Machine Learning?

About building programs with **tuneable parameters** that are adjusted automatically to improve their performance by **adapting to previously seen data**.

Categorize into 3 classes

- Supervised Learning

  ⇒ Data with **explicit** labels
- Unsupervised Learning

  ⇒ Data **without** labels
- Reinforcement Learning

  ⇒ Data with **implicit** labels

Supervised Learning

- In supervised learning, we have a dataset consisting of both **features** (input variables) and **labels** (output variables)

X1	X2	X3	X4	X5	X6	X7	label
0.5	0.1	-1.5	-9	100	20	0	20
4.7	1.2	0	-7	198	29	1	12

- The task is to construct an **estimator** (model) which enables to predict the labels of an instance given the set of features
- Two categories: **Classification** and **Regression**
  - Classification: the label is discrete
  - Regression: the label is continuous

Training and Testing Dataset

- To validate the generalization of a trained model, splitting a whole dataset into **training** and **testing** datasets
- Based on the training dataset, we fit and optimize predictive models
- Use testing datasets to evaluate the performance of trained models to ensure the model generalization

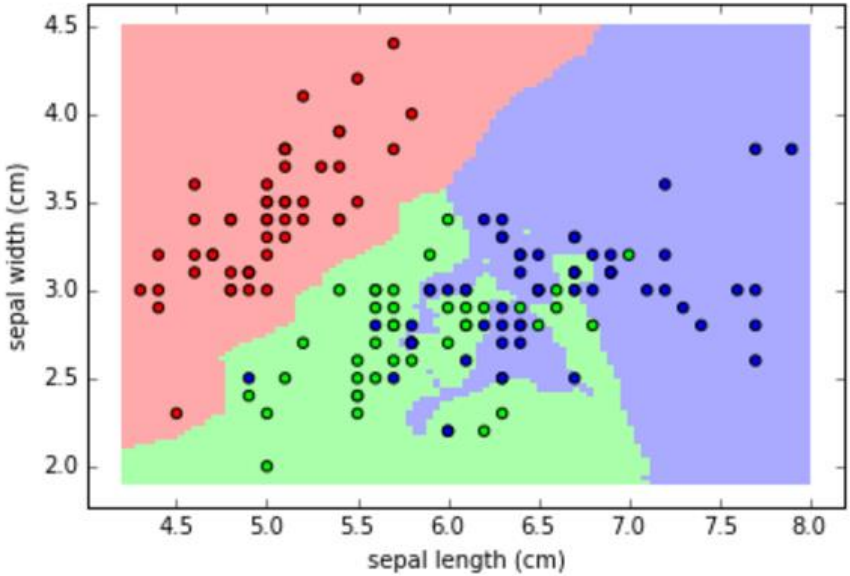
Generalization

- We hope that a model trained by the training dataset can seamlessly apply to unseen dataset

  ➔ Generalization to unseen data
- If the model **overfits** the training dataset, its performance on testing dataset will be worse
- Higher model complexity, easier to overfitting

Classification

- For example, k-Nearest Neighbor for iris classification problem:



Common Metrics of Classification

- Confusion matrix

	Predicted 0	Predicted 1
Actual 0	True Negative (TN)	False Positive (FP)
Actual 1	False Negative (FN)	True Positive (TP)

- Accuracy = $\frac{TN+TP}{ALL= (TN+TP+FN+FP)}$

More Metrics

	Predicted 0	Predicted 1
Actual 0	True Negative (TN)	False Positive (FP)
Actual 1	False Negative (FN)	True Positive (TP)

$$\text{Recall} = \frac{TP}{FN+TP}$$

$$\text{Precision} = \frac{TP}{TP+FP}$$

$$\text{F1 score} = 2\frac{\text{Recall} * \text{Precision}}{\text{Recall}+\text{Precision}}$$

```python
from sklearn import metrics
```

```python
metrics.confusion_matrix(y_test,y_pred)
```

```
array([[ 7,  0,  0],
       [ 0,  8,  4],
       [ 0,  1, 10]])
```

```python
metrics.accuracy_score(y_test,y_pred)
```

```
0.83333333333333337
```

```python
metrics.precision_score(y_test,y_pred,average="weighted")
```

```
0.85079365079365077
```

```python
metrics.recall_score(y_test,y_pred,average="weighted")
```

```
0.83333333333333337
```

```python
metrics.f1_score(y_pred=y_pred,y_true=y_test,average="weighted")
```

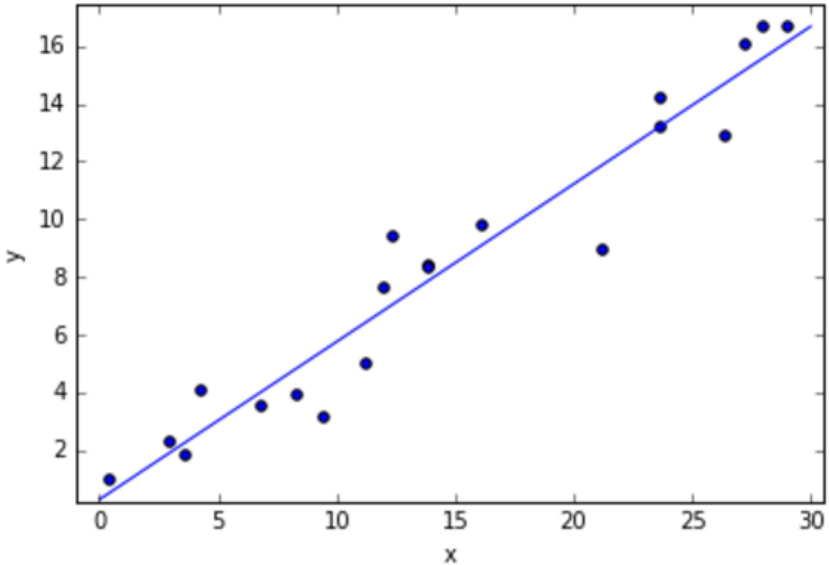
```
0.83142857142857152
```

```python
print(metrics.classification_report(y_test,y_pred))
```

```
             precision    recall  f1-score   support

          0       1.00      1.00      1.00         7
          1       0.89      0.67      0.76        12
          2       0.71      0.91      0.80        11

avg / total       0.85      0.83      0.83        30
```

Regression

- For example, fit a line to the data



Common Metrics in Regression

- Mean absolute error (MAE)
- Mean square error (MSE)
- R squared

```
metrics.r2_score(y_pred=y_pred,y_true=y_test)
```

```
0.82031173595813334
```

```
metrics.mean_absolute_error(y_pred=y_pred,y_true=y_test)
```

```
0.67860317120632041
```

```
metrics.mean_squared_error(y_pred=y_pred,y_true=y_test)
```

```
0.62402155001086912
```

Unsupervised Learning

- The data has no labels but we are interested in

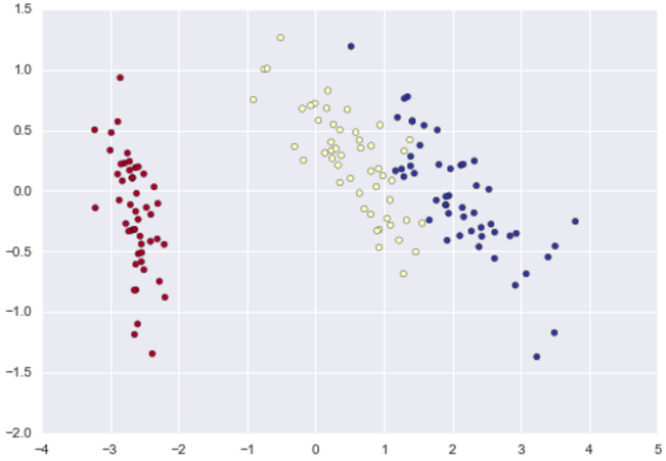
  (1) describing hidden structure from instances

  (2) finding similarity among instances
- Unsupervised learning comprises tasks such as

  dimensionality reduction and **clustering**

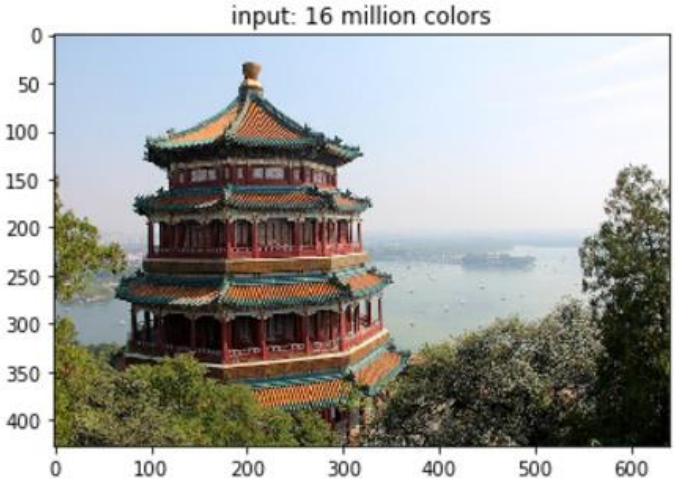
Dimensionality Reduction

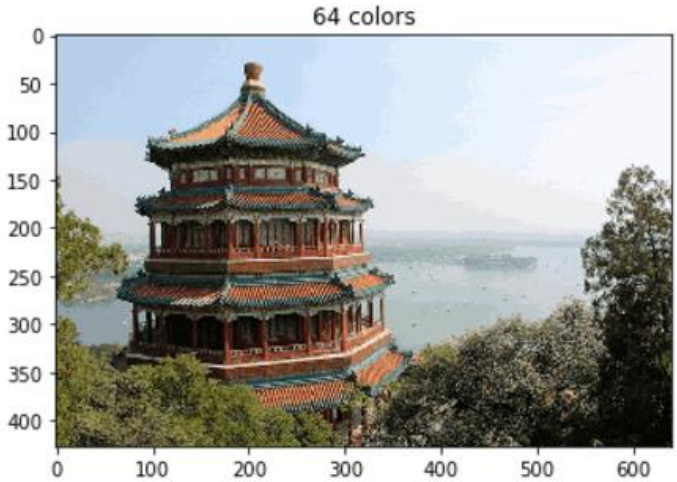
- For example, conduct PCA to visualize the iris dataset in two dimensions



Clustering

- For example, K-means clustering for data compression

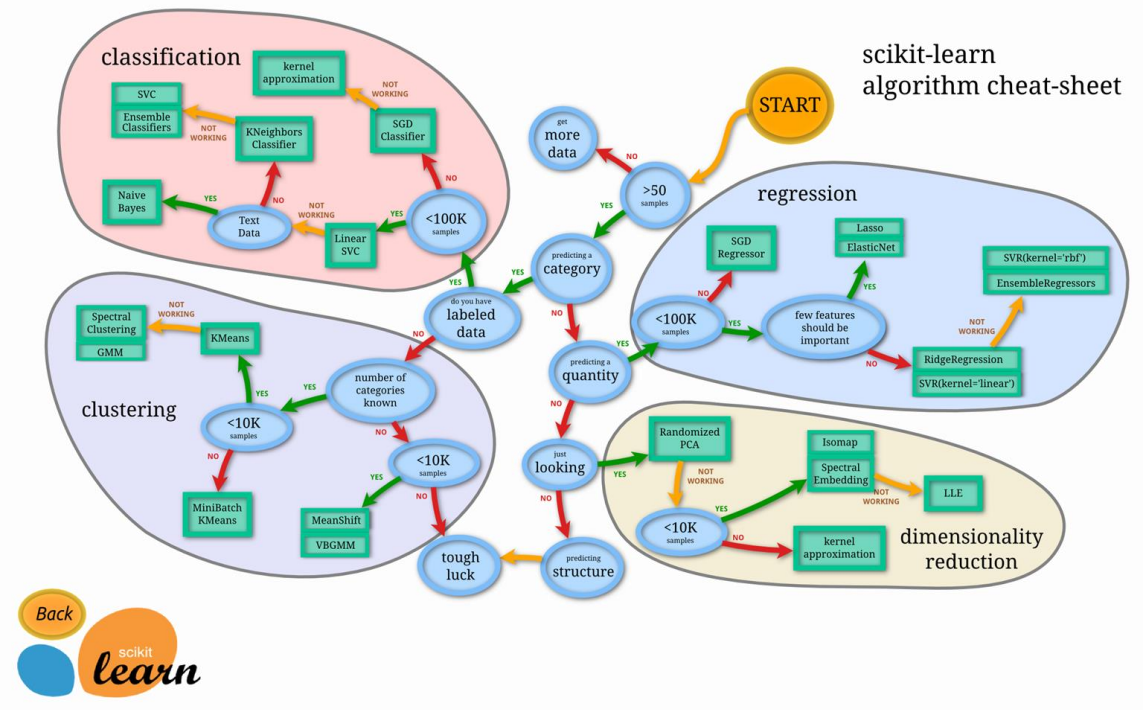




Use scikit-learn to apply ML algorithms

: condition

: method



Scikit-learn's Estimator Interface

All estimators

- **model.fit()** : fit training data, X: data, Y: label

  Supervised: model.fit(X , Y) ⇔ Unsupervised: model.fit(X)

Supervised estimator

- **model.predict():**

  predict the **label** of a set of data
- **Model.predict_proba():**

  return **probability** of each classes (some estimators)

Unsupervised estimator

- **model.transform():**

  transform new data into new basis by model
- **model.fit_transform():**

  fit and then transform

Supervised Learning In-depth

Do have the labels ☺

Model-based learning

- Linear regression
- Regression with Regularization
- Logistic regression
- Support vector machine
- Decision Tree
- Random Forests
- XGBoost

Instance-based learning

- Naive Bayesian model
- K-nearest neighbor (KNN)

Unsupervised Learning In-depth

Have no idea about labels ☹

Dimension reduction

- Principal component analysis

Clustering

- K-means clustering
- Hierarchical clustering
- DBSCAN

進行流程

1. 介紹主題
2. 說明如何使用 Python 完成
3. 範例說明 (example/)
4. 動手練習 (exercise/)

Supervised Learning In-depth

Do have the labels ☺

Model-based learning

- Linear regression
- Regression with Regularization
- Logistic regression
- Support vector machine
- Decision Tree
- Random Forests
- XGBoost

Instance-based learning

- Naive Bayesian model
- K-nearest neighbor (KNN)

Supervised Learning In-depth

Do have the labels ☺

Model-based learning

- **Linear regression**
- **Regression with regularization**
- Logistic regression
- Support vector machine
- Decision Tree
- Random Forests
- XGBoost

Instance-based learning

- Naive Bayesian model
- K-nearest neighbor (KNN)

Linear Regression

- Based on the assumption that a **linear** relationship exists between input (x_i) and output variables (y)

$$y = a_1 x_1 + a_2 x_2 + \cdots + a_n x_n$$

- Linear models are still powerful because input variables can be arbitrarily transformed
  - For example, polynomial regression

$$x_1 = v, x_2 = v^2, \ldots, and\ x_n = v^n$$

Example

example/01_linear_model.ipynb

- Here we start from a toy example: fitting a sine curve with additional noise

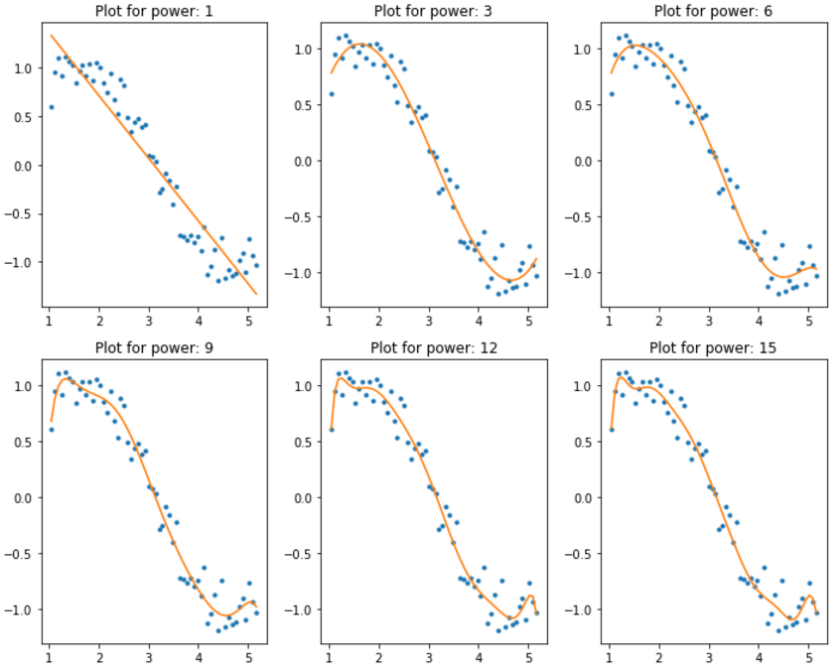
```
x = np.array([i*np.pi/180 for i in range(60,300,4)])
np.random.seed(100)
y = np.sin(x) + np.random.normal(0,0.15,len(x))
```

- Our goal is to estimate this function using polynomial regression with powers of x form 1 to 15

Linear Model in Python

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge
def example_regression(data,power,plot_dict,reg_type,alpha=0):
    # define estimator object
    type_dict = {'Linear' : LinearRegression(normalize=True),
                 'Lasso' : Lasso(alpha = alpha, normalize=True),
                 'Ridge' : Ridge(alpha = alpha, normalize=True)}

    # generate X of different powers
    X = ['x']
    if power >= 2:
        X.extend(['x_%d' %i for i in range(2,power+1)])

    # fit the model
    if reg_type in type_dict:
        model = type_dict[reg_type]
    model.fit(data[X],data['y'])
    y_pred = model.predict(data[X])
```

Results (linear regression)



As the model complexity increases, the models tends to fit even smaller deviations in the training data set.

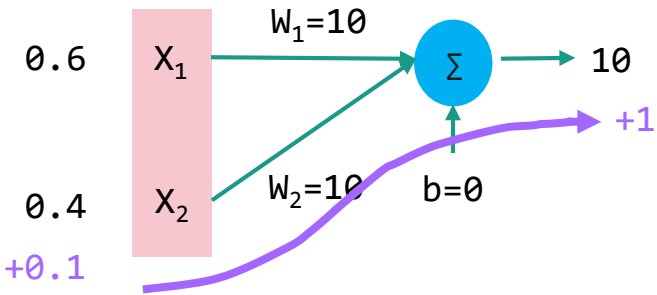
When Model Complexity Increases

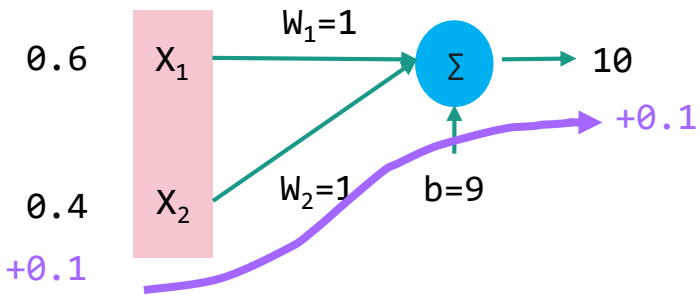
	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10
pow_1	3.7	2	-0.65	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_2	3.7	1.9	-0.54	-0.017	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_3	1.1	-1.4	3.4	-1.4	0.15	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_4	1.1	-1.1	2.9	-1.1	0.087	0.0051	NaN	NaN	NaN	NaN	NaN	NaN
pow_5	1	0.7	-0.86	1.8	-0.97	0.18	-0.012	NaN	NaN	NaN	NaN	NaN
pow_6	1	-6.1	16	-15	7.4	-2	0.28	-0.016	NaN	NaN	NaN	NaN
pow_7	0.98	-19	54	-61	36	-13	2.5	-0.26	0.011	NaN	NaN	NaN
pow_8	0.94	-66	2.1e+02	-2.9e+02	2.1e+02	-93	26	-4.2	0.39	-0.015	NaN	NaN
pow_9	0.94	-70	2.3e+02	-3.1e+02	2.4e+02	-1.1e+02	31	-5.5	0.57	-0.03	0.00054	NaN
pow_10	0.88	-4.6e+02	1.9e+03	-3.4e+03	3.5e+03	-2.3e+03	9.9e+02	-2.9e+02	57	-7.3	0.53	-0.017
pow_11	0.88	-5.4e+02	2.3e+03	-4.2e+03	4.4e+03	-3e+03	1.4e+03	-4.3e+02	93	-13	1.2	-0.062
pow_12	0.88	-9.9e+02	4.6e+03	-9.4e+03	1.1e+04	-9.2e+03	5.1e+03	-2.1e+03	5.9e+02	-1.2e+02	18	-1.7
pow_13	0.88	-1.4e+03	6.8e+03	-1.5e+04	2e+04	-1.7e+04	1.1e+04	-4.8e+03	1.6e+03	-3.9e+02	70	-8.9
pow_14	0.87	2.5e+03	-1.7e+04	4.9e+04	-8.3e+04	9.5e+04	-7.6e+04	4.4e+04	-1.9e+04	6.1e+03	-1.5e+03	2.6e+02
pow_15	0.87	1.8e+03	-1.2e+04	3.5e+04	-5.9e+04	6.6e+04	-5.1e+04	2.8e+04	-1.1e+04	3.3e+03	-6.9e+02	94

Magnitude of coefficients also increase

Generalization

- If coefficient magnitude is large, a small input deviation would lead to large output deviation
- For example





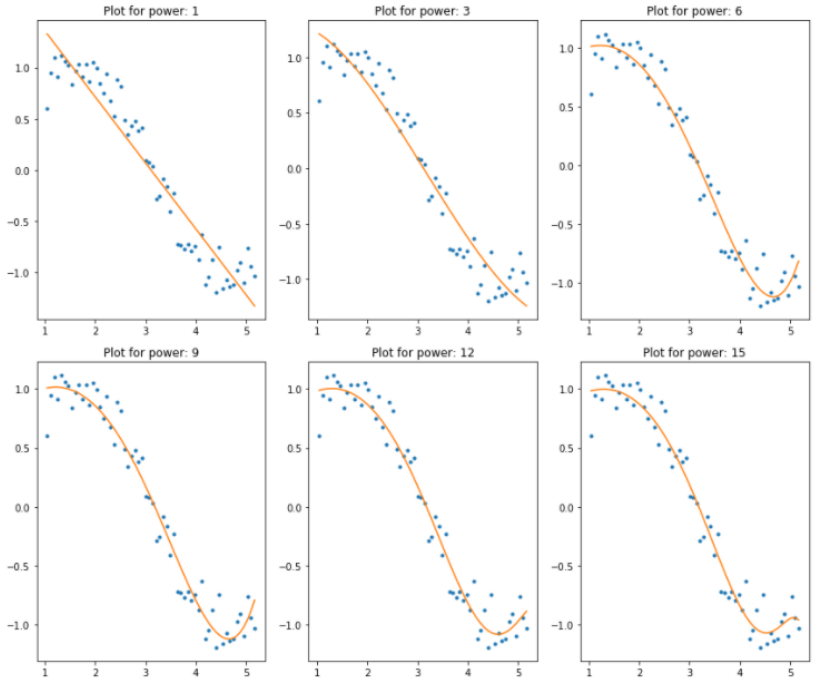
For better generalization, we usually add weight regularization

Ridge Regression

- Perform L2-regularization, i.e. add penalty to the square of the magnitude term into the cost function

$$\text{Cost} = \text{Prediction error} + \alpha \sum (\text{weights})^2$$

Result (Ridge Regression)



Effectively reduce the model complexity

Weight Regularization

	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10
pow_1	3.7	2	-0.65	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_2	3.7	1.8	-0.53	-0.019	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_3	2.7	1.3	0.24	-0.31	0.032	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_4	1.6	0.71	0.68	-0.28	-0.03	0.0091	NaN	NaN	NaN	NaN	NaN	NaN
pow_5	1.3	0.61	0.63	-0.19	-0.034	-0.00049	0.0015	NaN	NaN	NaN	NaN	NaN
pow_6	1.3	0.68	0.5	-0.15	-0.028	-0.002	0.0004	0.00019	NaN	NaN	NaN	NaN
pow_7	1.3	0.73	0.44	-0.14	-0.025	-0.0018	0.00022	0.0001	1.8e-05	NaN	NaN	NaN
pow_8	1.3	0.74	0.43	-0.14	-0.024	-0.0018	0.00022	9.9e-05	1.7e-05	3.3e-07	NaN	NaN
pow_9	1.3	0.72	0.45	-0.14	-0.025	-0.0[illegible]	0.00019	0.0001	2.1e-05	2e-06	-4.2e-07	NaN
pow_10	1.2	0.69	0.47	-0.13	-0.026	-0.0[illegible]4	0.00012	0.0001	2.4e-05	3.6e-06	1.3e-07	-1.4e-07
pow_11	1.2	0.67	0.48	-0.13	-0.026	-0.0[illegible]7	3.1e-05	8.6e-05	2.4e-05	4.4e-06	5e-07	-2.1e-08
pow_12	1.2	0.66	0.47	-0.12	-0.026	-0.0[illegible]9	-4.6e-05	7e-05	2.2e-05	4.5e-06	6.9e-07	5.5e-08
pow_13	1.1	0.67	0.46	-0.12	-0.025	-0.0[illegible]9	-0.0001	5.5e-05	2e-05	4.3e-06	7.4e-07	9.4e-08
pow_14	1.1	0.68	0.44	-0.11	-0.024	-0.0029	-0.00014	4.4e-05	1.7e-05	4e-06	7.3e-07	1.1e-07
pow_15	1.1	0.69	0.43	-0.11	-0.024	-0.0029	-0.00015	3.6e-05	1.5e-05	3.6e-06	6.9e-07	1.1e-07

Magnitude of coefficients do not increase significantly

Lasso Regression

- Perform **L1**-regularization, i.e. add penalty to the absolute value of the magnitude term into the cost function

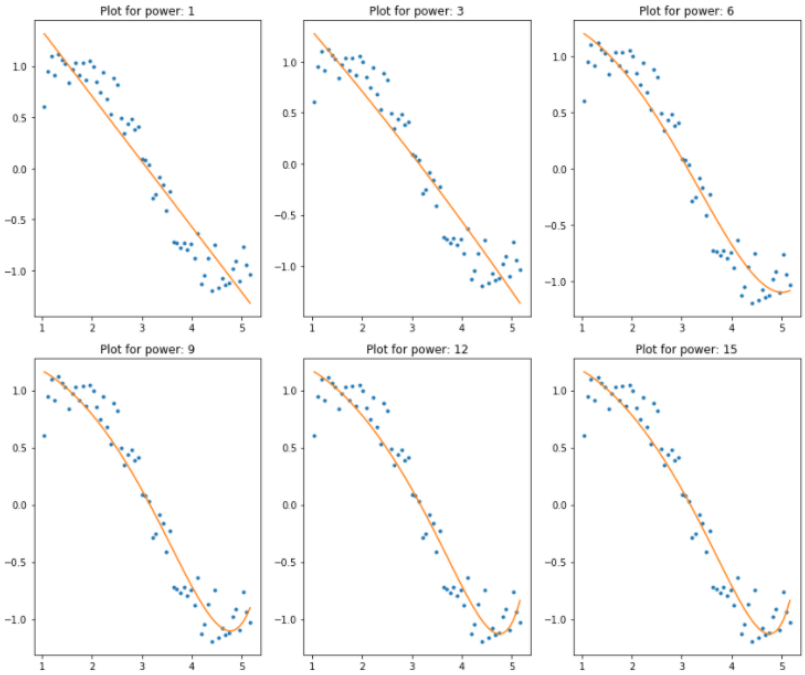
$$\text{Cost} = \text{Prediction error} + \boxed{\alpha \sum |\text{weights}|}$$

- LASSO stands for

  Least **Absolute** Shrinkage and **Selection** Operator

Result (Lasso Regression)

α = 0.001



Effectively reduce the model complexity

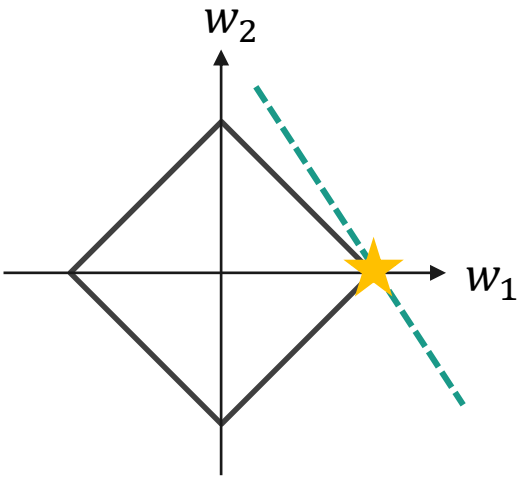
Sparsity ➔ Feature Selection

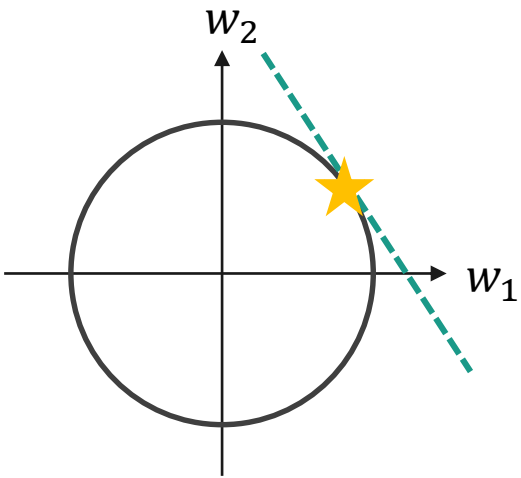
	rss	intercept	coef_x_1	coef_x_2	coef_x_3	coef_x_4	coef_x_5	coef_x_6	coef_x_7	coef_x_8	coef_x_9	coef_x_10
pow_1	3.7	2	-0.64	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_2	3.7	1.9	-0.54	-0.016	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_3	3.7	1.9	-0.54	-0.016	-0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
pow_4	3.1	1.5	-0.2	-0.11	-0	0.0015	NaN	NaN	NaN	NaN	NaN	NaN
pow_5	2.4	1.4	-0	-0.15	-0	0	0.00042	NaN	NaN	NaN	NaN	NaN
pow_6	2.2	1.4	-0	-0.15	-0	-0	0	7.8e-05	NaN	NaN	NaN	NaN
pow_7	2	1.3	-0	-0.13	-0.0025	-0	-0	0	1.6e-05	NaN	NaN	NaN
pow_8	1.9	1.3	-0	-0.12	-0.0043	-0	-0	0	0	3.1e-06	NaN	NaN
pow_9	1.8	1.3	-0	-0.12	-0.0044	-0	-0	0	0	0	5.9e-07	NaN
pow_10	1.9	1.3	-0	-0.12	-0.0025	-0	-0	0	0	0	0	1.1e-07
pow_11	1.9	1.3	-0	-0.13	-0.00044	-0	-0	0	0	0	0	0
pow_12	1.9	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0
pow_13	1.9	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0
pow_14	1.9	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0
pow_15	1.9	1.3	-0	-0.13	-0	-0	-0	0	0	0	0	0

High sparsity

Why Lasso Leads to Sparsity

- L1 $|w_1| + |w_2|$
- L2 $w_1{}^2 + w_2{}^2$





When L1 regularization is adopted,
it is more likely to converge at the corner

The Effect of α

- Change the magnitude of α to see what's going on
- For Lasso regression, larger α leads to higher sparsity

Exercise (15 mins)

exercise/ex01_linear_model.ipynb

- Boston house price prediction
- Hints
  - Consider nonlinear transformations of current features
  - Regression with regularization

Short Summary

- Increasing model complexity causes the weight explosion, and this may result in bad generalization
- Regularization helps limit the growth of model complexity
  - Lasso: add L1 regularization, especially beneficial to select features due to the property of sparsity
  - Ridge: add L2 regularization

Supervised Learning In-depth

Do have the labels ☺

Model-based learning

- Linear regression
- Regression with Regularization
- **Logistic regression**
- **Support vector machine**
- Decision Tree
- Random Forests
- XGBoost

Instance-based learning

- Naive Bayesian model
- K-nearest neighbor (KNN)

Logistic Regression (LR)

- Logistic regression is a linear model for **classification**
- **NOT** require a linear relationship between the input and output variables
- Can handle all sorts of relationships because there is **non-linear** log transformation applied to the prediction
- Output range = (0,1)

Mathematical Formulation of LR

- For a pair of input (x) and output (y), assume that the probability model

$$P(y|x) = \frac{1}{1 + e^{-yw^T x}}$$

- LR finds w by maximizing the likelihood

$$\max_{w} \prod_{i=1}^{k} P(y_i|x_i)\,,$$

$$i = 1, \dots, k \ (k \ training \ instances)$$

Formulation in Scikit-Learn

- Take log on likelihood

$$\max_{w} \prod_{i=1}^{k} P(y_i|x_i) = \max_{w} \sum_{i=1}^{k} \log(P(y_i|x_i))$$

$$= \max_{w} \sum_{i=1}^{k} -log(1 + e^{-y_i w^T x_i})$$

$$= \min_{w} \sum_{i=1}^{k} log(1 + e^{-y_i w^T x_i})$$

- Can add L1 or L2 regularization

Regularized Logistic Regression

- For example, L2 penalized logistic regression

$$\min_{w} \frac{1}{2} w^T w + \boldsymbol{C} \sum_{i=1}^{k} log(1 + e^{-y_i w^T x_i})$$

- Unlike the previous linear models, here adjust the weight of error term by $\boldsymbol{C}$
- Solvers: liblinear, lbfgs, newton-cg, sag

Solver Selection

Case	Solver
Small dataset or L1 penalty	liblinear
Multinomial or large dataset	lbfgs, sag or newton-cg
Very Large dataset	sag

- Only liblinear solver support L1 penalization

Logistic Regression in Python

```
from sklearn.linear_model import
LogisticRegression

logreg = LogiticRegression(
        # {newton-cg, lbfgs, liblinear, sag}
        solver = ‘liblinear’,
        # ‘l1’ only for liblinear solver
        penalty = ‘l2’,
        # {‘ovr’, ‘multinomial’}
        multi_class = ‘ovr’,
        # smaller values, stronger regularization
        C = 1.0
)
```

Example

example/ 02_logistic_multiclass.ipynb

- Two options to address multiclass problem: multinomial and One-vs-Rest

example/02_logistic_iris.ipynb

- Change the value of C to observe the results

Exercise (15 mins)

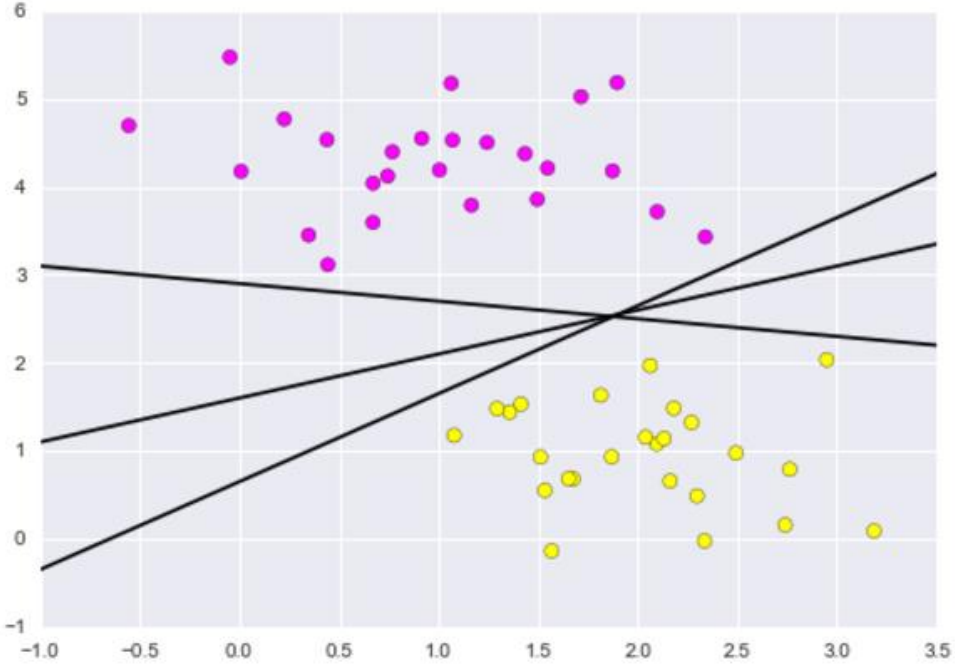
exercise/ex02_logistic_regression.ipynb

- Classification of handwritten digits

Support Vector Machine (SVM)

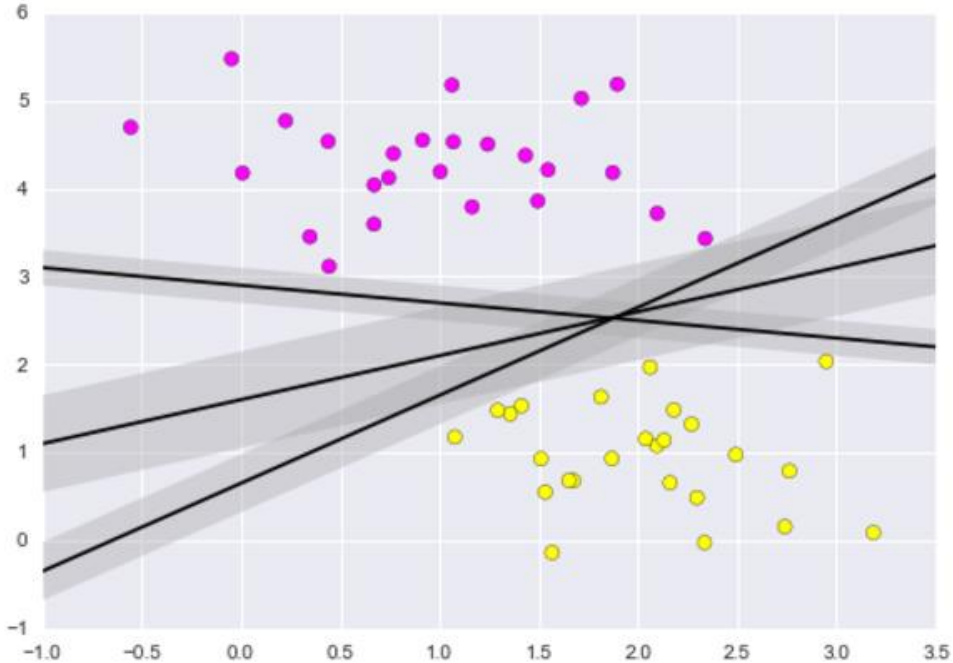
- A powerful method for both **classification** and **regression**
- Construct a **hyper-plane** or set of hyper-planes in a high dimensional space
- The hyper-plane has the **largest distance** (margin) to the nearest training data points of any classes
- Different **Kernel functions** can be specified for the decision function

An Example in 2D



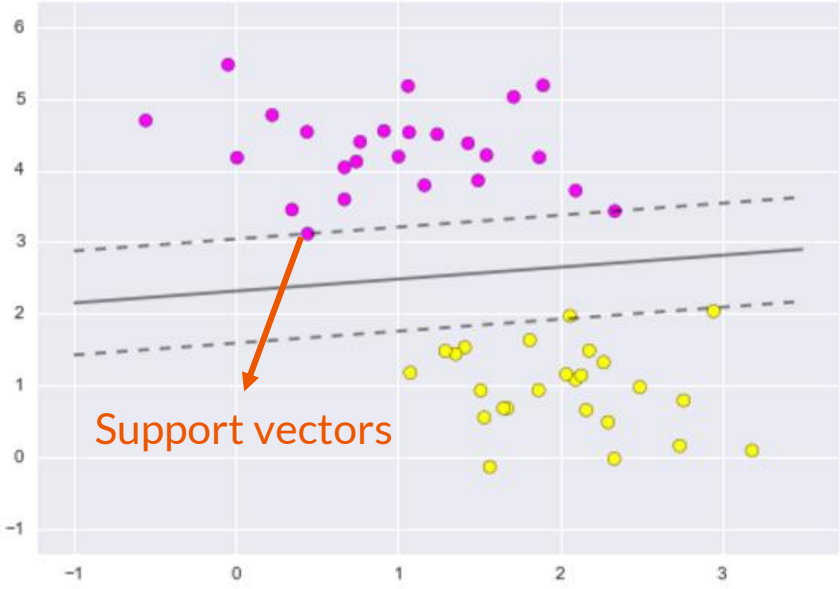
Which one line better separates the data points ?

Intuition of SVM: Maximal Margin



The middle fit clearly has the largest margin (perpendicular distance between data points)

Mathematical Formulation of SVM



$$w^T x + b = \begin{bmatrix} +1 \\ 0 \\ -1 \end{bmatrix}$$

$$\begin{cases} w^T x_i + b \geq 1 \text{ , if } y_i = 1 \\ w^T x_i + b \leq -1 \text{ , if } y_i = -1 \end{cases}$$

$$\Downarrow$$

$$y_i(w^T x_i + b) \geq 1$$

The decision function, $f(x) = sign(w^T x + b)$

Mathematical Formulation of SVM

- Distance between $w^T x + b = 1$ and $w^T x + b = -1$

$$D = \frac{2}{||w||} = \frac{2}{\sqrt{w^T w}}$$

- Maximizing D is equivalent to minimizing $w^T w$
- The **primal** problem of the SVM

$$\min_{w,b} \quad \frac{1}{2} w^T w$$
$$\text{subject to } \; y_i(w^T x_i + b) \geq 1,$$
$$i = 1, \dots, k$$

Not Always Linearly Separable

- Tolerate training error but try to minimize it

$$\min_{w,b} \quad \frac{1}{2} w^T w + C \sum_{i=1}^{k} \boldsymbol{\varepsilon_i}$$

$$\text{subject to} \quad y_i(w^T x_i - b) \geq 1 - \boldsymbol{\varepsilon_i},$$
$$\varepsilon_i \geq 0,$$
$$i = 1, \dots, k$$

- Project to a high dimensional feature space

$$x \rightarrow \phi(x)$$

but $\phi(x)$ may be very complex

From The Dual Problem of SVM

- Use Lagrange relexation to derive the dual problem of SVM (skip the details here, please check [1])
- The dual problem of SVM

Kernel function, $\boldsymbol{k(x_i, x_j)}$

$$\text{maximize } L_D = \sum_{i=1}^{k} \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \boxed{\phi(x_i)^T \phi(x_j)}$$

$$\text{subject to } \sum_{i=1}^{k} \alpha_i y_i = 0, \qquad \forall \alpha_i \geq 0$$

maybe easier than $\phi(x_i)$

[1] http://www.cmlab.csie.ntu.edu.tw/~cyy/learning/tutorials/SVM2.pdf

Radial Basis Function (RBF)

- The RBF kernel on two feature vectors x_i and x_j, is defined as

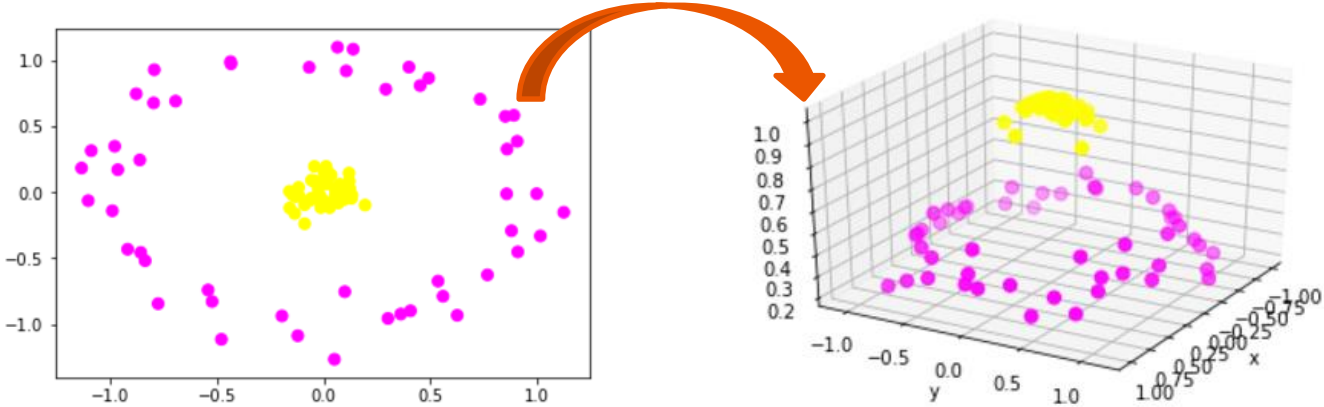
$$k(x_i, x_j) = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

- The feature space of kernel has an **infinite** number of dimensions

Example

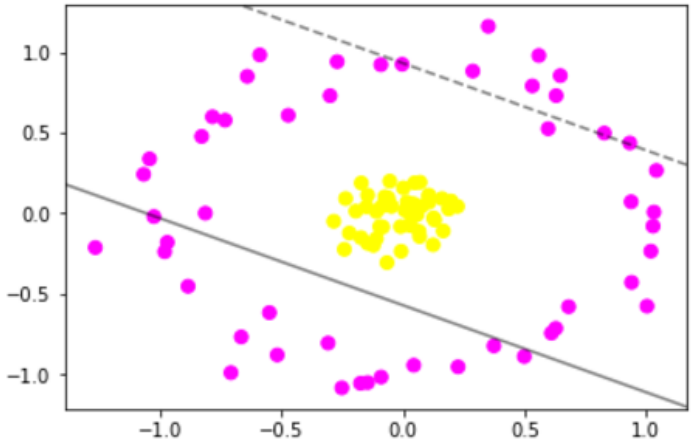
- Transform by the radial basis function

$\phi(x)$



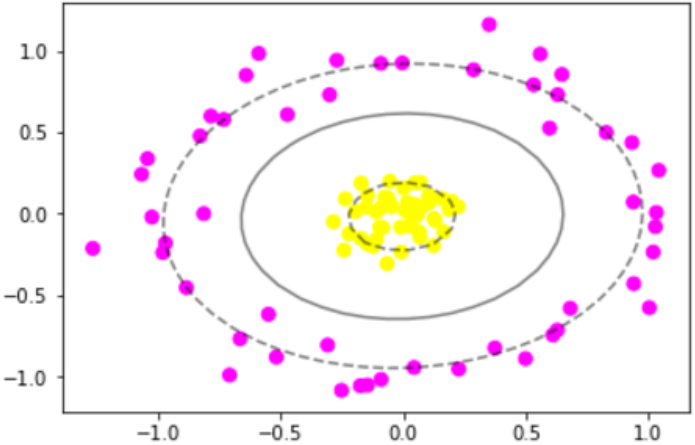
x $(x, \phi(x))$

Linear kernel versus RBF kernel



Linear kernel

RBF kernel



SVM in Python

```
from sklearn import svm
clf = svm.SVC(
        # {linear, poly, rbf}
        kernel = ‘linear’,
        # used only if kernel=‘poly’
        degree = 3,
        # enable predict_prob()
        probability = True,
        # smaller values, stronger regularization
        C = 1.0
)
```

Example - Support Vector

example/04_svm_support_vector.ipynb

- Support vectors, in actual, is a subset of training points in the decision function
- Obtain class probability by using probability = True

Example - Kernel Tricks

example/04_svm_kernel_trick.ipynb

- ‘linear’: linear kernel
- ‘poly’: polynomial kernel function (controlled by the parameter **degree**)
- ‘rbf’: radial basis function (default setting)

  ➔ Project to an **infinite** feature space

Exercise (15 mins)

exercise/ex03_svm_iris.ipynb

- Try to visualize the **decision functions**
- Try to circle out the **support vectors**
- Try to compare different **kernel** functions

- Apply SVM on the handwritten digits dataset in ex02

Supervised Learning In-depth

Do have the labels ☺

Model-based learning

- Linear regression
- Regression with Regularization
- Logistic regression
- Support vector machine
- **Decision Tree**
- **Random Forests**
- **XGBoost**

Instance-based learning

- Naive Bayesian model
- K-nearest neighbor (KNN)

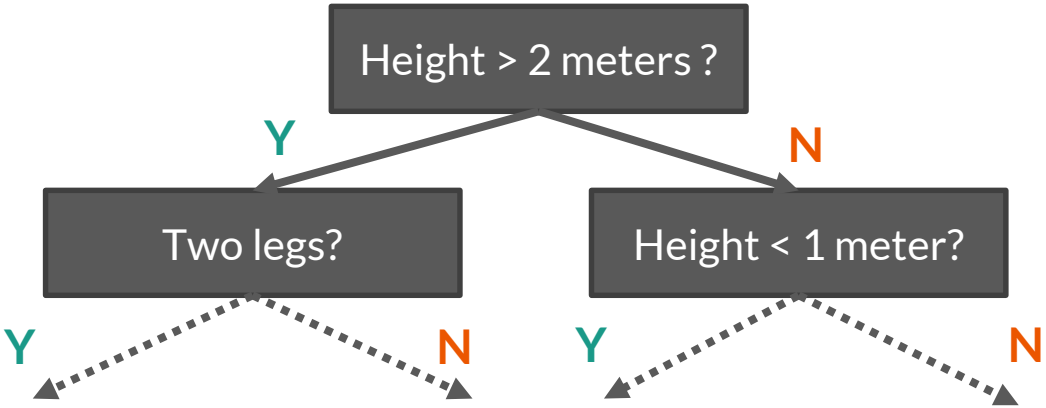
Decision Tree

- Used for both **classification** and **regression**
- Decided by a series of splits, and a split implies a **binary** cut on one dimension
- Each split should maximize the **information gain**
- Decision tree use **impurity measure** instead
  - Maximizing information gain

    = minimizing impurity measure

Decision Tree

- Simple to interpret and visualize



- No need to data normalization

Splitting Criteria

- Assumed that Class 1, 2, …, C and current node t

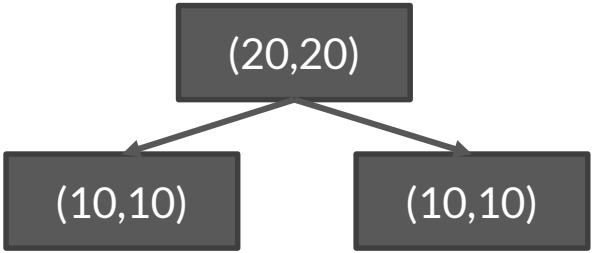
$$P(i|t) = \frac{\#\ of\ \boldsymbol{class\ i}\ samples\ at\ node\ t}{\#\ of\ total\ samples\ at\ node\ t}$$

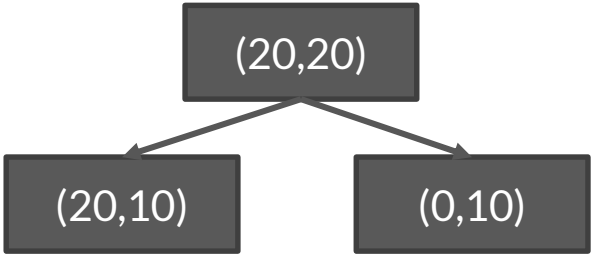
Gini impurity	Entropy
$I_G(t) = \sum_{i=1}^{C} P(i\|t)(1 - P(i\|t))$	$I_E(t) = -\sum_{i=1}^{C} P(i\|t) log(P(i\|t))$

- For a split (binary cut) to maximize the gain

  Gain = I(parent) – I(Left child) – I (right child)

For example (Gini)





$$I_P = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = 0.5$$

$$I_L = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = 0.5$$

$$I_R = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = 0.5$$

$$Gain = \frac{1}{2} - \frac{1}{2} \times \frac{1}{2} - \frac{1}{2} \times \frac{1}{2} = 0$$

$$I_P = \frac{1}{2} \times \frac{1}{2} + \frac{1}{2} \times \frac{1}{2} = \frac{1}{2}$$

$$I_L = \frac{2}{3} \times \frac{1}{3} + \frac{1}{3} \times \frac{2}{3} = \frac{4}{9}$$

$$I_R = 0 + 1 = 1$$

$$Gain = \frac{1}{2} - \frac{3}{4} \times \frac{4}{9} - \frac{1}{4} = \frac{1}{6}$$

Feature Importance

- The importance of a feature is computed as the **total reduction of the criteria** brought by that feature
- Sum of importance of all features is equal to 1
- In practice, we often use feature importance provided by a tree model to rank and select features

Decision Tree in Python

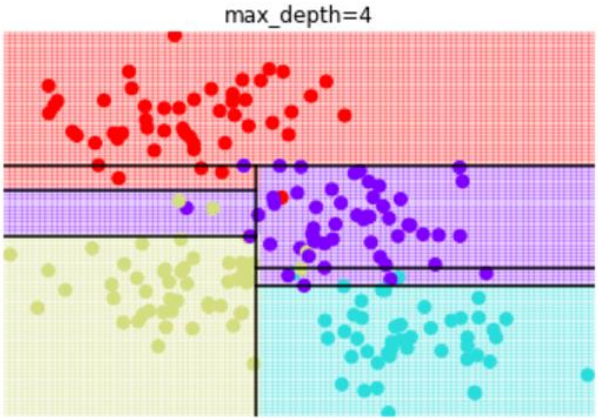
```
from sklearn.tree import DecisionTreeClassifier

clf = DecisionTreeClassifier(
        criterion = ‘gini’,
        max_depth = None,
        min_samples_split = 2,
        min_samples_leaf = 1,
)

# feature importance
clf.feature_importances_
```

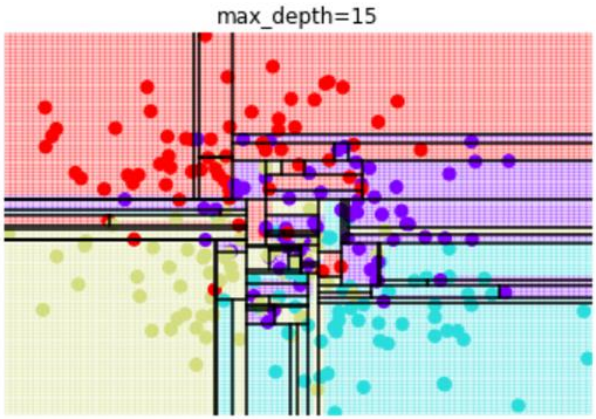
Example

example/05_decision_tree.ipynb

- criteria = {'gini', 'entropy'}
- max_depth = the maximum depth of a tree
  - If None, nodes are expanded until all leaves are pure or until all leaves contain less than **min_samples_split** samples
- min_sample_split: the minimum number of samples required to split
- min_sample_leaf: the minimum number of samples required to be at a leaf node

The effect of max_depth





- Deeper tree, more prone to overfitting
- Overfitting ➔ Complex decision boundaries

  ➔ Hard to generalize to unseen datasets

Tree Ensemble

- Aggregate multiple weak learners to a strong learner
- This strong learner should have better **generalization** and prevent overfitting

- Voting mechanism (classification) or taking average of all prediction values (regression)

Ensemble Method - Bagging

- Sample a subset of the training dataset to train a weak learner
- Each learner is **independent** and this bagging process can be parallelized
- Random Forests

Random Forest in Python

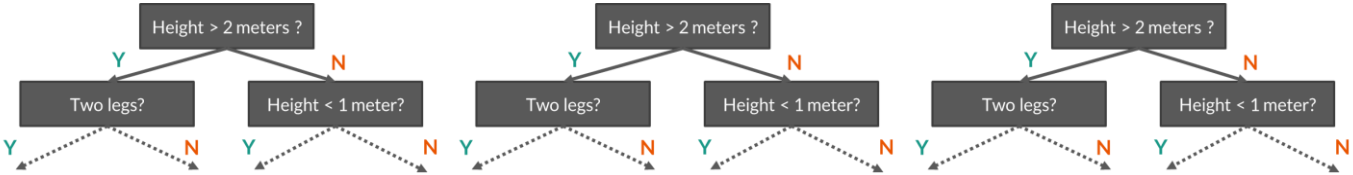
```
from sklearn.ensemble import
RandomForestClassifier

clf = RandomForestClassifier(
        # number of trees in this forest
        n_estimator = 100,
        criterion = ‘gini’,
        max_depth = 10,
        min_samples_split = 2,
        min_samples_leaf = 1,
)
```

```
# feature importance
clf.feature_importances_
```

Ensemble Method - Boosting

- Instead of sampling the input features, **sample the training data**
- At each iteration, the data that is wrongly classified will have its weight increased
- AdaBoost, Gradient Boosting Tree

Concepts of Boosting Tree



Tree 1 Tree 2 ..... Tree N

score1 score2 scoreN

y = score1 * learning_rate + score2 * learning_rate + ……+ scoreN * learning_rate

Gradient Boosting Tree in Python

```
from sklearn.ensemble import
GradientBoostingClassifier

clf = GradientBoostingClassifier(
        # number of boosting stages
        n_estimator = 100,
        learning_rate = 0.001,
        max_depth = 3,
        min_samples_split = 2,
        min_samples_leaf = 1,
)
```

```
# feature importance
clf.feature_importances_
```

Gradient Boosting Tree

example/06_gradient_boosting_regression.ipynb

- Compare among decision tree, random forests, and gradient boosting tree
- Apply on a regression problem
- Grid search

Grid Search of Hyper-parameter

```
from sklearn.model_selection import GridSearchCV
parameters = {‘max_depth’:[3, 5, 7, 9, 11],
              ‘criterion’:[‘gini’,’entropy’]
}

clf = DecisionTreeClassifier()

clf_grid = GridSearchCV(clf,parameters)
clf_grid.fit(X, Y)
```

Exercise

exercise/ex04_random_forest_digits_classification.ipynb

XGBoost

- Additive model (similar with gradient boosting tree)
- Feature sampling (similar with random forest)
- Add regularization in objective function
- Use 1st and 2nd derivative to help training

Installation

- Windows: http://www.jianshu.com/p/5b3e0489f1a8
- Mac/Linux: pip install xgboost

Example

example/11_xgboost.ipynb

Unsupervised Learning In-depth

Have no idea about labels ☹

Dimension reduction

- **Principal Component Analysis (PCA)**

Clustering

- K-means clustering
- Hierarchical clustering
- DBSCAN

Principal Component Analysis (PCA)

- PCA is to use **orthogonal** transformation to convert a set of data of a possible correlated variables into a set of values of **linearly uncorrelated** variables

- A new variable is one of linear combinations of the original variables

Linear Combination of Variables

- The original variables is noted as $x_1, x_2, \ldots, x_n$, and the new variables can be represented as

$$\left.\begin{aligned} z_1 &= a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n \\ z_2 &= a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n \\ &\vdots \\ z_n &= a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n \end{aligned}\right\} \text{Uncorrelated}$$

Covariance Matrix (symmetric)

- Can be viewed as $A = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & & a_{2n} \\ \vdots & & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} = \begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_n \end{bmatrix}$

$$z = A^T x \qquad A^T = [a_1 \quad a_2 \quad \cdots \quad a_n]$$

- **The transformed and uncorrelated** variables implies that the co-variance matrix is diagonal

$$\Sigma_z = E(zz^T) = AE(xx^T)A^T = A\Sigma_x A^T = \begin{bmatrix} {\sigma_1}^2 & 0 & \cdots & 0 \\ 0 & {\sigma_2}^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & {\sigma_n}^2 \end{bmatrix}$$

Assumption on Project Matrix

- Assumed that the project matrix is orthonormal

  ➔ $A^T A = I$ (identity matrix)

$$\Sigma_x = A^T \Sigma_z A = A^T \begin{bmatrix} {\sigma_1}^2 & 0 & \cdots & 0 \\ 0 & {\sigma_2}^2 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & {\sigma_n}^2 \end{bmatrix} A$$

- Spectral theorem: a real symmetric matrix can be "diagonalized" by an orthogonal matrix

Approximating Covarince Matrix

- Rank the orthogonal eigenvectors according to its eigenvalues
- The eigenvector with the largest eigenvalue is the first principal component
- Use top principal components to approximate the original covariance matrix

PCA in Python

```
from sklearn.decomposition import PCA
pca = PCA(
        n_components = 2,
        whiten = True,
        svd_solver = ‘auto’
)

# principal axes in feature space
pca.components_
# explained variance
pca.explained_variance_ratio_
```

Example

example/07_PCA.ipynb

- An example of 2D data
- Visualize PCA components
- Determine the number of components

Exercise

exercies/ex05_PCA.ipynb

Unsupervised Learning In-depth

Have no idea about labels ☹

Dimension reduction

- Principal Component Analysis

Clustering

- **K-means clustering**
- Hierarchical clustering
- DBSCAN

K-Means Clustering

- Unsupervised clustering based on “distance” among data points (usually, squared Euclidean distance)
- The goal is to minimize the within-cluster sum of squared errors (cluster inertia)
- **Should be careful of the range of each feature**

Procedure of K-means Clustering

1. Randomly pick K data points as initial cluster centers
2. Assign other data points to the closest centers
3. Re-calculate the center of each cluster
4. Repeat step 2 and 3 until the assignments of all data points are unchanged or the stopping criteria is satisfied (for example, maximal number of iterations)

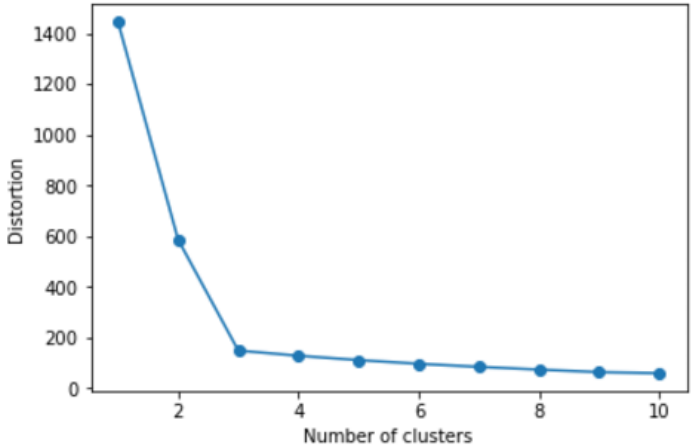
K-means Clustering in Python

```
from sklearn.cluster import KMeans
pca = KMeans(
        n_clusters = 3,
        init = ‘random’,
        n_init = 10,
        max_iter = 300,
        tol = 1e-4
)
```

- n_init = 10 means K-means clustering will perform **10 times** using different initial centers and adopt the clustering with minimal SSE

How to determine the best K

- Summation of with-cluster sum of squared error over all clusters
- Stored in attribute **inertia_**



Silhouette Analysis

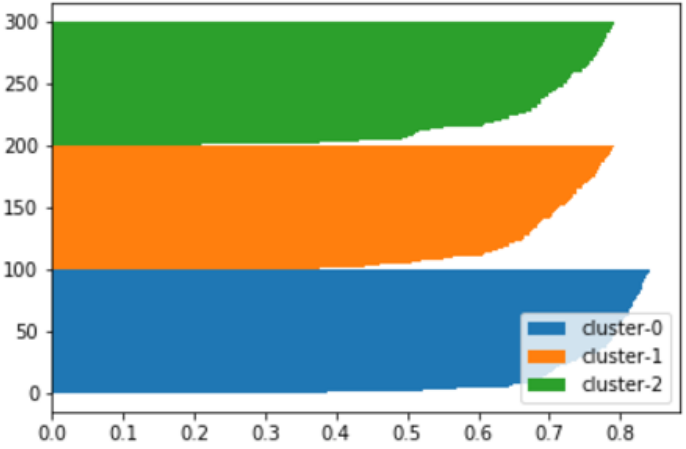
- Used to quantify the quality of clustering
- For every data point:

1. Calculate the average distances of all points in the **same** cluster, a
2. Calculate the average distance of all points in the **closest** cluster, b
3. Calculate **silhouette coefficient**, s

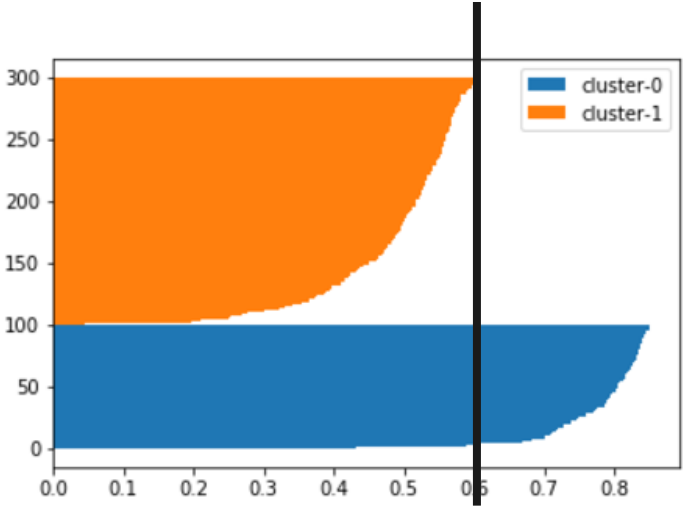
$$s = \frac{b - a}{\max(b, a)}$$

Quantification of Clustering Quality

- K=3



- K=2



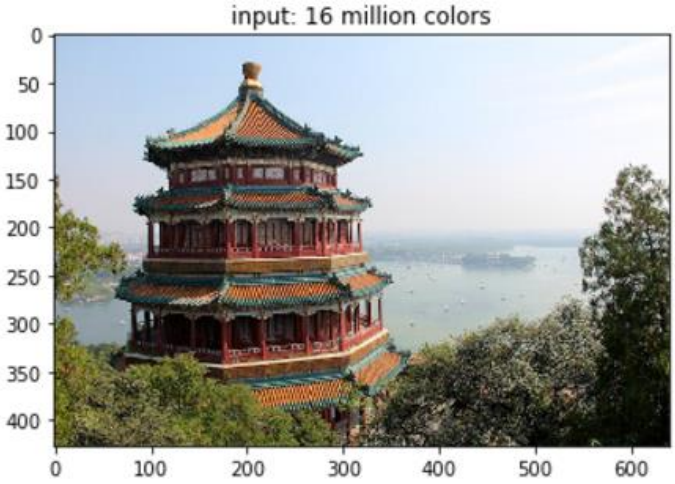
Example

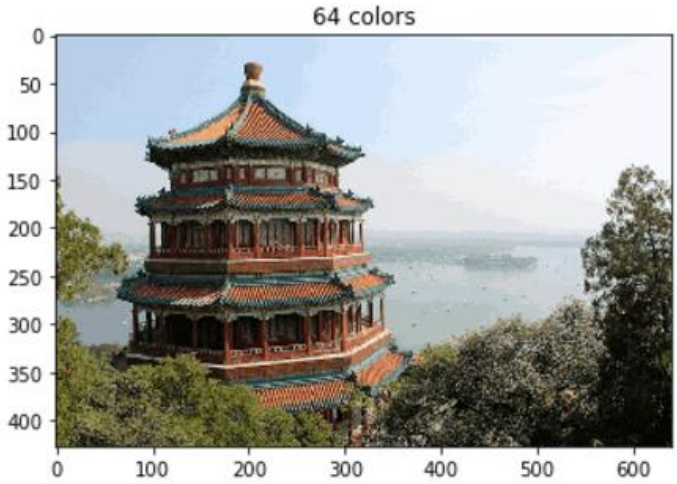
example/09_kmeans_clustering.ipynb

Exercise

exercise/kmeans_clustering_color_compression.ipynb

- Use MiniBatchKmeans instead for efficiency





Unsupervised Learning In-depth

Have no idea about labels ☹

Dimension reduction

- Principal Component Analysis

Clustering

- K-means clustering
- **Hierarchical clustering**
- DBSCAN

Hierarchical Clustering

- No need to specify the number of clusters in advance
- Procedures

1. Each data point is viewed as an individual cluster
2. Calculate the distance between any two clusters
3. Merge two closest clusters into one cluster
4. Repeat 2 and 3 until all data points are merged into one cluster

Agglomerative clustering

Hierarchical Clustering in Python

```
from sklearn.cluster import AgglomerativeClustering
ac = AgglomerativeClustering(
                n_clusters = 2,
                affinity = 'euclidean',
                linkage='complete')
```

- n_clusters= 2 means total number of clusters = 2
- Use ‘Euclidean’ as the distance metric

Example

example/10_hierarchical_clustering.ipynb

- A toy example to go through the procedure of hierarchical clustering step by step
- Visualize the dendrogram in Python

Unsupervised Learning In-depth

Have no idea about labels ☹

Dimension reduction

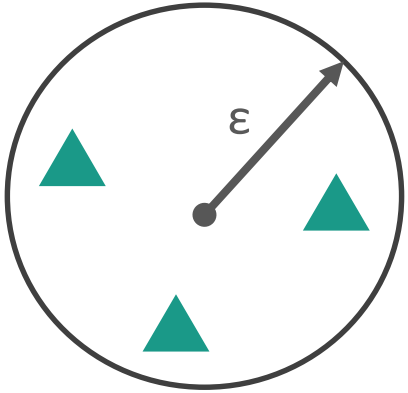
- Principal Component Analysis

Clustering

- K-means clustering
- Hierarchical clustering
- **DBSCAN**

DBSCAN

- Density-based Spatial Clustering of Applications with Noise
- Density: number of samples in range of ε



3 Types of Data Points

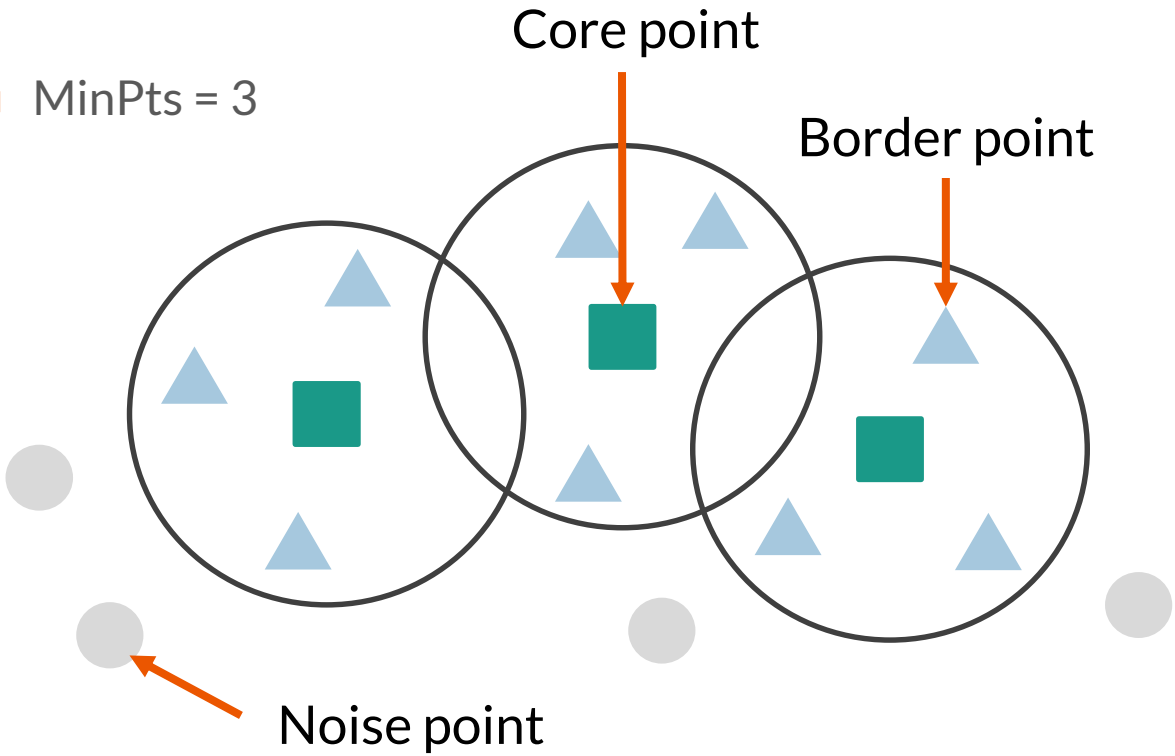
- A data point is a **core point** if there are at least MinPts neighboring data points in range of ε
- A data point is a **border point** if it is in range of a core point **and** contains less than MinPts neighboring data points
- A data point is a **noise point** if it is neither a core point nor a border point

Procedure of DBSCAN

- Two core points are **connected** if the distance between them less than ε
- An individual core point or a connected core point is a cluster
- Assign each border points to the corresponding clusters of core points

Example

- MinPts = 3



Difference among Clustering Methods

- Unlike K-means clustering, DBSCAN does not assume the spherical variance of clusters

- Unlike hierarchical clustering, DBSCAN allows to **ignore noise points**

DBSCAN in Python

```
from sklearn.cluster import DBSCAN
ac = DBSCAN(
        eps = 0.2,
        min_samples = 5,
        metric = 'euclidean' )
```

- eps = 0.2 defines range of ε = 0.2
- min_samples defines the MinPts to be a core point
- Use ‘Euclidean’ as the distance metric

Example

example/11_DBSCAN_and_review.ipynb

- Compare different clustering algorithms in a special case of moon-like data distribution

Exercise

exercise/ex08_plot_cluster_comparison.ipynb

- Show characteristics of different clustering algorithms on many interesting datasets

Recap: Supervised Learning

Method	Estimator in scikit-learn
Linear regression	`from sklearn.linear_model import LinearRegression`
Regression with regularization	`from sklearn.linear_model import Lasso, Ridge`
Logistic regression	`from sklearn.linear_model import LogisticRegression`
Support vector machine	`from sklearn.svm import SVC, SVR`
Decision tree	`from sklearn.tree import DecisionTreeClassifier, DecisionRegressor`
Random forests	`from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor`
XGBoost	`from xgboost import XGBClassifier`

Recap: Unsupervised Learning

Dimension reduction

Method	Estimator in scikit-learn
Principal component analysis	`from sklearn.decomposition import PCA`

Clustering

Method	Estimator in scikit-learn
K-means clustering	`from sklearn.decomposition import PCA`
Hierarchical clustering	`from sklearn.cluster import AgglomerativeClustering`
DBSCAN	`from sklearn.cluster import DBSCAN`

Thank you!

If any questions, please feel free to contact
cmchang@iis.sinica.edu.tw

中央研究院資訊科學所資料洞察實驗室