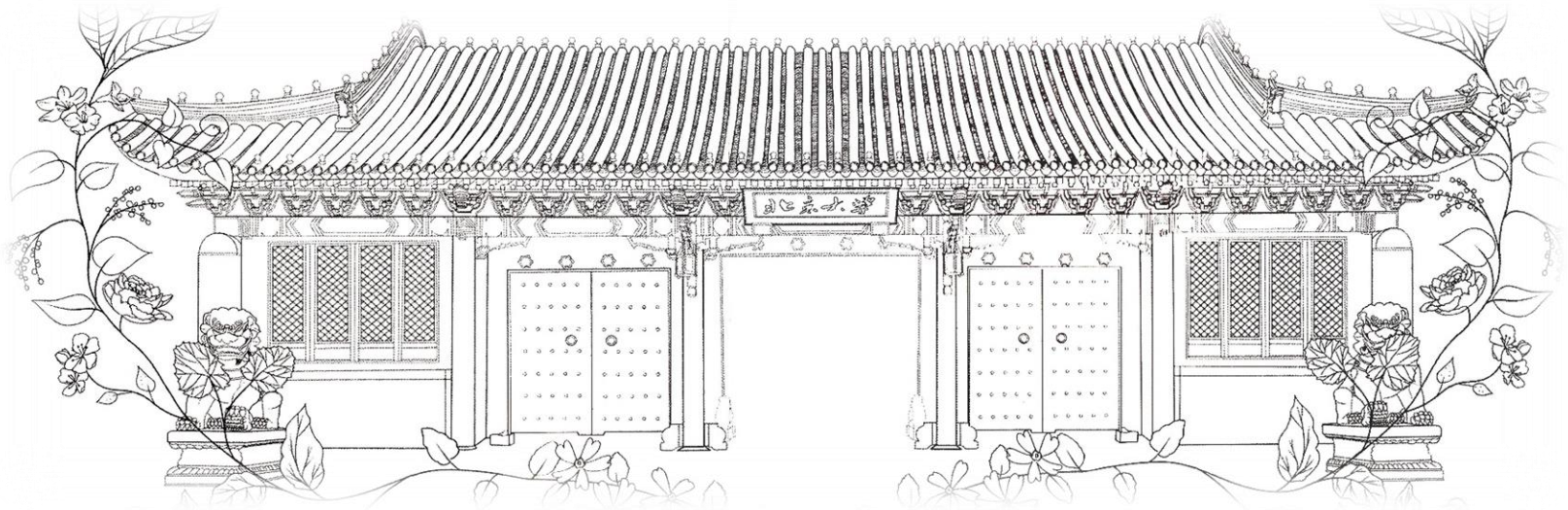
《Python数据分析》
机器学习基础



目录

- 机器学习的概念
  - 概念
  - 经典问题
    - 分类
    - 回归
    - 聚类
- 利用scikit-learn进行机器学习
  - 预处理
  - 分类
  - 回归
  - 聚类

机器学习的概念

- **机器学习**是一门多学科交叉专业，涵盖概率论知识，统计学知识，近似理论知识和复杂算法知识，使用计算机作为工具并致力于真实、实时的模拟人类学习方式，并将现有内容进行知识结构划分来有效提高学习效率。
- 机器学习有下面几种定义：

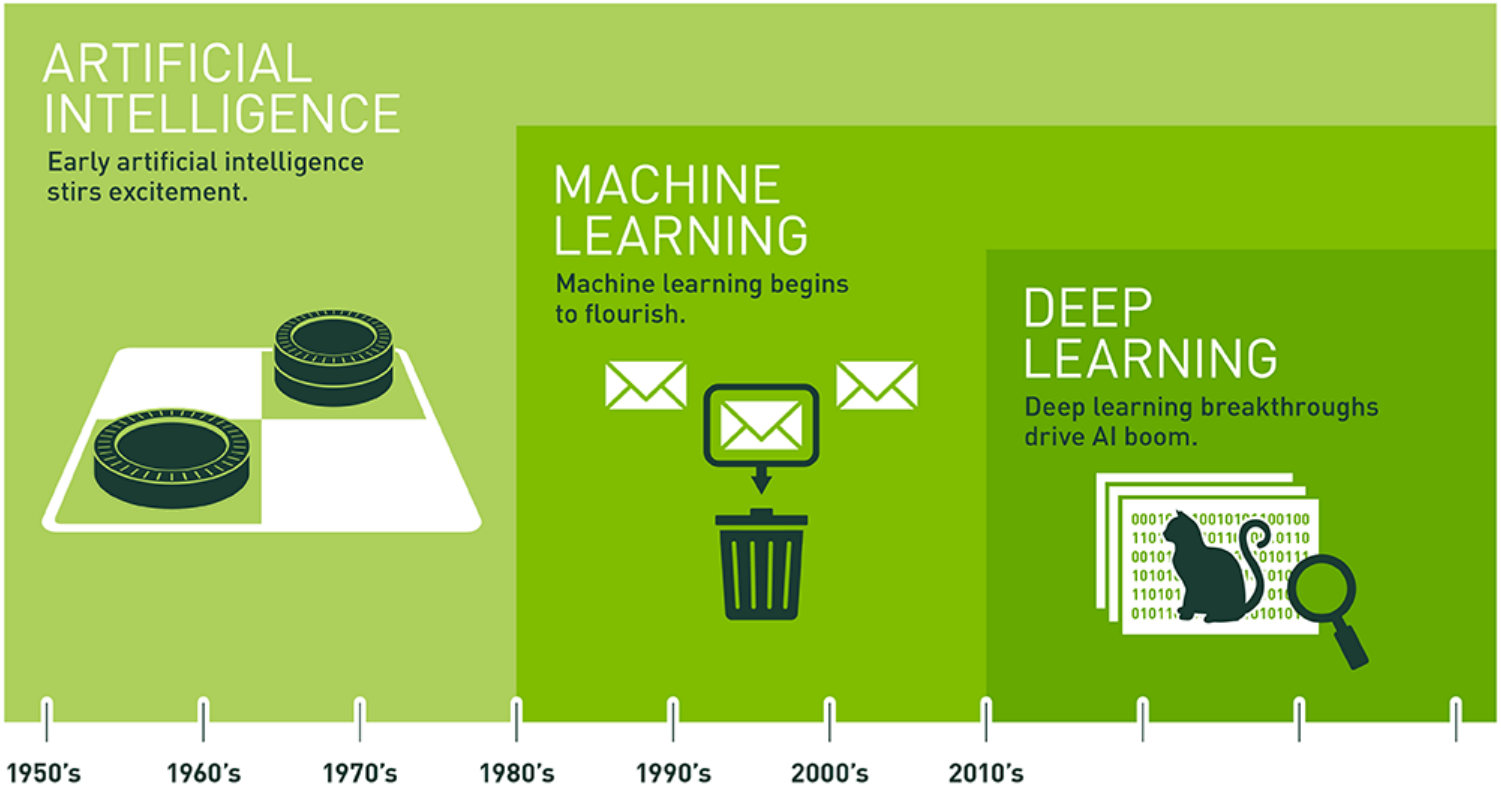
（1）机器学习是一门人工智能的科学，该领域的主要研究对象是人工智能，特别是如何在经验学习中改善具体算法的性能。

（2）机器学习是对能通过经验自动改进的计算机算法的研究。

（3）机器学习是用数据或以往的经验，以此优化计算机程序的性能标准。

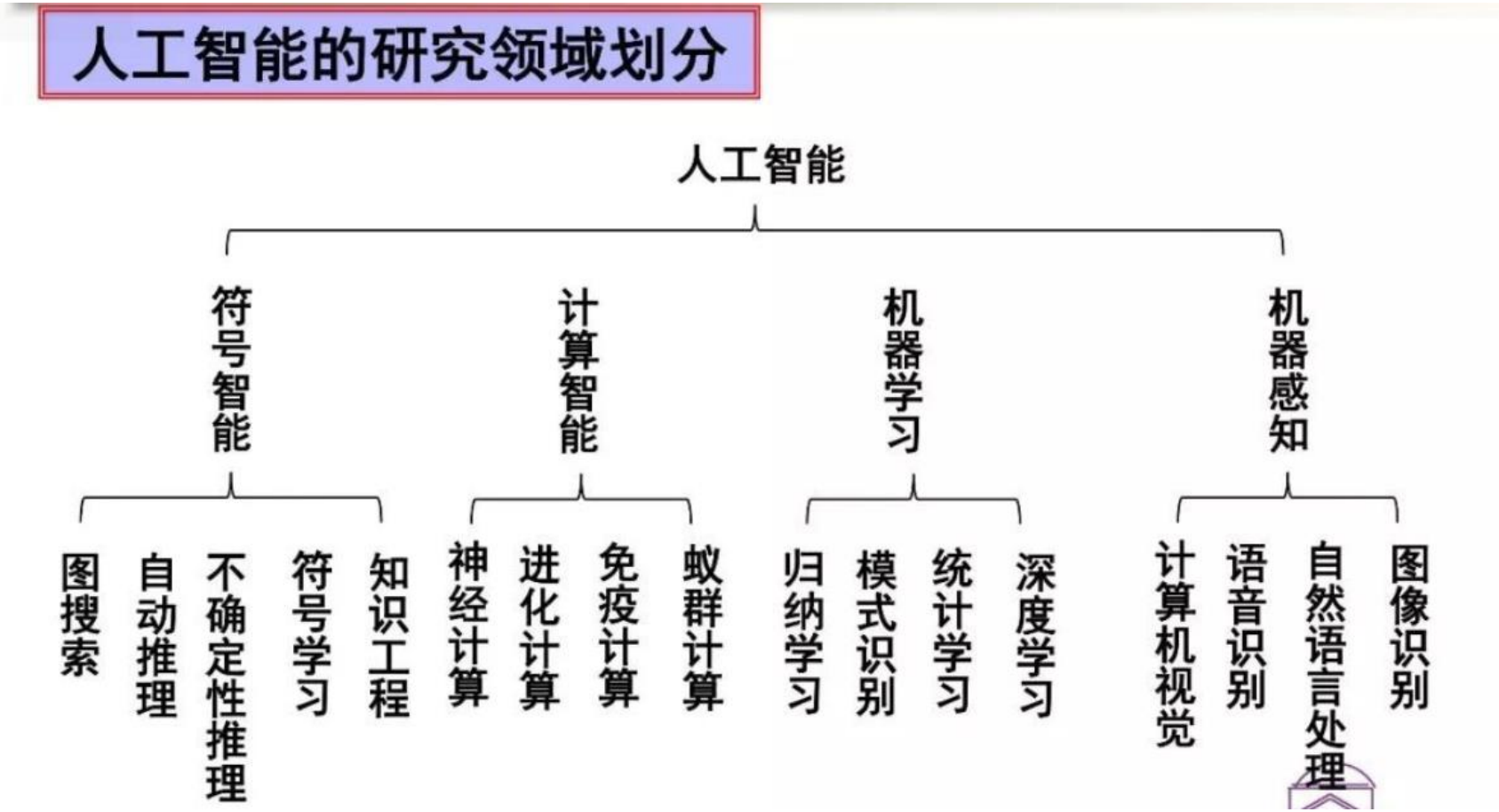
- 三个关键词：算法、经验、性能

机器学习的概念

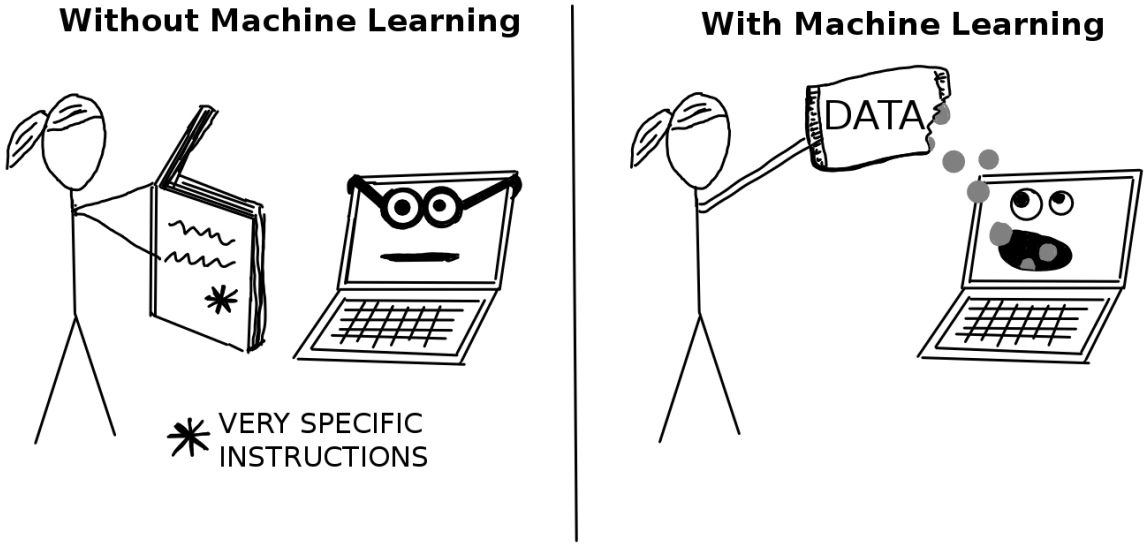


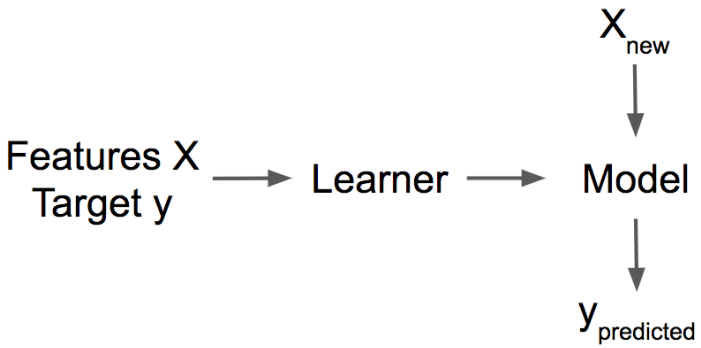
Since an early flush of optimism in the 1950s, smaller subsets of artificial intelligence – first machine learning, then deep learning, a subset of machine learning – have created ever larger disruptions.

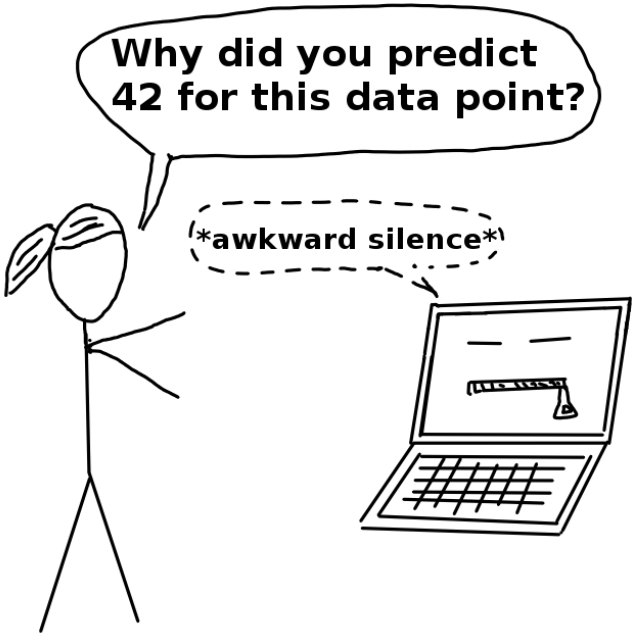
机器学习的概念


人工智能的研究领域划分
人工智能
符号智能
计算智能
机器学习
机器感知
知识工程
符号学习
不确定性推理
自动推理
图搜索
蚁群计算
免疫计算
进化计算
神经计算
深度学习
统计学习
模式识别
归纳学习
图像识别
自然语言处理
语音识别
计算机视觉

机器学习的概念

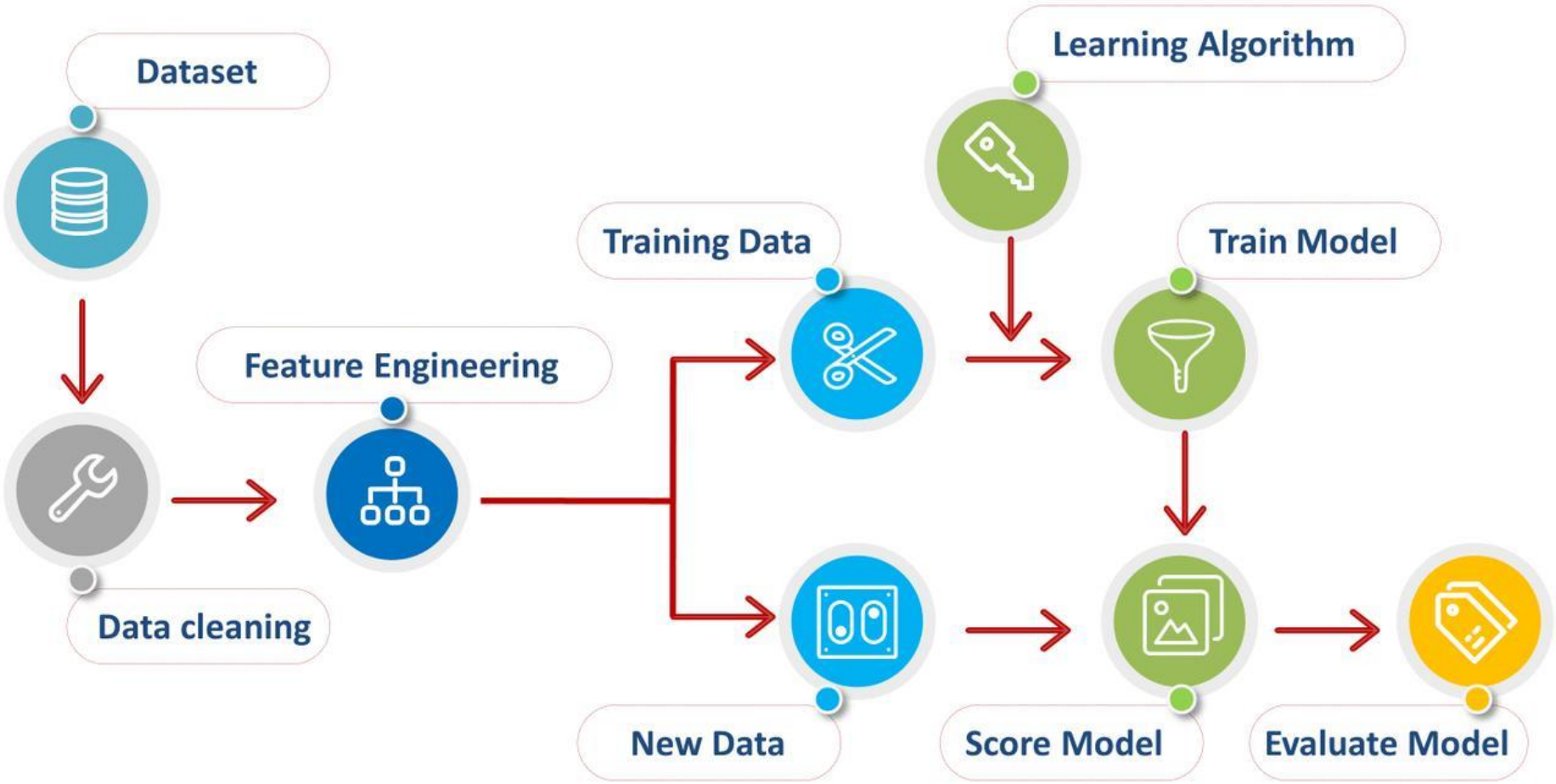






https://christophm.github.io/interpretable-ml-book/what-is-machine-learning.html

机器学习的一般流程


Dataset
Learning Algorithm
Training Data
Train Model
Feature Engineering
Data cleaning
New Data
Score Model
Evaluate Model

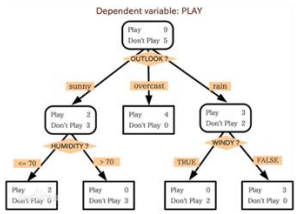
目录

- 机器学习的概念
  - 概念
  - 经典问题
    - 分类
    - 回归
    - 聚类
- 利用scikit-learn进行机器学习
  - 预处理
  - 分类
  - 回归
  - 聚类

分类

- 任务：识别对象所属的类别
- 应用场景：垃圾邮件检测、图像识别
- 算法：

1）决策树



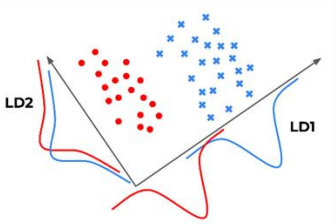
2）朴素贝叶斯

$$p(C|F_1,\dots,F_n)$$

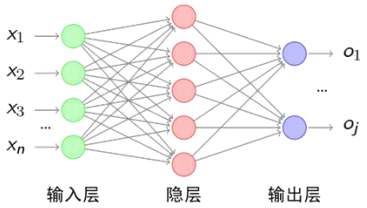
$$p(C|F_1,\dots,F_n) = \frac{p(C)\,p(F_1,\dots,F_n|C)}{p(F_1,\dots,F_n)}.$$

$$\text{posterior} = \frac{\text{prior} \times \text{likelihood}}{\text{evidence}}.$$

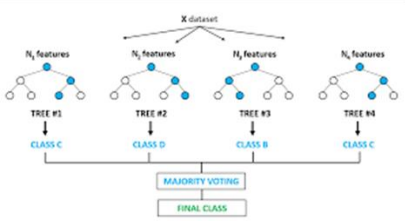
3）判别分析



4）神经网络

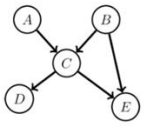


5）随机森林



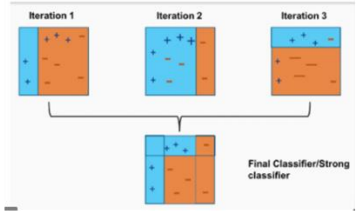
6）贝叶斯网络

$$p(a,b,c,d,e) = p(a)p(b)p(c|a,b)p(d|c)p(e|b,c)$$

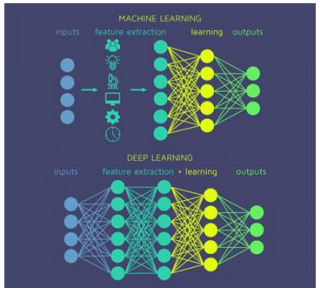


$p(E|B,C)$

7）Adaboost



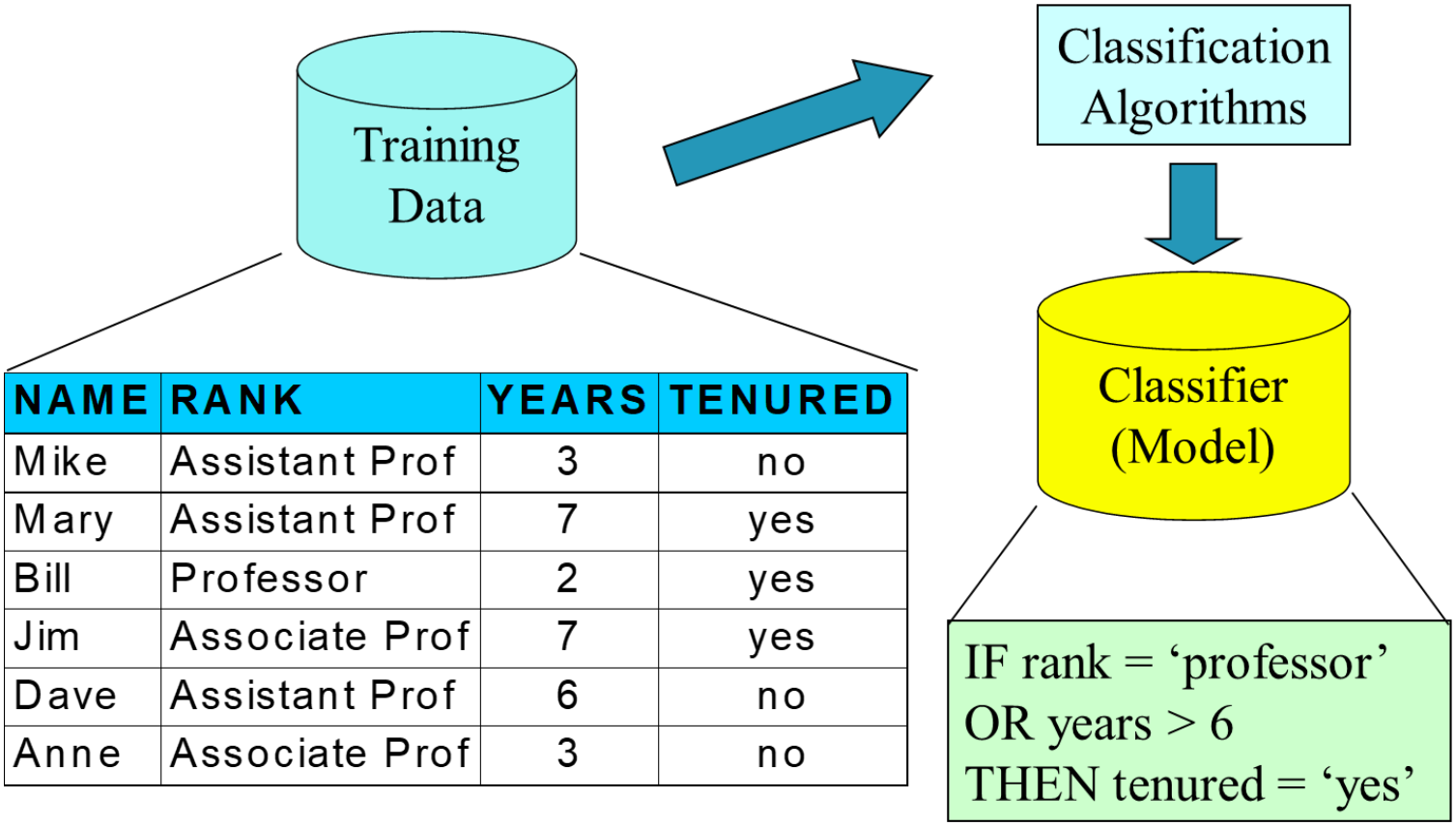
8）深度学习



……

分类

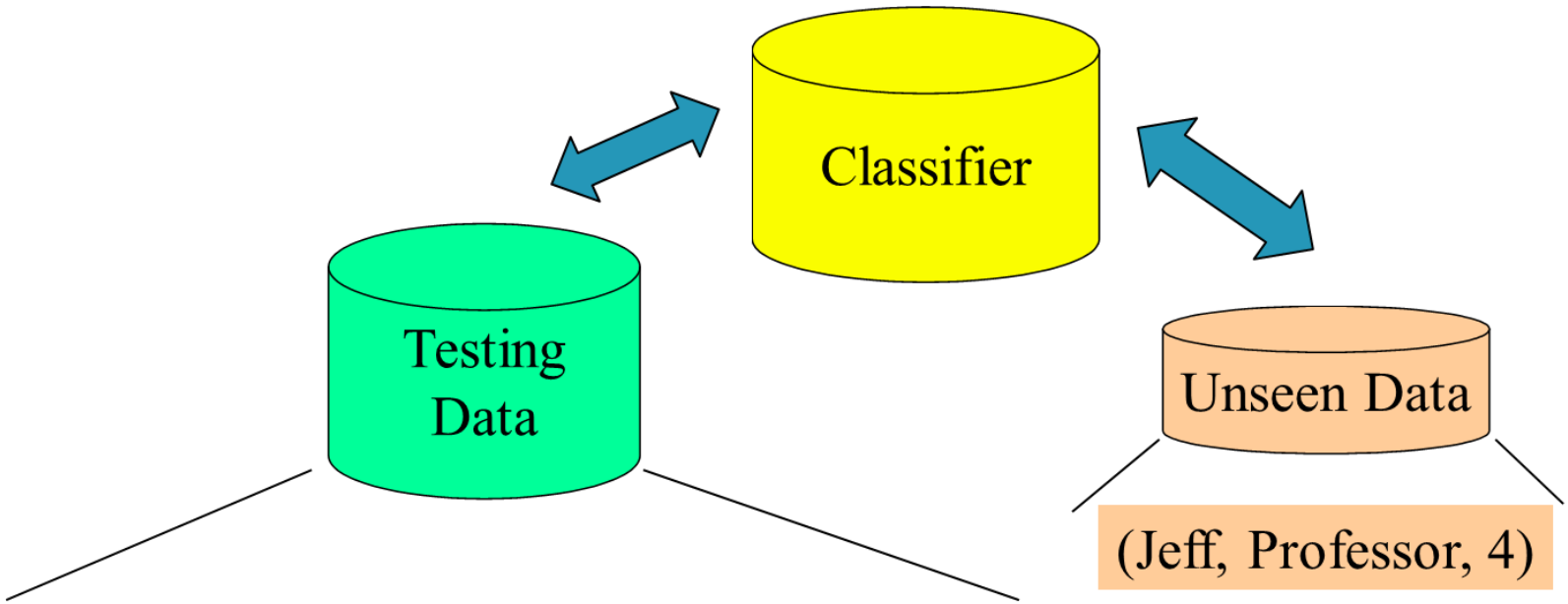
- 训练



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

分类

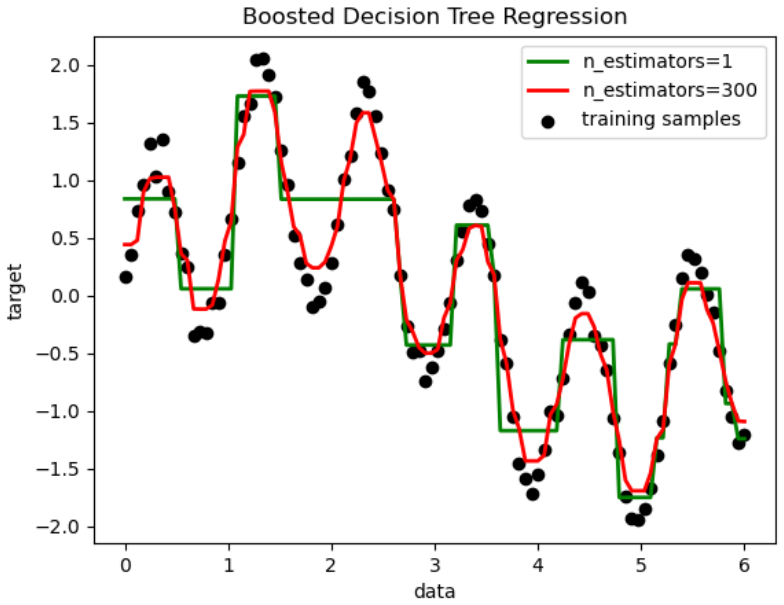
- 测试与应用



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

回归

- 任务：预测与对象相关的连续值属性
- 应用场景：股价预测、房屋定价
- 算法：
  - 线性回归
  - 支持向量回归机（SVR）
  - k近邻
  - 决策树
  - 随机森林
  - ……



聚类

- 任务：自动地将相似的样本分到同一组
- 应用场景：
  - 固有结构识别
    - 基于用户位置信息的商业选址
    - 客户细分
    - 生物种群固有结构认知
    - 图像分割
  - 异常检测
    - 恶意流量识别
    - 异常交易识别
  - 数据预处理
    - 离散化
    - 自动标记

Source image.

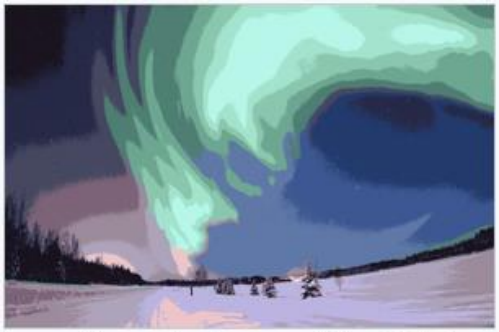


Image after running *k*-means with *k* = 16. Note that a common technique to improve performance for large images is to downsample the image, compute the clusters, and then reassign the values to the larger image if necessary.

聚类

- 算法：
  - 层次聚类
    AGNES (AGglomerative NESting Clustering)
    DIANA (DIvisive ANAlysis Clustering)
  - 划分聚类
    K-means
    K-medoids (PAM, Partitioning Around Medoid)
  - 基于密度的聚类
    DBSCAN (Density-Based Spatial Clustering of Application with Noise)
    OPTICS (Ordering Points To Identify the Clustering Structure)
    Mean Shift
    DP (Clustering by fast search and find of density peaks)
  - 谱聚类 (Spectral clustering)
  - BIRCH

目录

- 机器学习的概念
  - 概念
  - 经典问题
    - 分类
    - 回归
    - 聚类
- 利用scikit-learn进行机器学习
  - 预处理
  - 分类
  - 回归
  - 聚类

Scikit-learn

- Scikit-learn（以前称为scikits.learn，也称为sklearn）是基于 Python 语言的机器学习工具。
- 特点：
  - 简单高效的数据挖掘和数据分析工具
  - 可供所有人在各种环境中重复使用
  - 建立在 NumPy，SciPy 和 matplotlib 上
  - 开源，可商业使用 - BSD许可证

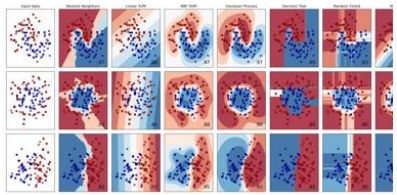
https://scikit-learn.org/stable/index.html

Scikit-learn

Classification

Identifying which category an object belongs to.

Applications: Spam detection, image recognition.
Algorithms: SVM, nearest neighbors, random forest, and more...

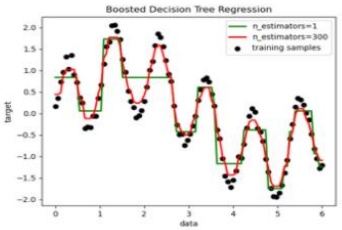


Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, nearest neighbors, random forest, and more...

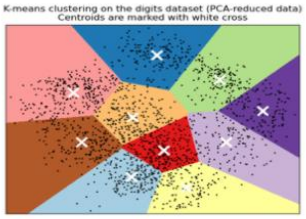


Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, and more...

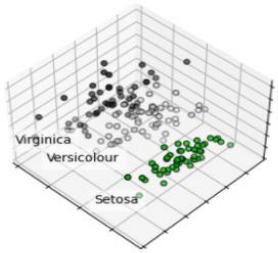


Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: k-Means, feature selection, non-negative matrix factorization, and more...

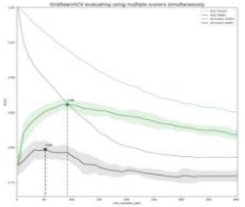


Examples

Model selection

Comparing, validating and choosing parameters and models.

Applications: Improved accuracy via parameter tuning
Algorithms: grid search, cross validation, metrics, and more...

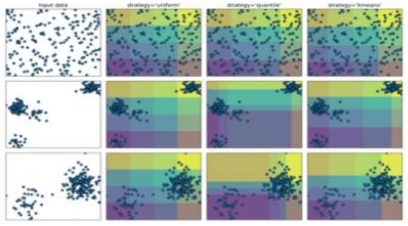


Examples

Preprocessing

Feature extraction and normalization.

Applications: Transforming input data such as text for use with machine learning algorithms.
Algorithms: preprocessing, feature extraction, and more...



Examples

预处理

- sklearn.preprocessing进行数据预处理的主要功能：
  - 标准化（Standardization）
  - 非线性变换
  - 归一化（Normalization）
  - 标称特征编码
  - 离散化
  - 推定缺失数据
  - 生成多项式特征
  - 定制转换器

标准化（Standardization）

- 将每个特征的均值处理为0、方差处理为1的形式。
- StandardScaler()

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

>>> scaler.mean_
array([1. ..., 0. ..., 0.33...])

>>> scaler.scale_
array([0.81..., 0.81..., 1.24...])

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

```
>>> X_scaled.mean(axis=0)
array([0., 0., 0.])

>>> X_scaled.std(axis=0)
array([1., 1., 1.])
```

标准化（Standardization）

- 将每个特征的分布转化到指定区间。
- MinMaxScaler()

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> min_max_scaler = preprocessing.MinMaxScaler()
>>> X_train_minmax = min_max_scaler.fit_transform(X_train)
>>> X_train_minmax
array([[0.5       , 0.        , 1.        ],
       [1.        , 0.5       , 0.33333333],
       [0.        , 1.        , 0.        ]])
```

```
>>> X_test = np.array([[-3., -1.,  4.]])
>>> X_test_minmax = min_max_scaler.transform(X_test)
>>> X_test_minmax
array([[-1.5       ,  0.        ,  1.66666667]])
```

默认参数
feature_range=(0, 1)

If `MinMaxScaler` is given an explicit `feature_range=(min, max)` the full formula is:

```
X_std = (X - X.min(axis=0)) / (X.max(axis=0) - X.min(axis=0))

X_scaled = X_std * (max - min) + min
```

标准化（Standardization）

- 将每个特征的分布转化到指定区间。
- MaxAbsScaler()

```
>>> X_train = np.array([[ 1., -1.,  2.],
...                     [ 2.,  0.,  0.],
...                     [ 0.,  1., -1.]])
...
>>> max_abs_scaler = preprocessing.MaxAbsScaler()
>>> X_train_maxabs = max_abs_scaler.fit_transform(X_train)
>>> X_train_maxabs
array([[ 0.5, -1. ,  1. ],
       [ 1. ,  0. ,  0. ],
       [ 0. ,  1. , -0.5]])
>>> X_test = np.array([[ -3., -1.,  4.]])
>>> X_test_maxabs = max_abs_scaler.transform(X_test)
>>> X_test_maxabs
array([[-1.5, -1. ,  2. ]])
>>> max_abs_scaler.scale_
array([2.,  1.,  2.])
```

转换后最大绝对值为1

归一化（Normalization）

- 将每个样本转化到具有相同的模。
- normalize()

```
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]
>>> X_normalized = preprocessing.normalize(X, norm='l2')

>>> X_normalized
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])
```

- Normalizer()

```
>>> normalizer = preprocessing.Normalizer().fit(X)  # fit does nothing
>>> normalizer
Normalizer()
```

```
>>> normalizer.transform(X)
array([[ 0.40..., -0.40...,  0.81...],
       [ 1.  ...,  0.  ...,  0.  ...],
       [ 0.  ...,  0.70..., -0.70...]])

>>> normalizer.transform([[-1.,  1., 0.]])
array([[-0.70...,  0.70...,  0.  ...]])
```

标称特征编码

- 把标称型特征(categorical features) 转换为整数编码(integer codes)
- OrdinalEncoder()

```
>>> enc = preprocessing.OrdinalEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OrdinalEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari']])
array([[0., 1., 1.]])
```

- OneHotEncoder()

```
>>> enc = preprocessing.OneHotEncoder()
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder()
>>> enc.transform([['female', 'from US', 'uses Safari'],
...                ['male', 'from Europe', 'uses Safari']]).toarray()
array([[1., 0., 0., 1., 0., 1.],
       [0., 1., 1., 0., 0., 1.]])
```

标称特征编码

- OneHotEncoder()
- 可以指定类别（决定编码位数）

```
>>> genders = ['female', 'male']
>>> locations = ['from Africa', 'from Asia', 'from Europe', 'from US']
>>> browsers = ['uses Chrome', 'uses Firefox', 'uses IE', 'uses Safari']
>>> enc = preprocessing.OneHotEncoder(categories=[genders, locations, browsers])
>>> # Note that for there are missing categorical values for the 2nd and 3rd
>>> # feature
>>> X = [['male', 'from US', 'uses Safari'], ['female', 'from Europe', 'uses Firefox']]
>>> enc.fit(X)
OneHotEncoder(categories=[['female', 'male'],
                          ['from Africa', 'from Asia', 'from Europe',
                           'from US'],
                          ['uses Chrome', 'uses Firefox', 'uses IE',
                           'uses Safari']])
>>> enc.transform([['female', 'from Asia', 'uses Chrome']]).toarray()
array([[1., 0., 0., 1., 0., 0., 1., 0., 0., 0.]])
```

离散化

- 分箱：将每个特征的取值转化为分箱编号
- KBinsDiscretizer()

```
>>> X = np.array([[ -3., 5., 15 ],
...               [  0., 6., 14 ],
...               [  6., 3., 11 ]])
>>> est = preprocessing.KBinsDiscretizer(n_bins=[3, 2, 2], encode='ordinal').fit(X)
```

- 分箱边界
  - feature 1: $[-\infty, -1), [-1, 2), [2, \infty)$
  - feature 2: $[-\infty, 5), [5, \infty)$
  - feature 3: $[-\infty, 14), [14, \infty)$

```
>>> est.transform(X)
array([[ 0., 1., 1.],
       [ 1., 1., 1.],
       [ 2., 0., 0.]])
```

离散化

- 二值化：将每个特征根据阈值转化为0、1取值
- Binarizer()

```
>>> X = [[ 1., -1.,  2.],
...      [ 2.,  0.,  0.],
...      [ 0.,  1., -1.]]

>>> binarizer = preprocessing.Binarizer().fit(X)  # fit does nothing
>>> binarizer
Binarizer()

>>> binarizer.transform(X)
array([[1., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.]])
```

- 阈值默认为0，可调整：

```
>>> binarizer = preprocessing.Binarizer(threshold=1.1)
>>> binarizer.transform(X)
array([[0., 0., 1.],
       [1., 0., 0.],
       [0., 0., 0.]])
```

生成多项式特征

- 通过增加一些输入数据的非线性特征来增加模型的复杂度。一个简单通用的办法是使用多项式特征，包括高次项和交互项。
- PolynomialFeatures()

```
>>> import numpy as np
>>> from sklearn.preprocessing import PolynomialFeatures
>>> X = np.arange(6).reshape(3, 2)
>>> X
array([[0, 1],
       [2, 3],
       [4, 5]])
>>> poly = PolynomialFeatures(2)
>>> poly.fit_transform(X)
array([[ 1.,  0.,  1.,  0.,  0.,  1.],
       [ 1.,  2.,  3.,  4.,  6.,  9.],
       [ 1.,  4.,  5., 16., 20., 25.]])
```

- 特征转化：

(X_1, X_2) to $(1, X_1, X_2, X_1^2, X_1X_2, X_2^2)$.

生成多项式特征

- 可以选择只保留高次交互项
- 通过参数interaction_only=True指定：

```
>>> X = np.arange(9).reshape(3, 3)
>>> X
array([[0, 1, 2],
       [3, 4, 5],
       [6, 7, 8]])
>>> poly = PolynomialFeatures(degree=3, interaction_only=True)
>>> poly.fit_transform(X)
array([[  1.,   0.,   1.,   2.,   0.,   0.,   2.,   0.],
       [  1.,   3.,   4.,   5.,  12.,  15.,  20.,  60.],
       [  1.,   6.,   7.,   8.,  42.,  48.,  56., 336.]])
```

- 特征转化：

(X_1, X_2, X_3) to $(1, X_1, X_2, X_3, X_1X_2, X_1X_3, X_2X_3, X_1X_2X_3)$.

定制转换器

- 根据需求从任意函数中实现一个转换器
- FunctionTransformer()
- 例如可以实现对数转换

```
>>> import numpy as np
>>> from sklearn.preprocessing import FunctionTransformer
>>> transformer = FunctionTransformer(np.log1p, validate=True)
>>> X = np.array([[0, 1], [2, 3]])
>>> transformer.transform(X)
array([[0.        , 0.69314718],
       [1.09861229, 1.38629436]])
```

目录

- 机器学习的概念
  - 概念
  - 经典问题
    - 分类
    - 回归
    - 聚类
- 利用scikit-learn进行机器学习
  - 预处理
  - 分类
  - 回归
  - 聚类

普通最小二乘

- 根据给定的X和y，根据如下目标，求得系数w：

$$\min_{w} ||Xw - y||_2^2$$

- 根据w，可以对每个样本x进行预测得到$\tilde{y}$：

$$\hat{y}(w, x) = w_0 + w_1 x_1 + \ldots + w_p x_p$$

- LinearRegression()：回归模型

```
>>> from sklearn import linear_model
>>> reg = linear_model.LinearRegression()
>>> reg.fit([[0, 0], [1, 1], [2, 2]], [0, 1, 2])
LinearRegression()
>>> reg.coef_
array([0.5, 0.5])
```

普通最小二乘

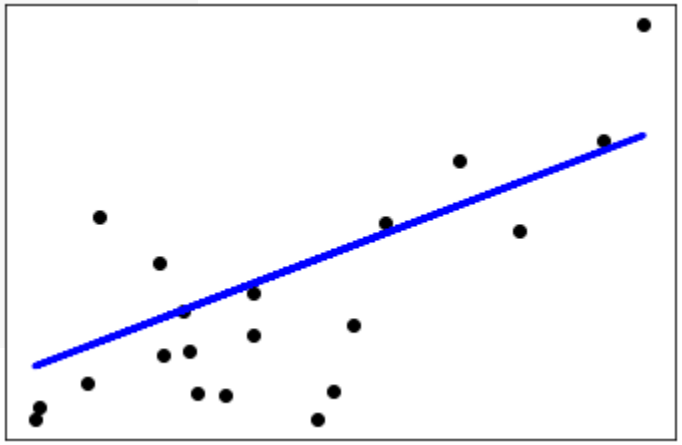
- LinearRegression()

```
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, r2_score

# Load the diabetes dataset
diabetes_X, diabetes_y = datasets.load_diabetes(return_X_y=True)

# Use only one feature
diabetes_X = diabetes_X[:, np.newaxis, 2]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes_y[:-20]
diabetes_y_test = diabetes_y[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

普通最小二乘

- LinearRegression()

```
# Make predictions using the testing set
diabetes_y_pred = regr.predict(diabetes_X_test)

# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(diabetes_y_test, diabetes_y_pred))
# The coefficient of determination: 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(diabetes_y_test, diabetes_y_pred))

# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, diabetes_y_pred, color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



逻辑回归(Logistic regression)

- LogisticRegression()：
- 分类模型，别名：logit regression、maximum-entropy classification (MaxEnt)、log-linear classifier
- 经典的基于不同的正则项，可以有不同的模型，
- l_2正则逻辑回归的优化目标：

$$\min_{w,c} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

- l_1正则逻辑回归的优化目标：

$$\min_{w,c} \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1).$$

- $Elastic - Net$正则逻辑回归的优化目标：

$$\min_{w,c} \frac{1-\rho}{2} w^T w + \rho \|w\|_1 + C \sum_{i=1}^{n} \log(\exp(-y_i(X_i^T w + c)) + 1),$$

逻辑回归(Logistic regression)

- LogisticRegression()：

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

from sklearn.linear_model import LogisticRegression
from sklearn import datasets
from sklearn.preprocessing import StandardScaler

X, y = datasets.load_digits(return_X_y=True)

X = StandardScaler().fit_transform(X)

# classify small against large digits
y = (y > 4).astype(int)

l1_ratio = 0.5  # L1 weight in the Elastic-Net regularization

fig, axes = plt.subplots(3, 3)
```

逻辑回归(Logistic regression)

```
# Set regularization parameter
for i, (C, axes_row) in enumerate(zip((1, 0.1, 0.01), axes)):
    # turn down tolerance for short training time
    clf_l1_LR = LogisticRegression(C=C, penalty='l1', tol=0.01, solver='saga')
    clf_l2_LR = LogisticRegression(C=C, penalty='l2', tol=0.01, solver='saga')
    clf_en_LR = LogisticRegression(C=C, penalty='elasticnet', solver='saga',
                                   l1_ratio=l1_ratio, tol=0.01)
    clf_l1_LR.fit(X, y)
    clf_l2_LR.fit(X, y)
    clf_en_LR.fit(X, y)

    coef_l1_LR = clf_l1_LR.coef_.ravel()
    coef_l2_LR = clf_l2_LR.coef_.ravel()
    coef_en_LR = clf_en_LR.coef_.ravel()

    # coef_l1_LR contains zeros due to the
    # L1 sparsity inducing norm

    sparsity_l1_LR = np.mean(coef_l1_LR == 0) * 100
    sparsity_l2_LR = np.mean(coef_l2_LR == 0) * 100
    sparsity_en_LR = np.mean(coef_en_LR == 0) * 100
```

逻辑回归(Logistic regression)

```
    print("C=%.2f" % C)
    print("{:<40} {:.2f}%".format("Sparsity with L1 penalty:", sparsity_l1_LR))
    print("{:<40} {:.2f}%".format("Sparsity with Elastic-Net penalty:",
                                  sparsity_en_LR))
    print("{:<40} {:.2f}%".format("Sparsity with L2 penalty:", sparsity_l2_LR))
    print("{:<40} {:.2f}".format("Score with L1 penalty:",
                                 clf_l1_LR.score(X, y)))
    print("{:<40} {:.2f}".format("Score with Elastic-Net penalty:",
                                 clf_en_LR.score(X, y)))
    print("{:<40} {:.2f}".format("Score with L2 penalty:",
                                 clf_l2_LR.score(X, y)))

    if i == 0:
        axes_row[0].set_title("L1 penalty")
        axes_row[1].set_title("Elastic-Net\nl1_ratio = %s" % l1_ratio)
        axes_row[2].set_title("L2 penalty")

    for ax, coefs in zip(axes_row, [coef_l1_LR, coef_en_LR, coef_l2_LR]):
        ax.imshow(np.abs(coefs.reshape(8, 8)), interpolation='nearest',
                  cmap='binary', vmax=1, vmin=0)
        ax.set_xticks(())
        ax.set_yticks(())

    axes_row[0].set_ylabel('C = %s' % C)

plt.show()
```

决策树

【回顾】

- 利用一系列特征值分割进行分类回归的模型。
- 如何利用树模型完成分类回归任务？
  - 分类任务：树中每一个节点对应一个类别
  - 回归任务：树中每一个节点对应一个预测值，多个节点之间的值可能不同。对比线性模型等，树模型的预测结果更像是分段定值函数。

决策树

- 分类：DecisionTreeClassifier()

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

- 预测结果：

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

- 预测概率：

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

决策树

- 结果可视化：

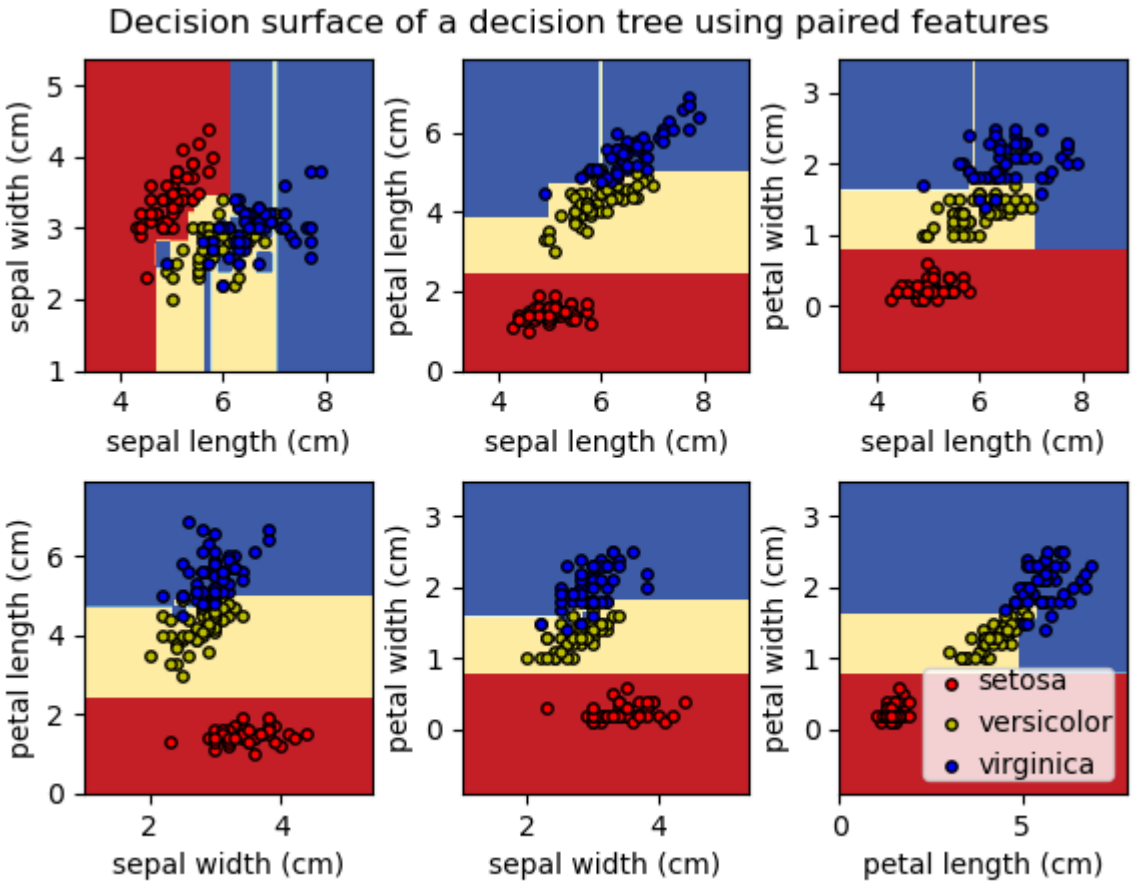
```
import numpy as np
import matplotlib.pyplot as plt

from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, plot_tree

# Parameters
n_classes = 3
plot_colors = "ryb"
plot_step = 0.02

# Load data
iris = load_iris()

for pairidx, pair in enumerate([[0, 1], [0, 2], [0, 3],
                                [1, 2], [1, 3], [2, 3]]):
    # We only take the two corresponding features
    X = iris.data[:, pair]
    y = iris.target

    # Train
    clf = DecisionTreeClassifier().fit(X, y)

    # Plot the decision boundary
    plt.subplot(2, 3, pairidx + 1)
```

决策树

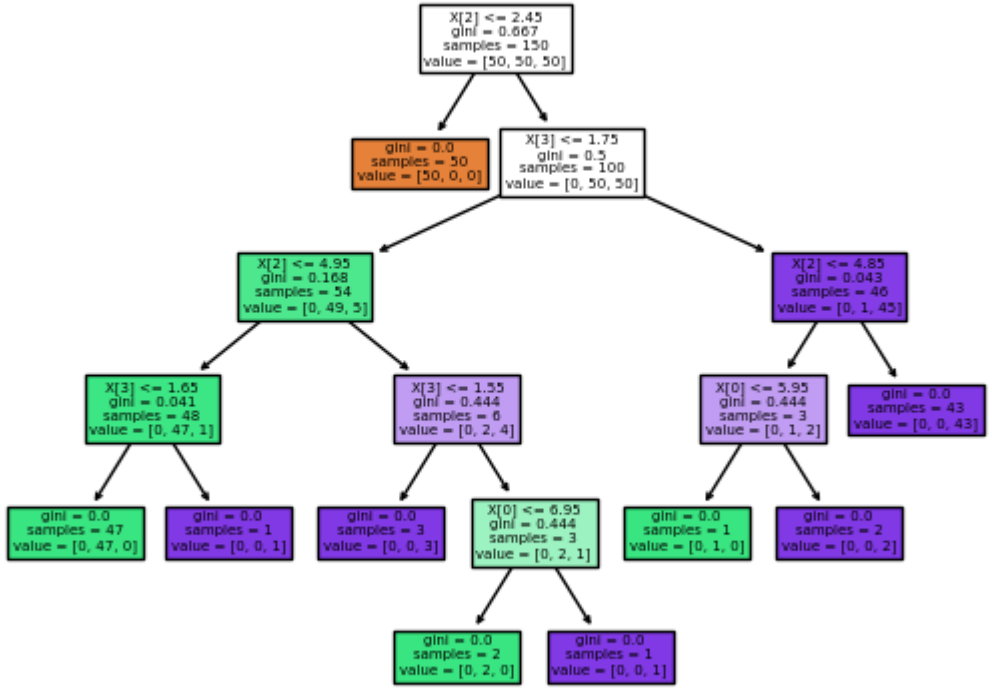
np.meshgrid:生成网格点坐标
plt.contourf: 填充三维等高线

```
    x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
    y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                         np.arange(y_min, y_max, plot_step))
    plt.tight_layout(h_pad=0.5, w_pad=0.5, pad=2.5)

    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    cs = plt.contourf(xx, yy, Z, cmap=plt.cm.RdYlBu)

    plt.xlabel(iris.feature_names[pair[0]])
    plt.ylabel(iris.feature_names[pair[1]])

    # Plot the training points
    for i, color in zip(range(n_classes), plot_colors):
        idx = np.where(y == i)
        plt.scatter(X[idx, 0], X[idx, 1], c=color, label=iris.target_names[i],
                    cmap=plt.cm.RdYlBu, edgecolor='black', s=15)

plt.suptitle("Decision surface of a decision tree using paired features")
plt.legend(loc='lower right', borderpad=0, handletextpad=0)
plt.axis("tight")

plt.figure()
clf = DecisionTreeClassifier().fit(iris.data, iris.target)
plot_tree(clf, filled=True)
plt.show()
```

决策树


Decision surface of a decision tree using paired features
sepal width (cm)
sepal length (cm)
petal length (cm)
sepal length (cm)
petal width (cm)
sepal length (cm)
petal length (cm)
sepal width (cm)
petal width (cm)
sepal width (cm)
petal width (cm)
petal length (cm)
setosa
versicolor
virginica

决策树


X[2] <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
gini = 0.0
samples = 50
value = [50, 0, 0]
X[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
X[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
X[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
X[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
X[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
X[0] <= 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
X[0] <= 6.95
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

决策树

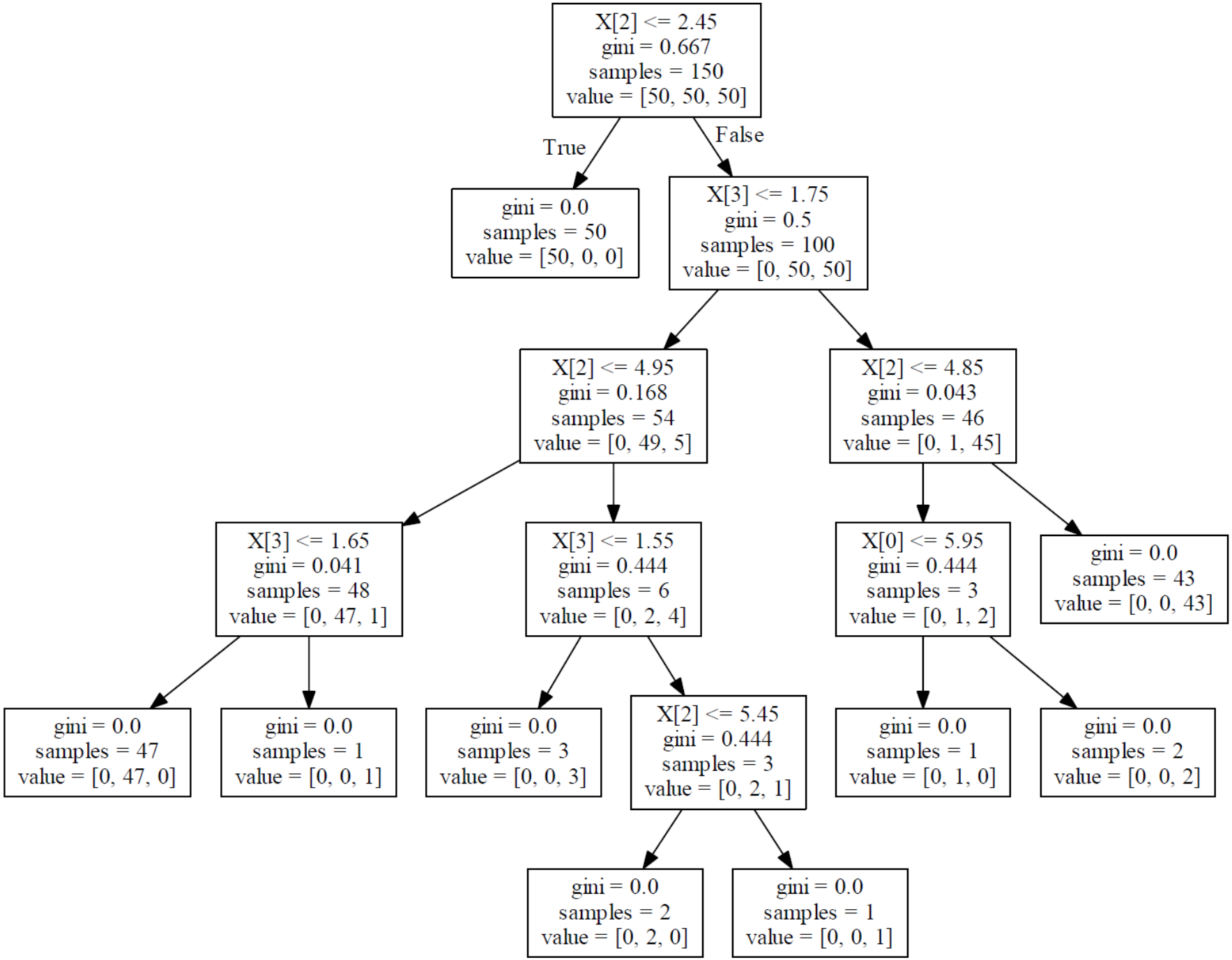
- 也在安装python-graphviz包后利用export_graphviz导出可视化结果
- 安装方式：

  conda install python-graphviz

或 pip install graphviz

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

```
dot_data = tree.export_graphviz(clf, out_file=None,
                      feature_names=iris.feature_names,
                      class_names=iris.target_names,
                      filled=True, rounded=True,
                      special_characters=True)
graph = graphviz.Source(dot_data)
graph
```


X[2] <= 2.45
gini = 0.667
samples = 150
value = [50, 50, 50]
True
False
gini = 0.0
samples = 50
value = [50, 0, 0]
X[3] <= 1.75
gini = 0.5
samples = 100
value = [0, 50, 50]
X[2] <= 4.95
gini = 0.168
samples = 54
value = [0, 49, 5]
X[2] <= 4.85
gini = 0.043
samples = 46
value = [0, 1, 45]
X[3] <= 1.65
gini = 0.041
samples = 48
value = [0, 47, 1]
X[3] <= 1.55
gini = 0.444
samples = 6
value = [0, 2, 4]
X[0] <= 5.95
gini = 0.444
samples = 3
value = [0, 1, 2]
gini = 0.0
samples = 43
value = [0, 0, 43]
gini = 0.0
samples = 47
value = [0, 47, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]
gini = 0.0
samples = 3
value = [0, 0, 3]
X[2] <= 5.45
gini = 0.444
samples = 3
value = [0, 2, 1]
gini = 0.0
samples = 1
value = [0, 1, 0]
gini = 0.0
samples = 2
value = [0, 0, 2]
gini = 0.0
samples = 2
value = [0, 2, 0]
gini = 0.0
samples = 1
value = [0, 0, 1]

决策树

- 树也已可以用文本形式表示：

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.tree import export_text
>>> iris = load_iris()
>>> decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
>>> decision_tree = decision_tree.fit(iris.data, iris.target)
>>> r = export_text(decision_tree, feature_names=iris['feature_names'])
>>> print(r)
|--- petal width (cm) <= 0.80
|   |--- class: 0
|--- petal width (cm) >  0.80
|   |--- petal width (cm) <= 1.75
|   |   |--- class: 1
|   |--- petal width (cm) >  1.75
|   |   |--- class: 2
```

决策树

- 回归：DecisionTreeRegressor()
- 基本用法：

```
>>> 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])
```

决策树

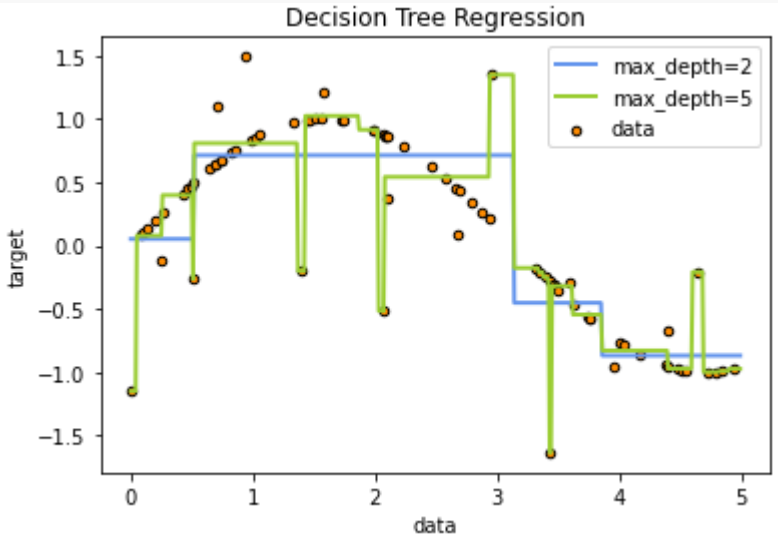
- 回归：DecisionTreeRegressor()

```
import numpy as np
from sklearn.tree import DecisionTreeRegressor
import matplotlib.pyplot as plt

# Create a random dataset
rng = np.random.RandomState(1)
X = np.sort(5 * rng.rand(80, 1), axis=0)
y = np.sin(X).ravel()
y[::5] += 3 * (0.5 - rng.rand(16))

# Fit regression model
regr_1 = DecisionTreeRegressor(max_depth=2)
regr_2 = DecisionTreeRegressor(max_depth=5)
regr_1.fit(X, y)
regr_2.fit(X, y)

# Predict
X_test = np.arange(0.0, 5.0, 0.01)[:, np.newaxis]
y_1 = regr_1.predict(X_test)
y_2 = regr_2.predict(X_test)
```

决策树

- 回归：DecisionTreeRegressor()

```
# Plot the results
plt.figure()
plt.scatter(X, y, s=20, edgecolor="black",
            c="darkorange", label="data")
plt.plot(X_test, y_1, color="cornflowerblue",
         label="max_depth=2", linewidth=2)
plt.plot(X_test, y_2, color="yellowgreen", label="max_depth=5", linewidth=2)
plt.xlabel("data")
plt.ylabel("target")
plt.title("Decision Tree Regression")
plt.legend()
plt.show()
```



朴素贝叶斯

- sklearn.naive_bayes.GaussianNB()

```
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
iris = datasets.load_iris()
gnb = GaussianNB()

# 训练模型
gnb.fit(iris.data, iris.target)

# 进行预测
y_pred = gnb.predict(iris.data)

y_pred
```

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

朴素贝叶斯

- sklearn.naive_bayes.GaussianNB()
- 高斯朴素贝叶斯：特征似然值用高斯函数估计

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

```
>>> from sklearn.datasets import load_iris
>>> from sklearn.model_selection import train_test_split
>>> from sklearn.naive_bayes import GaussianNB
>>> X, y = load_iris(return_X_y=True)
>>> X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.5, random_state=0)
>>> gnb = GaussianNB()
>>> y_pred = gnb.fit(X_train, y_train).predict(X_test)
>>> print("Number of mislabeled points out of a total %d points : %d"
...       % (X_test.shape[0], (y_test != y_pred).sum()))
Number of mislabeled points out of a total 75 points : 4
```

目录

- 机器学习的概念
  - 概念
  - 经典问题
    - 分类
    - 回归
    - 聚类
- 利用scikit-learn进行机器学习
  - 预处理
  - 分类
  - 回归
  - 聚类

聚类

- sklearn的聚类算法在sklearn.cluster模块下
- 每个聚类算法(clustering algorithm)都有两个变体:
- 一个是类（class）, 它实现了 fit 方法来学习训练数据的簇（cluster）;
- 另一个函数（function），当给定训练数据，返回与不同簇对应的整数标签数组（array）。
- 对于类来说，训练数据上的标签可以在 labels_ 属性中找到。

K-means

- sklearn.cluster.Kmeans()

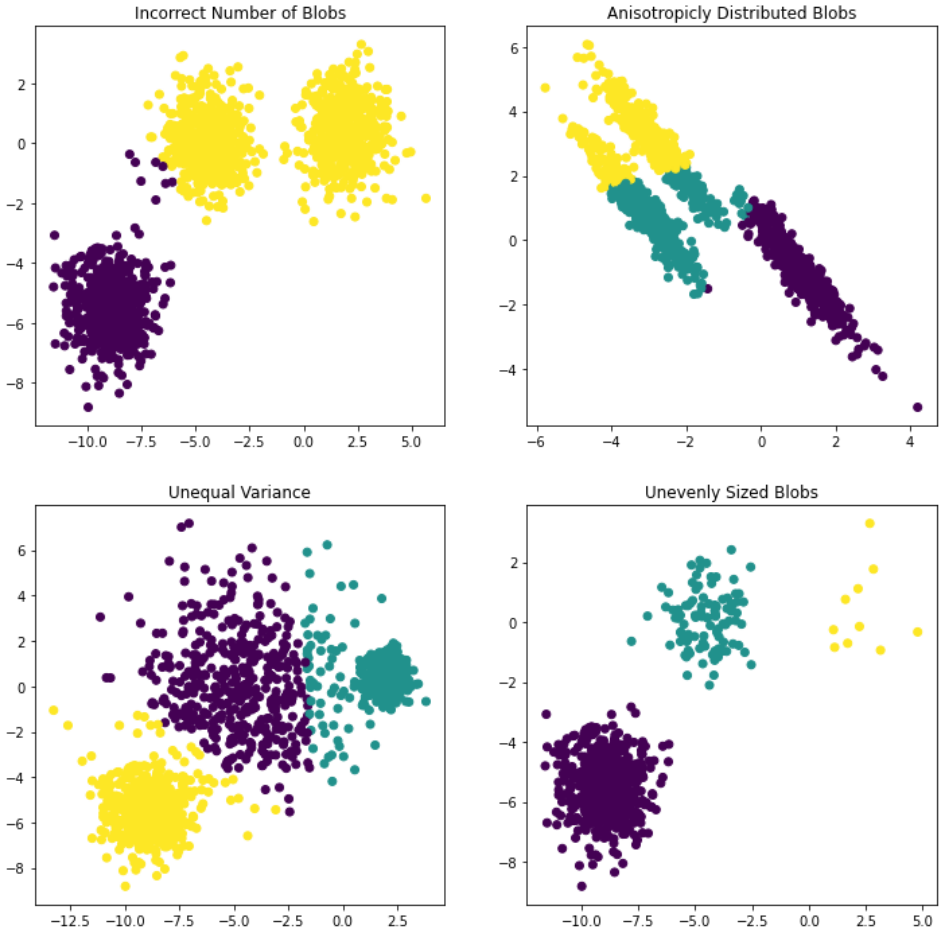
make_blobs：生成各向异性的高斯分布数据

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

from sklearn.cluster import KMeans
from sklearn.datasets import make_blobs

plt.figure(figsize=(12, 12))

n_samples = 1500
random_state = 170
X, y = make_blobs(n_samples=n_samples, random_state=random_state)

# Incorrect number of clusters
y_pred = KMeans(n_clusters=2, random_state=random_state).fit_predict(X)

plt.subplot(221)
plt.scatter(X[:, 0], X[:, 1], c=y_pred)
plt.title("Incorrect Number of Blobs")

# Anisotropicly distributed data
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X_aniso = np.dot(X, transformation)
y_pred = KMeans(n_clusters=3, random_state=random_state).fit_predict(X_aniso)

plt.subplot(222)
plt.scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)
plt.title("Anisotropicly Distributed Blobs")
```

K-means

```
# Anisotropicly distributed data
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
X_aniso = np.dot(X, transformation)
y_pred = KMeans(n_clusters=3, random_state=random_state).fit_predict(X_aniso)

plt.subplot(222)
plt.scatter(X_aniso[:, 0], X_aniso[:, 1], c=y_pred)
plt.title("Anisotropicly Distributed Blobs")

# Different variance
X_varied, y_varied = make_blobs(n_samples=n_samples,
                                cluster_std=[1.0, 2.5, 0.5],
                                random_state=random_state)
y_pred = KMeans(n_clusters=3, random_state=random_state).fit_predict(X_varied)

plt.subplot(223)
plt.scatter(X_varied[:, 0], X_varied[:, 1], c=y_pred)
plt.title("Unequal Variance")

# Unevenly sized blobs
X_filtered = np.vstack((X[y == 0][:500], X[y == 1][:100], X[y == 2][:10]))
y_pred = KMeans(n_clusters=3,
                random_state=random_state).fit_predict(X_filtered)

plt.subplot(224)
plt.scatter(X_filtered[:, 0], X_filtered[:, 1], c=y_pred)
plt.title("Unevenly Sized Blobs")

plt.show()
```

K-means


Incorrect Number of Blobs
Anisotropicly Distributed Blobs
Unequal Variance
Unevenly Sized Blobs

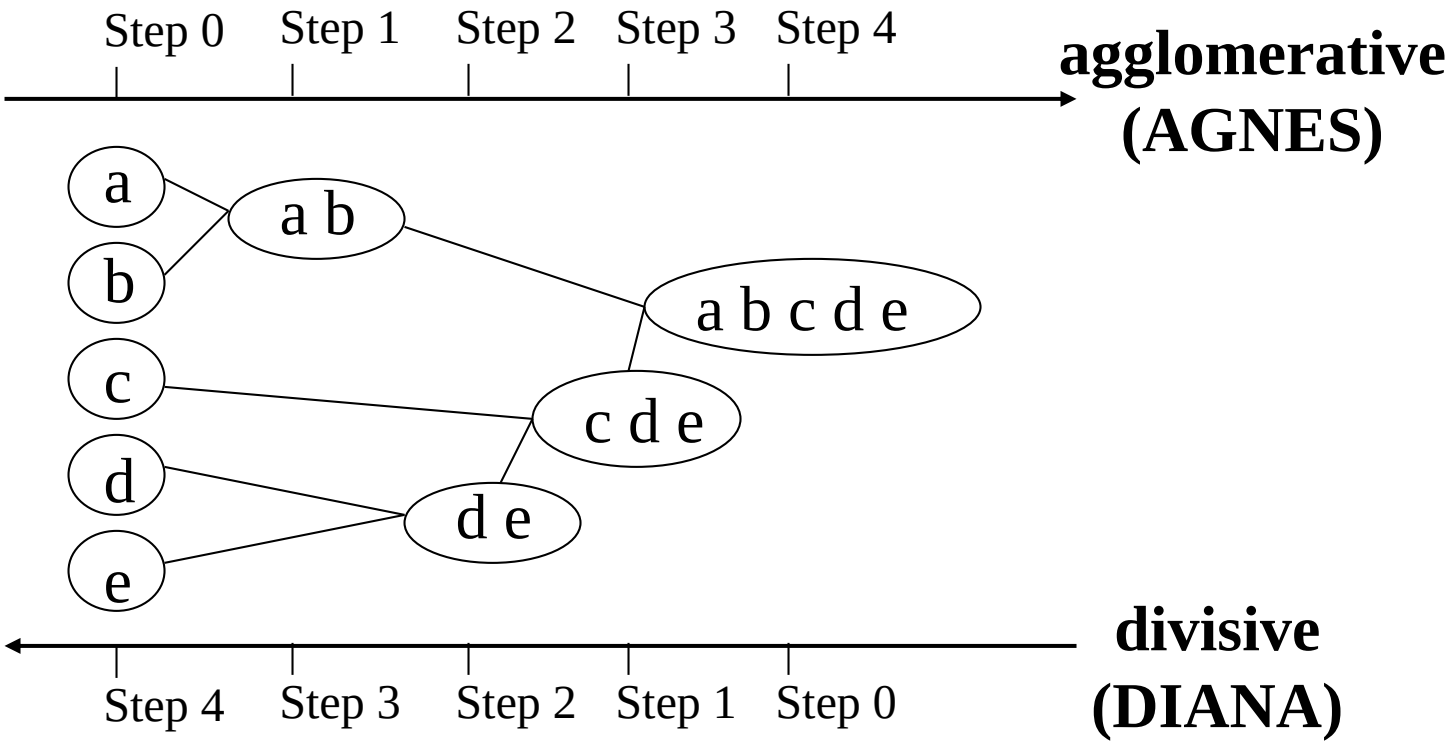
层次聚类

```
import numpy as np

from matplotlib import pyplot as plt
from scipy.cluster.hierarchy import dendrogram
from sklearn.datasets import load_iris
from sklearn.cluster import AgglomerativeClustering


def plot_dendrogram(model, **kwargs):
    # Create linkage matrix and then plot the dendrogram

    # create the counts of samples under each node
    counts = np.zeros(model.children_.shape[0])
    n_samples = len(model.labels_)
    for i, merge in enumerate(model.children_):
        current_count = 0
        for child_idx in merge:
            if child_idx < n_samples:
                current_count += 1  # leaf node
            else:
                current_count += counts[child_idx - n_samples]
        counts[i] = current_count

    linkage_matrix = np.column_stack([model.children_, model.distances_,
                                      counts]).astype(float)

    # Plot the corresponding dendrogram
    dendrogram(linkage_matrix, **kwargs)
```

层次聚类

- 基本思想：

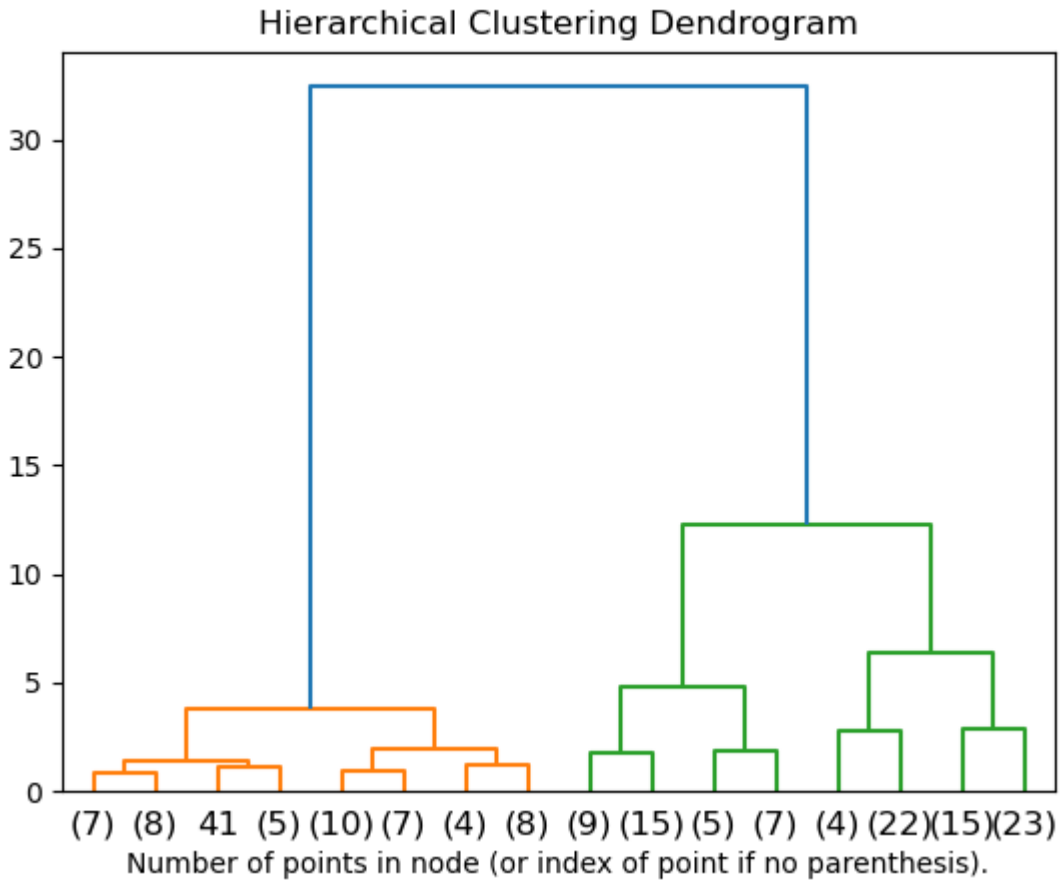
  根据一定准则，对给定待聚类数据集合进行层次化分解。



层次聚类

```
iris = load_iris()
X = iris.data

# setting distance_threshold=0 ensures we compute the full tree.
model = AgglomerativeClustering(distance_threshold=0, n_clusters=None)

model = model.fit(X)
plt.title('Hierarchical Clustering Dendrogram')
# plot the top three levels of the dendrogram
plot_dendrogram(model, truncate_mode='level', p=3)
plt.xlabel("Number of points in node (or index of point if no parenthesis).")
plt.show()
```

层次聚类


Hierarchical Clustering Dendrogram
30
25
20
15
10
5
0
(7) (8) 41 (5) (10) (7) (4) (8) (9) (15) (5) (7) (4) (22) (15) (23)
Number of points in node (or index of point if no parenthesis).

层次聚类

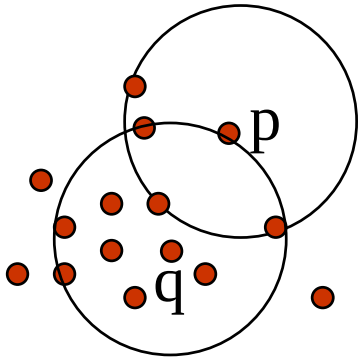
- sklearn.cluster.AgglomerativeClustering()
- 自下向上，依次聚合
- 聚合方式：
- **Ward**： 最小化所有聚类内的平方差总和。这是一种方差最小化(variance-minimizing)的优化方向， 这是与k-means 的目标函数相似的优化方法，但是用 凝聚分层（agglomerative hierarchical）的方法处理。
- **Maximum** 或 **complete linkage**： 最小化成对聚类间最远样本距离。
- **Average linkage**： 最小化成对聚类间平均样本距离值。
- **Single linkage**： 最小化成对聚类间最近样本距离值。

DBSCAN

- 是一种基于高密度连接区域的密度聚类算法。
- 参数：
  - Eps: 邻居最大半径；
  - MinPts: 核心点最小邻居阈值；
  - $N_{Eps}(p)$: {q belongs to D | dist(p,q) ≤ Eps}

- 直接密度可达(Directly density-reachable)
  - 称从点q到点p关于Eps和minPts密度可达，若满足以下条件：

1. *p* belongs to $N_{Eps}(q)$
2. core point condition: |NEps (q)| ≥ MinPts



MinPts = 5

Eps = 1 cm

DBSCAN

- sklearn.cluster.DBSCAN()

```
import numpy as np

from sklearn.cluster import DBSCAN
from sklearn import metrics
from sklearn.datasets import make_blobs
from sklearn.preprocessing import StandardScaler


# #############################################################################
# Generate sample data
centers = [[1, 1], [-1, -1], [1, -1]]
X, labels_true = make_blobs(n_samples=750, centers=centers, cluster_std=0.4,
                            random_state=0)

X = StandardScaler().fit_transform(X)
```

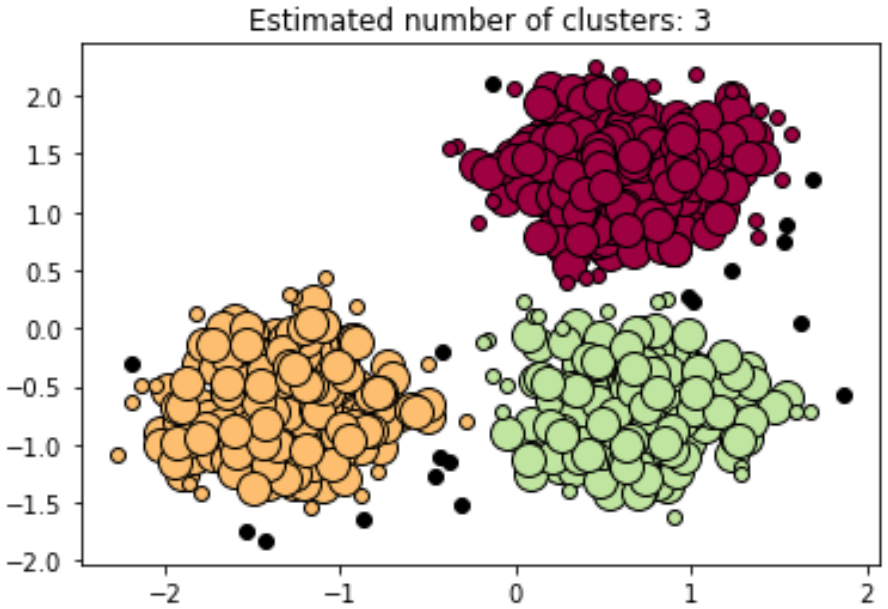
DBSCAN

```
# #############################################################################
# Compute DBSCAN
db = DBSCAN(eps=0.3, min_samples=10).fit(X)
core_samples_mask = np.zeros_like(db.labels_, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
labels = db.labels_

# Number of clusters in labels, ignoring noise if present.
n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
n_noise_ = list(labels).count(-1)

print('Estimated number of clusters: %d' % n_clusters_)
print('Estimated number of noise points: %d' % n_noise_)
print("Homogeneity: %0.3f" % metrics.homogeneity_score(labels_true, labels))
print("Completeness: %0.3f" % metrics.completeness_score(labels_true, labels))
print("V-measure: %0.3f" % metrics.v_measure_score(labels_true, labels))
print("Adjusted Rand Index: %0.3f"
      % metrics.adjusted_rand_score(labels_true, labels))
print("Adjusted Mutual Information: %0.3f"
      % metrics.adjusted_mutual_info_score(labels_true, labels))
print("Silhouette Coefficient: %0.3f"
      % metrics.silhouette_score(X, labels))
```

DBSCAN

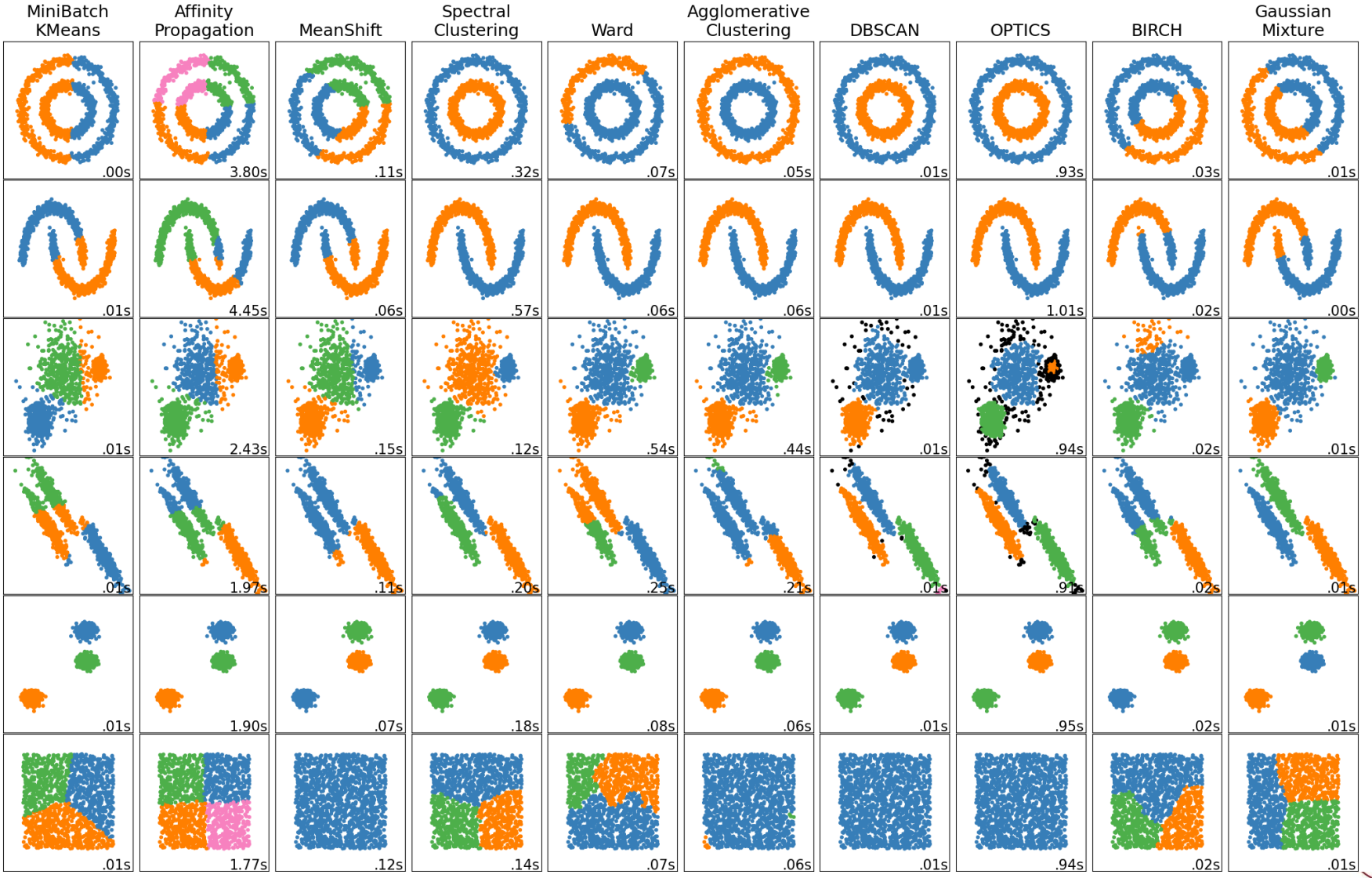
```
# ##############################################################################
# Plot result
import matplotlib.pyplot as plt

# Black removed and is used for noise instead.
unique_labels = set(labels)
colors = [plt.cm.Spectral(each)
          for each in np.linspace(0, 1, len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
        # Black used for noise.
        col = [0, 0, 0, 1]

    class_member_mask = (labels == k)

    xy = X[class_member_mask & core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=14)

    xy = X[class_member_mask & ~core_samples_mask]
    plt.plot(xy[:, 0], xy[:, 1], 'o', markerfacecolor=tuple(col),
             markeredgecolor='k', markersize=6)

plt.title('Estimated number of clusters: %d' % n_clusters_)
plt.show()
```

DBSCAN



- Estimated number of clusters: 3
- Estimated number of noise points: 18
- Homogeneity: 0.953
- Completeness: 0.883
- V-measure: 0.917
- Adjusted Rand Index: 0.952
- Adjusted Mutual Information: 0.916
- Silhouette Coefficient: 0.626

聚类算法比较


MiniBatch KMeans
Affinity Propagation
MeanShift
Spectral Clustering
Ward
Agglomerative Clustering
DBSCAN
OPTICS
BIRCH
Gaussian Mixture
.00s
3.80s
.11s
.32s
.07s
.05s
.01s
.93s
.03s
.01s
.01s
4.45s
.06s
.57s
.06s
.06s
.01s
1.01s
.02s
.00s
.01s
2.43s
.15s
.12s
.54s
.44s
.01s
.94s
.02s
.01s
.01s
1.97s
.11s
.20s
.25s
.21s
.01s
.91s
.02s
.01s
.01s
1.90s
.07s
.18s
.08s
.06s
.01s
.95s
.02s
.01s
.01s
1.77s
.12s
.14s
.07s
.06s
.01s
.94s
.02s
.01s

模型评价

- 分类模型

Scoring	Function	Comment
Classification		
'accuracy'	`metrics.accuracy_score`	
'balanced_accuracy'	`metrics.balanced_accuracy_score`	
'top_k_accuracy'	`metrics.top_k_accuracy_score`	
'average_precision'	`metrics.average_precision_score`	
'neg_brier_score'	`metrics.brier_score_loss`	
'f1'	`metrics.f1_score`	for binary targets
'f1_micro'	`metrics.f1_score`	micro-averaged
'f1_macro'	`metrics.f1_score`	macro-averaged
'f1_weighted'	`metrics.f1_score`	weighted average
'f1_samples'	`metrics.f1_score`	by multilabel sample
'neg_log_loss'	`metrics.log_loss`	requires `predict_proba` support
'precision' etc.	`metrics.precision_score`	suffixes apply as with 'f1'
'recall' etc.	`metrics.recall_score`	suffixes apply as with 'f1'
'jaccard' etc.	`metrics.jaccard_score`	suffixes apply as with 'f1'
'roc_auc'	`metrics.roc_auc_score`	
'roc_auc_ovr'	`metrics.roc_auc_score`	
'roc_auc_ovo'	`metrics.roc_auc_score`	
'roc_auc_ovr_weighted'	`metrics.roc_auc_score`	
'roc_auc_ovo_weighted'	`metrics.roc_auc_score`	

模型评价

- 回归模型

Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance
'neg_mean_absolute_percentage_error'	metrics.mean_absolute_percentage_error

模型评价

- 聚类模型

Clustering	
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score
'adjusted_rand_score'	metrics.adjusted_rand_score
'completeness_score'	metrics.completeness_score
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score
'homogeneity_score'	metrics.homogeneity_score
'mutual_info_score'	metrics.mutual_info_score
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score
'rand_score'	metrics.rand_score
'v_measure_score'	metrics.v_measure_score

模型评价

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

from sklearn import svm, datasets
from sklearn import metrics

# 准备数据
iris = datasets.load_iris()
X = iris.data
y = iris.target

# 取出第一类和第二类
X, y = X[y != 2], y[y != 2]
n_samples, n_features = X.shape
n_samples

# 加入噪声特征
random_state = np.random.RandomState(0)
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]
```

模型评价

```
# 随机选取80个样本作为训练集，其余作为测试集
t = np.array(range(100))
np.random.shuffle(t)
train_idx = t >= 20

train_X = X[train_idx,:]
train_y = y[train_idx]
text_X = X[~train_idx,:]
test_y= y[~train_idx]

# 训练模型
svc_clf = svm.SVC(kernel='linear', probability=True,
                  random_state=random_state)
svc_clf = svc_clf.fit(train_X, train_y)
```

```
# 准确率 sklearn.metrics.accuracy_score()
y_pre = svc_clf.predict(text_X)
metrics.accuracy_score(test_y,y_pre)
```

```
0.7
```

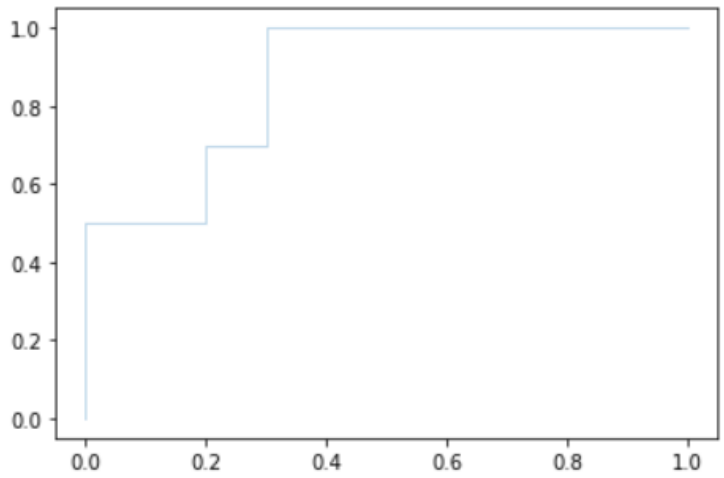
```
# 混淆矩阵
cnf_matrix = metrics.confusion_matrix(test_y, y_pre)
print(cnf_matrix)
```

```
[[8 2]
 [4 6]]
```

模型评价

```
# ROC曲线 sklearn.metrics.roc_curve()
from sklearn.metrics import roc_curve, auc
tprs = []
aucs = []
mean_fpr = np.linspace(0, 1, 100)

probas_ = svc_clf.predict_proba(text_X)
fpr, tpr, thresholds = roc_curve(test_y, probas_[:, 1])
tprs.append(np.interp(mean_fpr, fpr, tpr))
tprs[-1][0] = 0.0
roc_auc = auc(fpr, tpr)
aucs.append(roc_auc)
plt.plot(fpr, tpr, lw=1, alpha=0.3,
         label='ROC Curve (AUC = %0.2f)' % roc_auc)
```

```
[<matplotlib.lines.Line2D at 0x244d3aab100>]
```



模型列表

1. Supervised learning

- 1.1. Linear Models
- 1.2. Linear and Quadratic Discriminant Analysis
- 1.3. Kernel ridge regression
- 1.4. Support Vector Machines
- 1.5. Stochastic Gradient Descent
- 1.6. Nearest Neighbors
- 1.7. Gaussian Processes
- 1.8. Cross decomposition
- 1.9. Naive Bayes
- 1.10. Decision Trees
- 1.11. Ensemble methods
- 1.12. Multiclass and multioutput algorithms
- 1.13. Feature selection
- 1.14. Semi-supervised learning
- 1.15. Isotonic regression
- 1.16. Probability calibration
- 1.17. Neural network models (supervised)

2. Unsupervised learning

- 2.1. Gaussian mixture models
- 2.2. Manifold learning
- 2.3. Clustering
- 2.4. Biclustering
- 2.5. Decomposing signals in components (matrix factorization problems)
- 2.6. Covariance estimation
- 2.7. Novelty and Outlier Detection
- 2.8. Density Estimation
- 2.9. Neural network models (unsupervised)

https://scikit-learn.org/stable/user_guide.html

关联规则挖掘

- 在海量交易数据中找出关联规则

关联规则挖掘

• **交易*T***	• 是指由若干个商品所构成的集合$T = \{i_1, i_2, \ldots, i_t\}$ • 注意T是所有商品的全集I的非空子集
• **数据库*D***	• 记录下的所有交易所构成的集合
• **项集**	• 由若干商品所构成的集合
• **k-项集**	• 包含k个元素的项集

关联规则的形式化定义：

$$X \Rightarrow Y, \text{ where } X \subset I, \quad Y \subset I \text{ and } X \cap Y = \emptyset$$

关联规则挖掘

交易	项集
T1	Bread, Jelly, Peanut, Butter
T2	Bread, Butter
T3	Bread, Jelly
T4	Bread, Milk, Butter
T5	Chips, Milk
T6	Bread, Chips
T7	Bread, Milk
T8	Chips, Jelly

我们的目标：找到像Bread ⇒ Butter这样的关联规则

支持度

- 一个项集（或元素）X的**支持度**是指：数据库中包含该项集（或元素）的交易所占的比例，记作Support。

$$Support(X) = \frac{\#(X)}{n}$$

项集	支持度
Bread, Butter	3/8
……	
Bread, Butter, Chips	0/8
……	
Bread, Butter, Chips, Jelly	0/8
……	
Bread, Butter, Chips, Jelly, Milk, Peanut	0/8

支持度

- 一个项集（或元素）X的**支持度**是指：数据库中包含该项集（或元素）的交易所占的比例，记作Support。

$$Support(X) = \frac{\#(X)}{n}$$

交易	项集
T1	Bread, Jelly, Peanut, Butter
T2	Bread, Butter
T3	Bread, Jelly
T4	Bread, Milk, Butter
T5	Chips, Milk
T6	Bread, Chips
T7	Bread, Milk
T8	Chips, Jelly

项集	支持度
Bread	6/8
Butter	3/8
Chips	2/8
Jelly	3/8
Milk	3/8
Peanut	1/8

支持度与置信度

- **关联规则$X \Rightarrow Y$的支持度**：同时包含X和Y的交易所占的比例，记作Support。（度量关联规则出现的频率）

$$Support(X \Rightarrow Y) = \frac{\#(X \cup Y)}{n}$$

- **关联规则$X \Rightarrow Y$的置信度**：同时包含X和Y的交易个数与包含X的交易个数的比值，记作Confidence。（度量关联规则的强度）

$$Confidence(X \Rightarrow Y) = \frac{\#(X \cup Y)}{\#(X)}$$

- 置信度可以表示为支持度的比值：

$$Confidence(X \Rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$$

- 可解释为**条件概率**：$P(Y|X)$

$$Support(X) = \frac{\#(X)}{n}$$

算例

Database TDB

Tid	Items
10	A, C, D
20	B, C, E
30	A, B, C, E
40	B, E

$Sup_{min} = 2$

C_1 — 1st scan →

Itemset	sup
{A}	2
{B}	3
{C}	3
{D}	1
{E}	3

L_1 →

Itemset	sup
{A}	2
{B}	3
{C}	3
{E}	3

C_2

Itemset
{A, B}
{A, C}
{A, E}
{B, C}
{B, E}
{C, E}

2nd scan →

C_2

Itemset	sup
{A, B}	1
{A, C}	2
{A, E}	1
{B, C}	2
{B, E}	3
{C, E}	2

←

L_2

Itemset	sup
{A, C}	2
{B, C}	2
{B, E}	3
{C, E}	2

C_3

Itemset
{B, C, E}

3rd scan →

L_3

Itemset	sup
{B, C, E}	2

Apriori工具包

- 安装工具包

pip install efficient-apriori

- 测试：

In [1]:
```
from efficient_apriori import apriori
transactions = [('eggs', 'bacon', 'soup'),
                ('eggs', 'bacon', 'apple'),
                ('soup', 'bacon', 'banana')]
itemsets, rules = apriori(transactions, min_support=0.5,  min_confidence=1)
print(rules)  # [{eggs} -> {bacon}, {soup} -> {bacon}]
```

```
[{eggs} -> {bacon}, {soup} -> {bacon}]
```

Apriori工具包

- 查看、筛选结果

In [2]:
```
print(itemsets)
```

```
{1: {('eggs',): 2, ('bacon',): 3, ('soup',): 2}, 2: {('bacon', 'eggs'): 2, ('bacon', 'soup'): 2}}
```

In [3]:
```
# Print out every rule with 2 items on the left hand side,
# 1 item on the right hand side, sorted by lift
rules_rhs = filter(lambda rule: len(rule.lhs) == 1 and len(rule.rhs) == 1, rules)
list(rules_rhs)
```

Out[3]:
```
[{eggs} -> {bacon}, {soup} -> {bacon}]
```

In [4]:
```
rules_rhs = filter(lambda rule: len(rule.lhs) == 1 and len(rule.rhs) == 1, rules)
result = sorted(rules_rhs, key=lambda rule: rule.lift)
print(list(result))
```

```
[{eggs} -> {bacon}, {soup} -> {bacon}]
```

In [5]:
```
for rule in result:
    print(rule) # Prints the rule and its confidence, support, lift, ...
```

```
{eggs} -> {bacon} (conf: 1.000, supp: 0.667, lift: 1.000, conv: 0.000)
{soup} -> {bacon} (conf: 1.000, supp: 0.667, lift: 1.000, conv: 0.000)
```

Apriori工具包

- 测试：

In [6]:
```
transactions = [('A', 'C', 'D'),
                ('B', 'C', 'E'),
                ('A', 'B', 'C', 'E'),
                ('B', 'E')]
```

In [7]:
```
itemsets, rules = apriori(transactions, min_support=0.5,  min_confidence=1)
itemsets
```

Out[7]:
```
{1: {('A',): 2, ('C',): 3, ('B',): 3, ('E',): 3},
 2: {('A', 'C'): 2, ('B', 'C'): 2, ('B', 'E'): 3, ('C', 'E'): 2},
 3: {('B', 'C', 'E'): 2}}
```

In [8]:
```
rules
```

Out[8]:
```
[{A} -> {C}, {E} -> {B}, {B} -> {E}, {C, E} -> {B}, {B, C} -> {E}]
```

处理大规模数据

- 如果要分析的数据较大，无法直接导入内存，可以传递一个返回生成器的函数，而不是交易的列表

In [11]:
```
import pandas as pd
def data_generator(filename):
    """
    Data generator, needs to return a generator to be called several times.
    """
    def data_gen():
        with open(filename) as file:
            for line in file:
                yield tuple(k.strip() for k in line.split(','))
    return data_gen
```

In [12]:
```
# file_path = "https://github.com/seratch/apriori.js/blob/master/dataset.csv"
transactions = data_generator("dataset.csv")
itemsets, rules = apriori(transactions, min_support=0.5,  min_confidence=1)
itemsets
```

Out[12]:
```
{1: {('A',): 2, ('C',): 3, ('B',): 3, ('E',): 3},
 2: {('A', 'C'): 2, ('B', 'C'): 2, ('B', 'E'): 3, ('C', 'E'): 2},
 3: {('B', 'C', 'E'): 2}}
```

In [13]:
```
rules
```

Out[13]:
```
[{A} -> {C}, {E} -> {B}, {B} -> {E}, {C, E} -> {B}, {B, C} -> {E}]
```

练习

1. 对census_income数据集进行预处理（包括但不限于填充缺失值、数据标准化、标称数据编码）。以最后一列“>50K”为正类，“<=50K”为负类，选择至少4中分类模型进行建模，并进行结果评估和比较（可利用准确率、精准率、召回率、F值、ROC、AUC等指标）。

http://archive.ics.uci.edu/ml/datasets/Census+Income

练习

2. 对预处理后的census_income数据集中除最后一列的所有特征选择不同模型进行聚类，并进行结果比较和解释。

3. 利用data_generater读取“store_data.csv”并进行关联规则挖掘，找出支持度不小于0.0045，置信度不小于0.2的关联规则，并对结果进行解释。

谢谢！