第6讲　k-近邻算法

- kNN算法
- sklearn.neighbors
- 数据预处理

kNN算法

- **相似度**：向量间距离度量(**Distance Measure**)的一种常用方式

  两个向量的距离函数dist(x_i, x_j)需要满足一些基本性质

非负性(Positivity Separation)： **dist(x_i, x_j) >= 0**

同一性(Constancy of Self-Similarity)：**dist(x_i, x_j) = 0, iff x_i= x_j**

对称性(Symmetry)： **dist(x_i, x_j) = dist(x_j, x_i)**

三角不等式(triangular inequality)： **dist(x_i, x_j) <= dist(x_i, x_k) <= dist(x_k, x_j)**

kNN算法

- 向量$x_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ 与$x_j = (x_{j1}, x_{j2}, \ldots, x_{jd})$之间距离
  - 欧氏距离(Euclidean Distance)：

$$dist(x_i, x_j) = \|x_i - x_j\|_2 = \sqrt{\sum_{k=1}^{d} |x_{ik} - x_{jk}|^2}$$

  - 曼哈顿距离(Manhattan Distance)：

$$dist(x_i, x_j) = |x_i - x_j|_1 = \sum_{k=1}^{d} |x_{ik} - x_{jk}|$$

kNN算法

- 向量x$_i$=($x_{i1}, x_{i2}, \ldots, x_{id}$) 与x$_j$=($x_{j1}, x_{j2}, \ldots, x_{jd}$)之间距离
  - 闵可夫斯基距离(Minkowski Distance)：

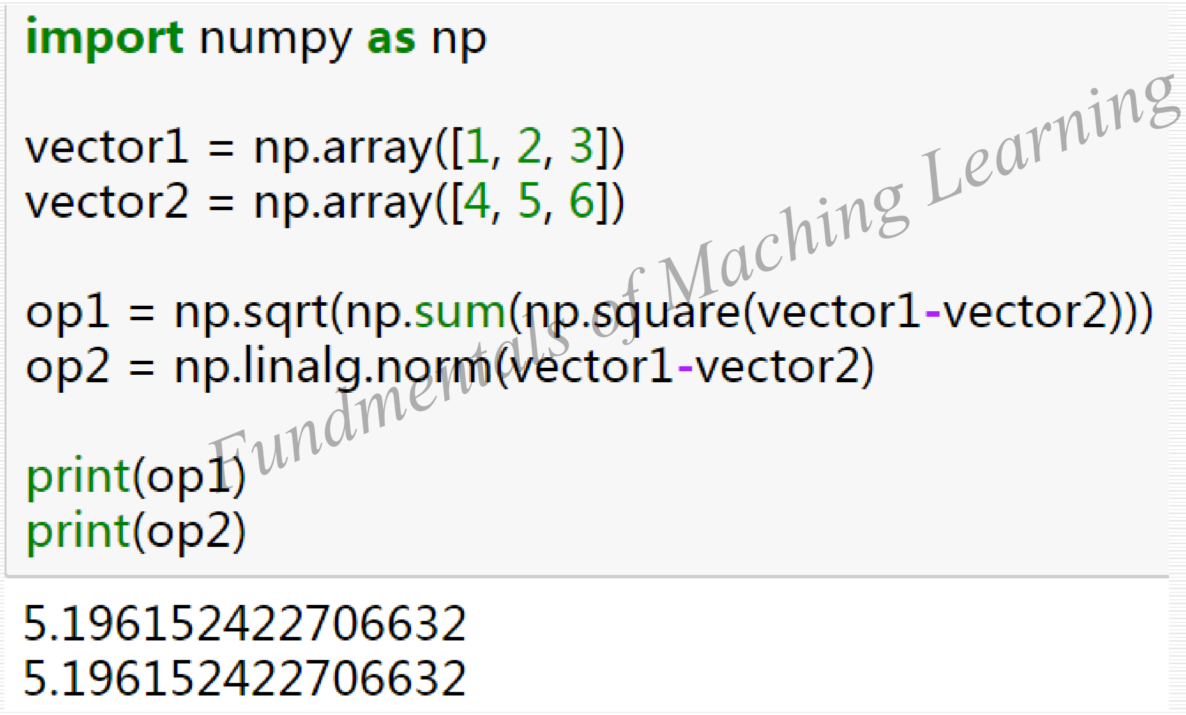
$$dist(x_i, x_j) = \sqrt[r]{\sum_{k=1}^{d} \left| x_{ik} - x_{jk} \right|^r}$$

  - 切比雪夫距离(Chebyshev Distance)： $r = \infty$

$$dist(x_i, x_j) = \max \left| x_{ik} - x_{jk} \right|$$

kNN算法

◆ 示例：计算欧氏距离

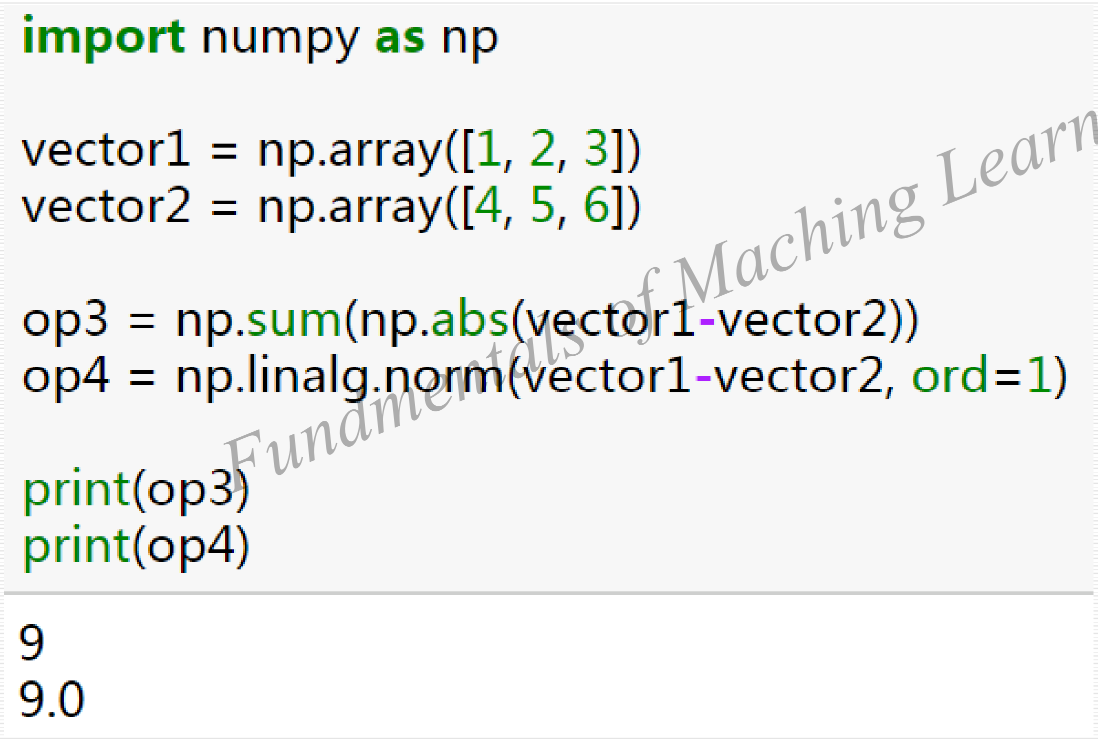


```python
import numpy as np

vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 5, 6])

op1 = np.sqrt(np.sum(np.square(vector1-vector2)))
op2 = np.linalg.norm(vector1-vector2)

print(op1)
print(op2)
```

```
5.196152422706632
5.196152422706632
```

kNN算法

- 示例：计算曼哈顿距离

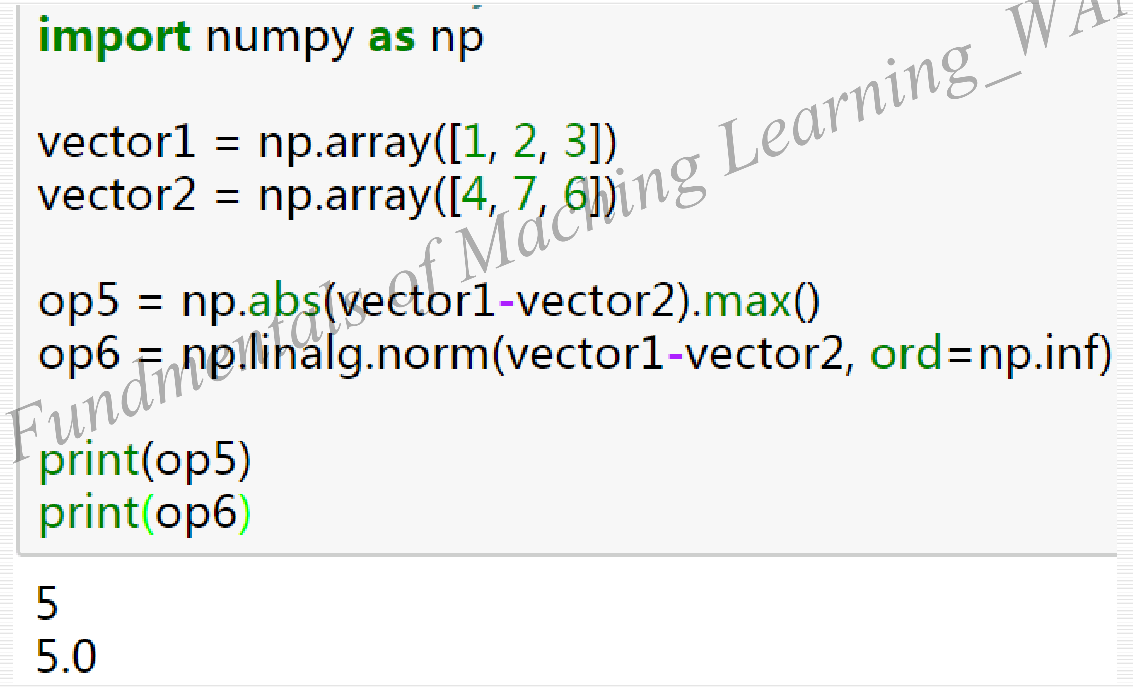


```
import numpy as np

vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 5, 6])

op3 = np.sum(np.abs(vector1-vector2))
op4 = np.linalg.norm(vector1-vector2, ord=1)

print(op3)
print(op4)
```

```
9
9.0
```

kNN算法

◆ 示例：切比雪夫距离(Chebyshev Distance)

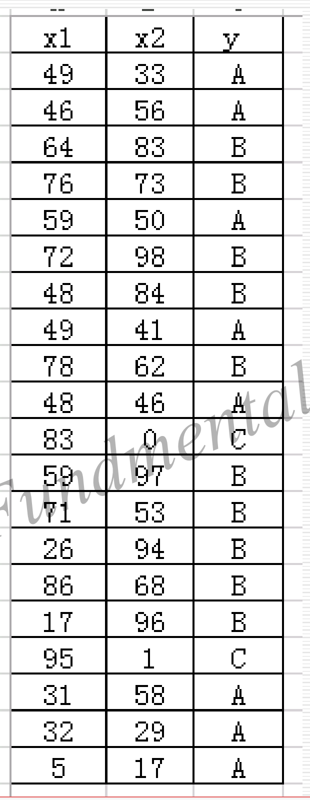


```
import numpy as np

vector1 = np.array([1, 2, 3])
vector2 = np.array([4, 7, 6])

op5 = np.abs(vector1-vector2).max()
op6 = np.linalg.norm(vector1-vector2, ord=np.inf)

print(op5)
print(op6)
```

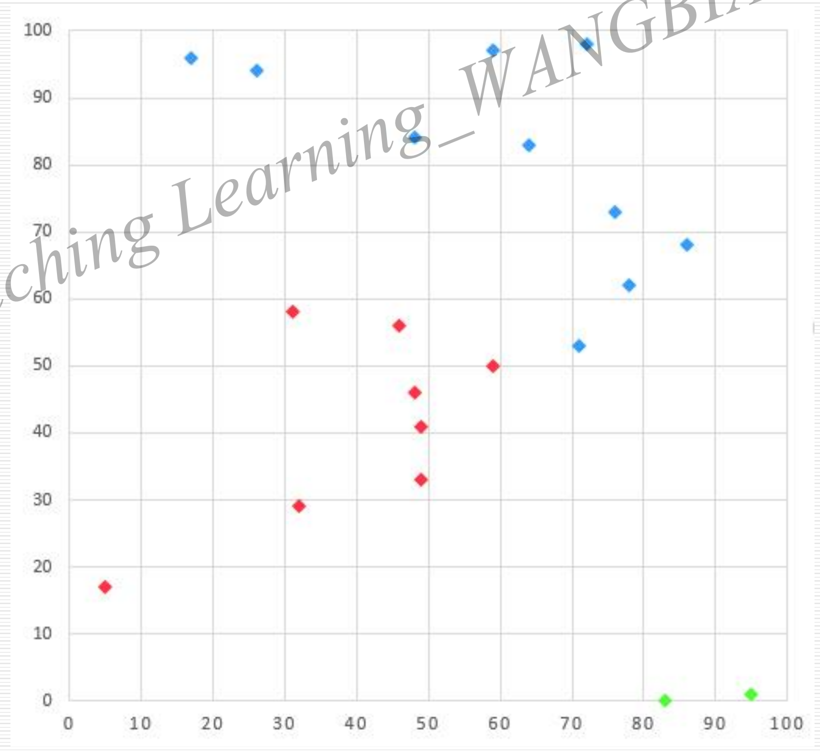
```
5
5.0
```

kNN算法

◆ 示例1：已知20个数据样本，其分类结果如下图(三个类别)



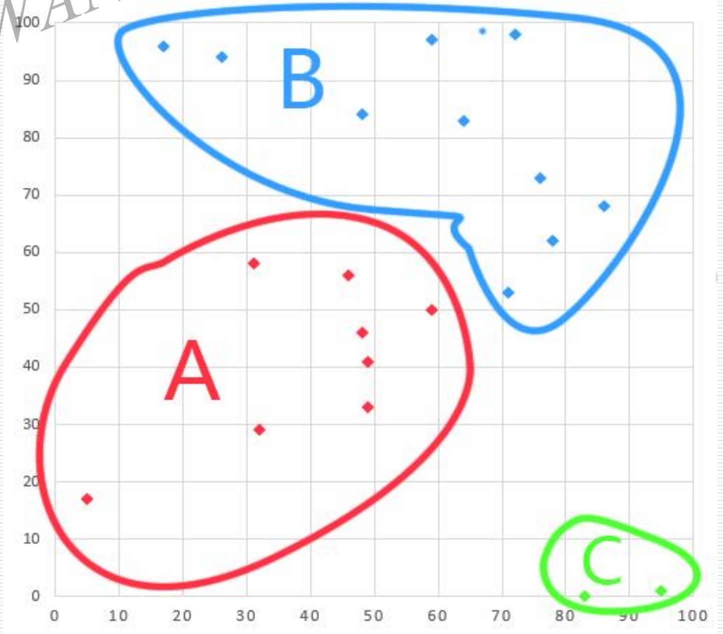
x1	x2	y
49	33	A
46	56	A
64	83	B
76	73	B
59	50	A
72	98	B
48	84	B
49	41	A
78	62	B
48	46	A
83	0	C
59	97	B
71	53	B
26	94	B
86	68	B
17	96	B
95	1	C
31	58	A
32	29	A
5	17	A



kNN算法

◆ **示例1：已知20个数据样本，其分类结果如下图(三个类别)**

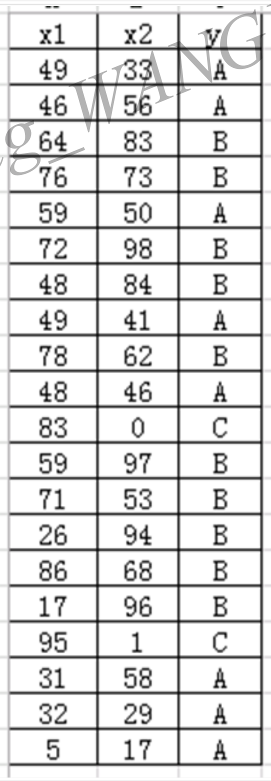
- 样本数据集分为A、B、C三个类别
- 问题：通过20个已知类别的样本，对一个新数据（x=60, y=64）进行分类



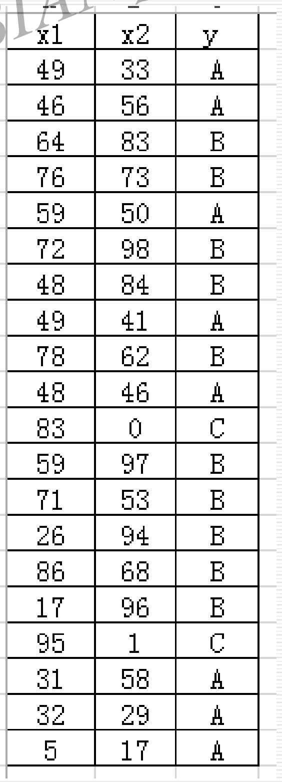
kNN算法

◆ **示例1：已知20个数据样本，其分类结果如下图(三个类别)**

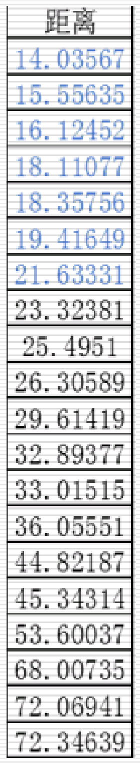
- 先计算新数据与各样本数据间距离
- 按距离递增次序排序
- 选取与当前点距离最小的*k*个点，k = 7
- 确定前*k*个点所属类别的出现概率



x1	x2	y
49	33	A
46	56	A
64	83	B
76	73	B
59	50	A
72	98	B
48	84	B
49	41	A
78	62	B
48	46	A
83	0	C
59	97	B
71	53	B
26	94	B
86	68	B
17	96	B
95	1	C
31	58	A
32	29	A
5	17	A



x1	x2	y
49	33	A
46	56	A
64	83	B
76	73	B
59	50	A
72	98	B
48	84	B
49	41	A
78	62	B
48	46	A
83	0	C
59	97	B
71	53	B
26	94	B
86	68	B
17	96	B
95	1	C
31	58	A
32	29	A
5	17	A



距离
14.03567
15.55635
16.12452
18.11077
18.35756
19.41649
21.63331
23.32381
25.4951
26.30589
29.61419
32.89377
33.01515
36.05551
44.82187
45.34314
53.60037
68.00735
72.06941
72.34639

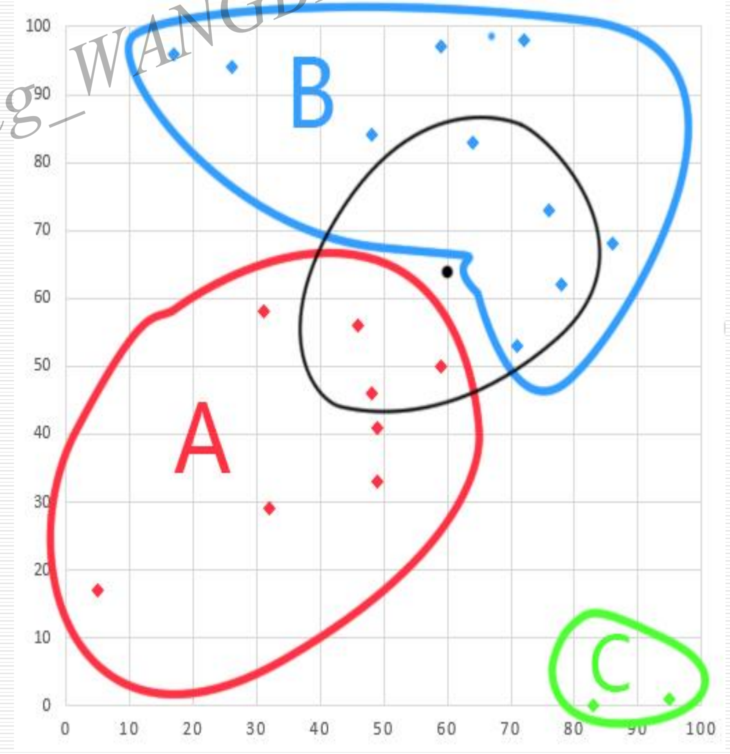
kNN算法

◆ **示例1：已知20个数据样本，其分类结果如下图(三个类别)**

- 7个点中有4个属于B类，3个属于A类，P(A) = 3 / 20, P(B) = 4 / 20

- 返回前k个点出现概率最高的类别作为当前点的预测类别

由此将新数据归为？类



kNN算法

KNN思想

- 当一个未知标签的数据需要分类时，
- 求其与训练集中每个数据对象的距离，
- 然后选择离未知数据最近的***k***个点，
- 分别查看其归属，
- 最后将未知数据归入包含点数最多的类别。

kNN算法

输入：训练样本，*k*为近邻数，未知数据点x

输出：x所属的类别

处理：

（1）计算当前数据点与已知类别数据集中的每个点之间的距离；

（2）按照距离递增次序排序；

（3）选取与当前点距离最小的*k*个点；

（4）确定前*k*个点所属类别的出现概率；

（5）返回前*k*个点出现概率最高的类别作为当前点的预测类别。

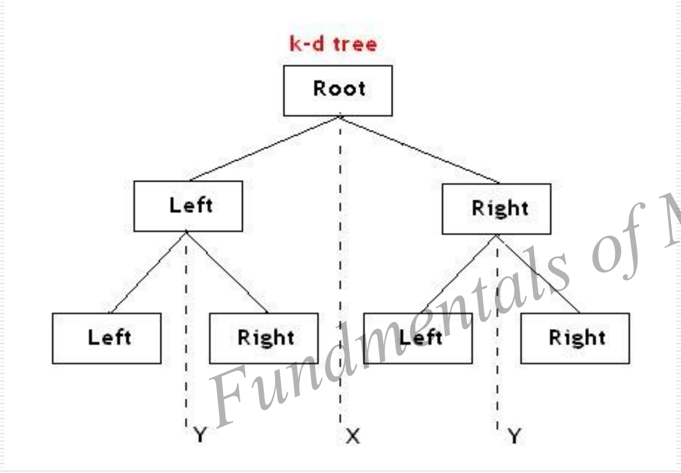
kNN算法

□ *k*NN算法性能

- 最简单有效的分类方法
- 无需估计参数，即无需训练，属非参数模型
- 只计算“最近”邻居样本。当样本不平衡时，*k*个邻居中大容量类的样本占多数或占极少数。
- 计算复杂度高，需计算新的数据点与样本集中每个数据的距离，时间复杂度是$O(n)$，空间度杂度也高。

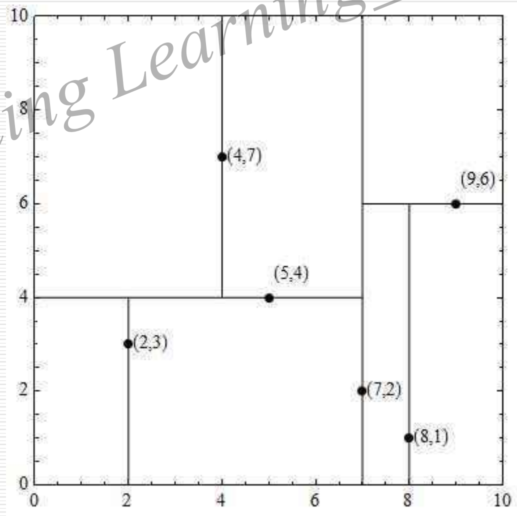
kNN算法

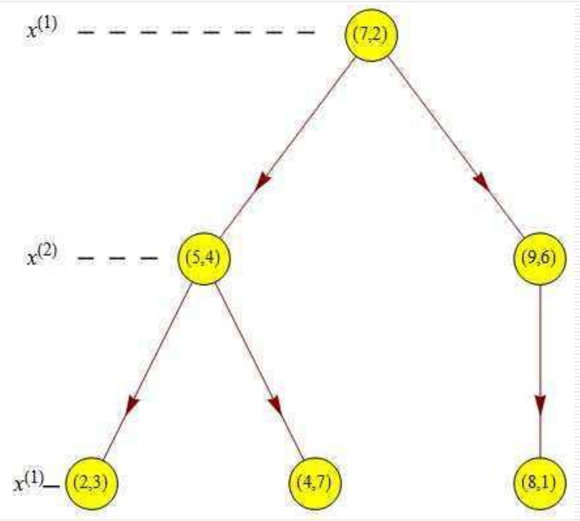
- kNN实现：kd 树



- 构造 kd 树，搜索 kd 树

- **示例**：给定一个二维空间的数据集：T={(2,3), (5,4), (9,6,), (4,7), (8,1), (7,2)}构建一个平衡kd树





李航著. 统计学习方法（第2版）. 北京: 清华大学出版社, 2019.5

sklearn.neighbors

- **class**：sklearn.neighbors.KNeighborsClassifier(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)

https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html#sklearn.neighbors.KNeighborsClassifier

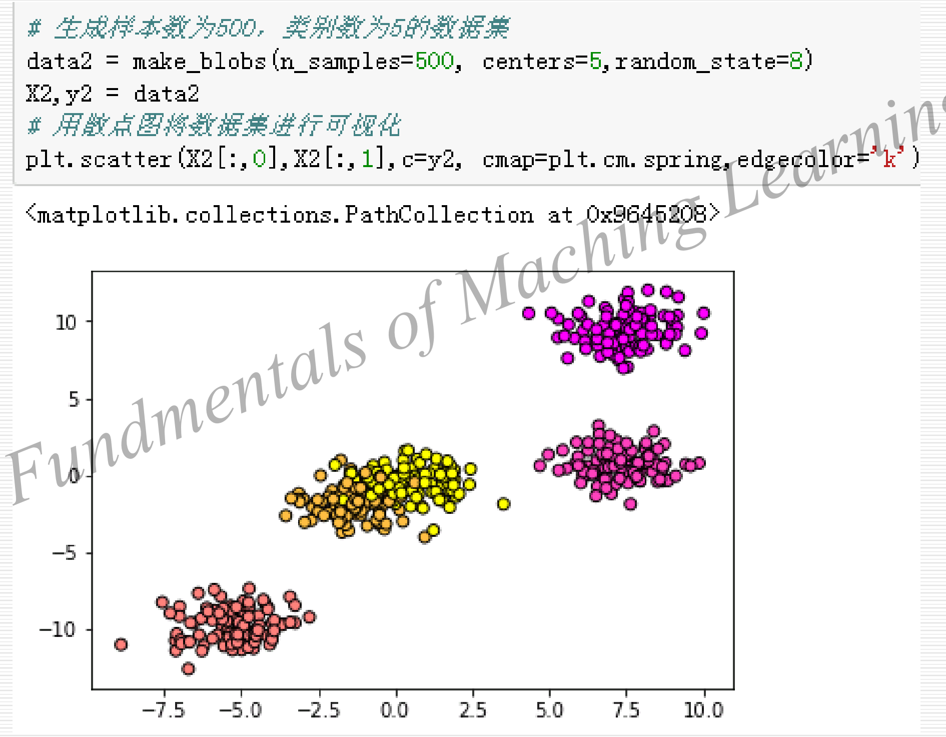
sklearn.neighbors

Methods

fit(self, X, y)	Fit the model using X as training data and y as target values
get_params(self[, deep])	Get parameters for this estimator.
kneighbors(self[, X, n_neighbors, …])	Finds the K-neighbors of a point.
kneighbors_graph(self[, X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict(self, X)	Predict the class labels for the provided data.
predict_proba(self, X)	Return probability estimates for the test data X.
score(self, X, y[, sample_weight])	Return the mean accuracy on the given test data and labels.
set_params(self, **params)	Set the parameters of this estimator.

sklearn.neighbors

◆ 示例：sklearn.neighbors.KNeigborsClassifier

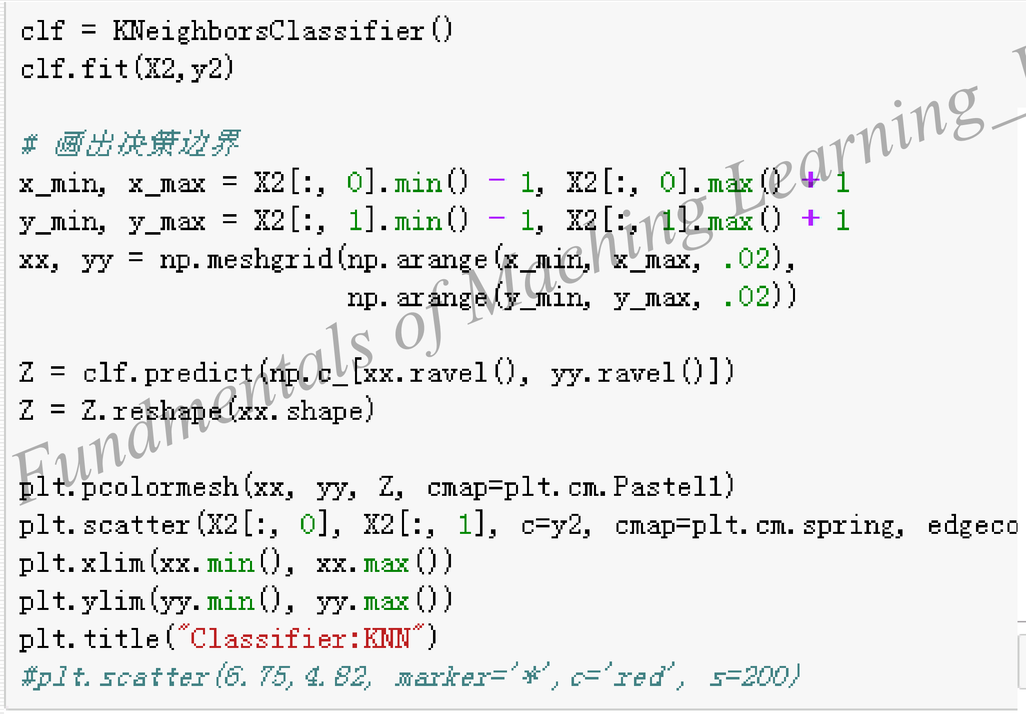


```
# 生成样本数为500，类别数为5的数据集
data2 = make_blobs(n_samples=500, centers=5,random_state=8)
X2,y2 = data2
# 用散点图将数据集进行可视化
plt.scatter(X2[:,0],X2[:,1],c=y2, cmap=plt.cm.spring,edgecolor='k')
```

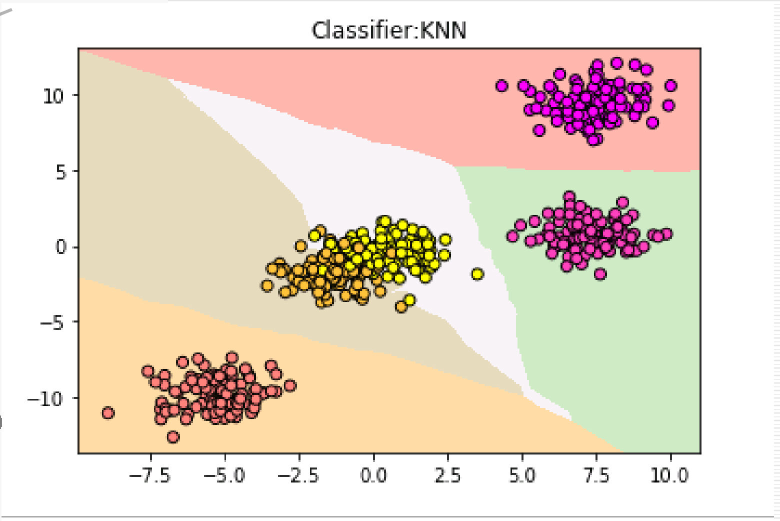
```
<matplotlib.collections.PathCollection at 0x9645208>
```

sklearn.neighbors

◆ 示例：sklearn.neighbors.KNeigborsClassifier



```
clf = KNeighborsClassifier()
clf.fit(X2,y2)

# 画出决策边界
x_min, x_max = X2[:, 0].min() - 1, X2[:, 0].max() + 1
y_min, y_max = X2[:, 1].min() - 1, X2[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, .02),
                     np.arange(y_min, y_max, .02))

Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Pastel1)
plt.scatter(X2[:, 0], X2[:, 1], c=y2, cmap=plt.cm.spring, edgeco
plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.title("Classifier:KNN")
#plt.scatter(6.75,4.82, marker='*', c='red', s=200)
```



```
print('模型正确率: {:.2f}'.format(clf.score(X2,y2)))
```

```
模型正确率: 0.96
```

sklearn.neighbors

- **sklearn.neighbors.KNeigborsClassifier**：适合多类别分类
- **sklearn.neighbors.KNeighborsRegressor**：回归预测

sklearn.neighbors

- **class：** sklearn.neighbors.KNeighborsRegressor(n_neighbors=5, *, weights='uniform', algorithm='auto', leaf_size=30, p=2, metric='minkowski', metric_params=None, n_jobs=None, **kwargs)

  https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsRegressor.html#sklearn.neighbors.KNeighborsRegressor

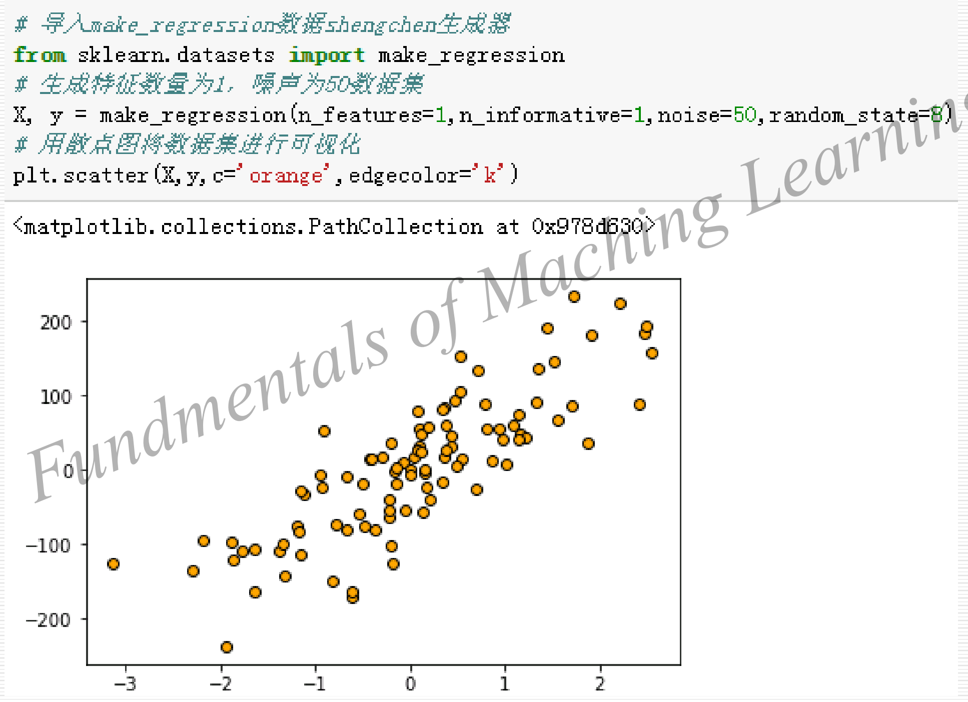
sklearn.neighbors

Methods

fit(self, X, y)	Fit the model using X as training data and y as target values
get_params(self[, deep])	Get parameters for this estimator.
kneighbors(self[, X, n_neighbors, …])	Finds the K-neighbors of a point.
kneighbors_graph(self[, X, n_neighbors, mode])	Computes the (weighted) graph of k-Neighbors for points in X
predict(self, X)	Predict the target for the provided data
score(self, X, y[, sample_weight])	Return the coefficient of determination R^2 of the prediction.
set_params(self, **params)	Set the parameters of this estimator.

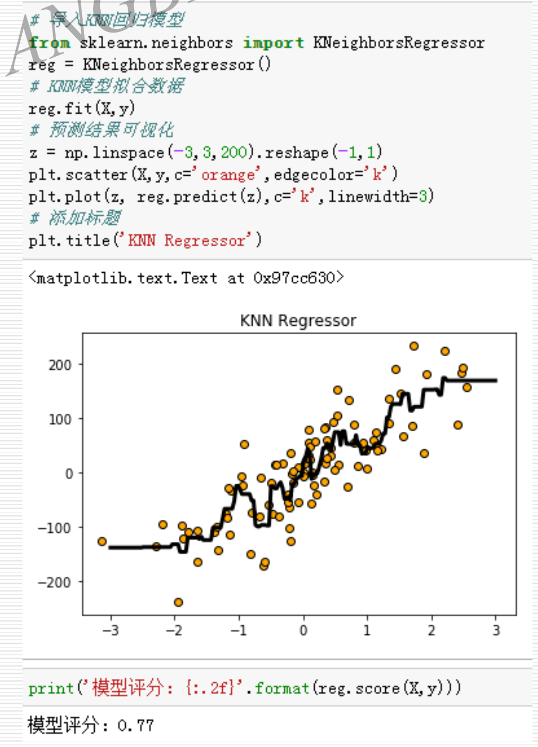
sklearn.neighbors

◆ 示例：sklearn.neighbors.KNeigborsRegressor



```
# 导入make_regression数据shengchen生成器
from sklearn.datasets import make_regression
# 生成特征数量为1，噪声为50数据集
X, y = make_regression(n_features=1,n_informative=1,noise=50,random_state=8)
# 用散点图将数据集进行可视化
plt.scatter(X,y,c='orange',edgecolor='k')
```

```
<matplotlib.collections.PathCollection at 0x978d630>
```



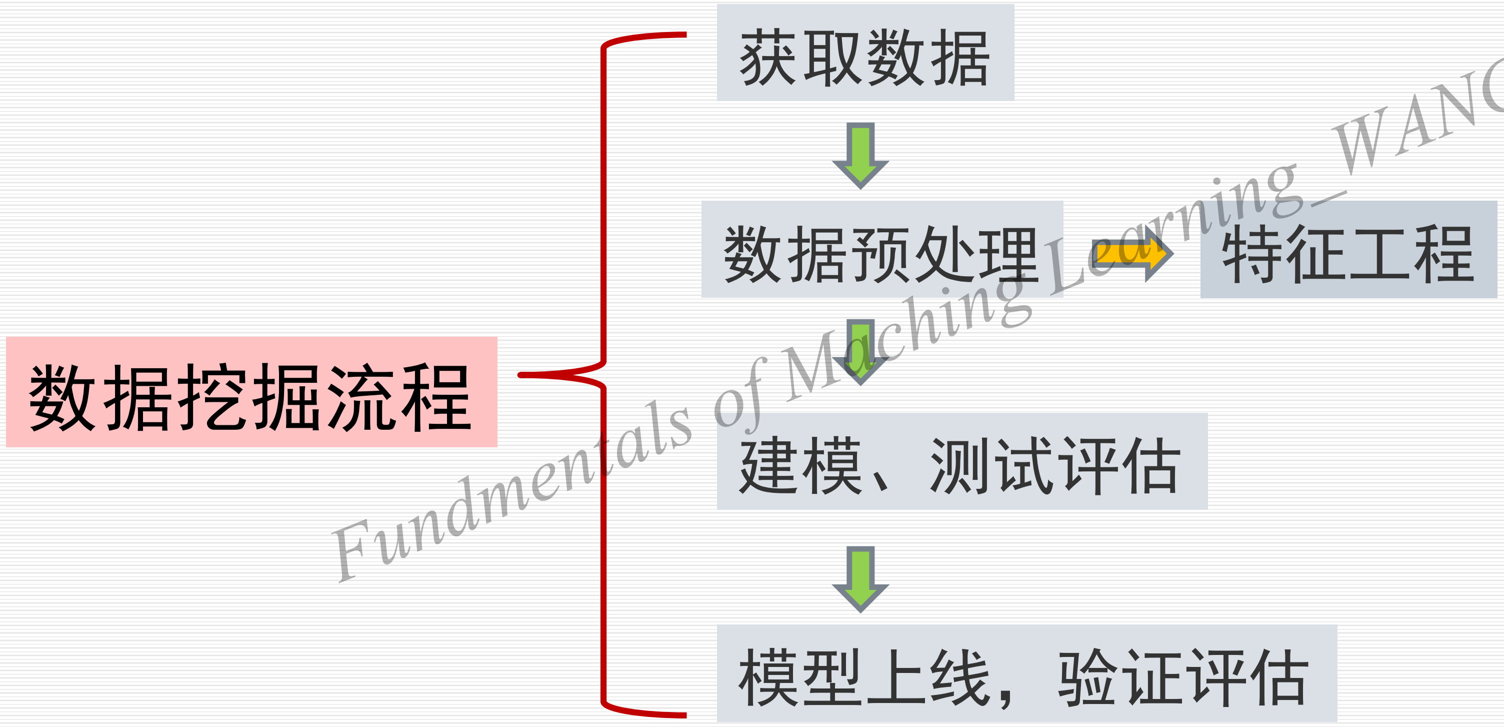
```
# 导入KNN回归模型
from sklearn.neighbors import KNeighborsRegressor
reg = KNeighborsRegressor()
# KNN模型拟合数据
reg.fit(X,y)
# 预测结果可视化
z = np.linspace(-3,3,200).reshape(-1,1)
plt.scatter(X,y,c='orange',edgecolor='k')
plt.plot(z, reg.predict(z),c='k',linewidth=3)
# 添加标题
plt.title('KNN Regressor')
```

```
<matplotlib.text.Text at 0x97cc630>
```

```
print('模型评分：{:.2f}'.format(reg.score(X,y)))
```

```
模型评分：0.77
```

数据预处理



数据预处理

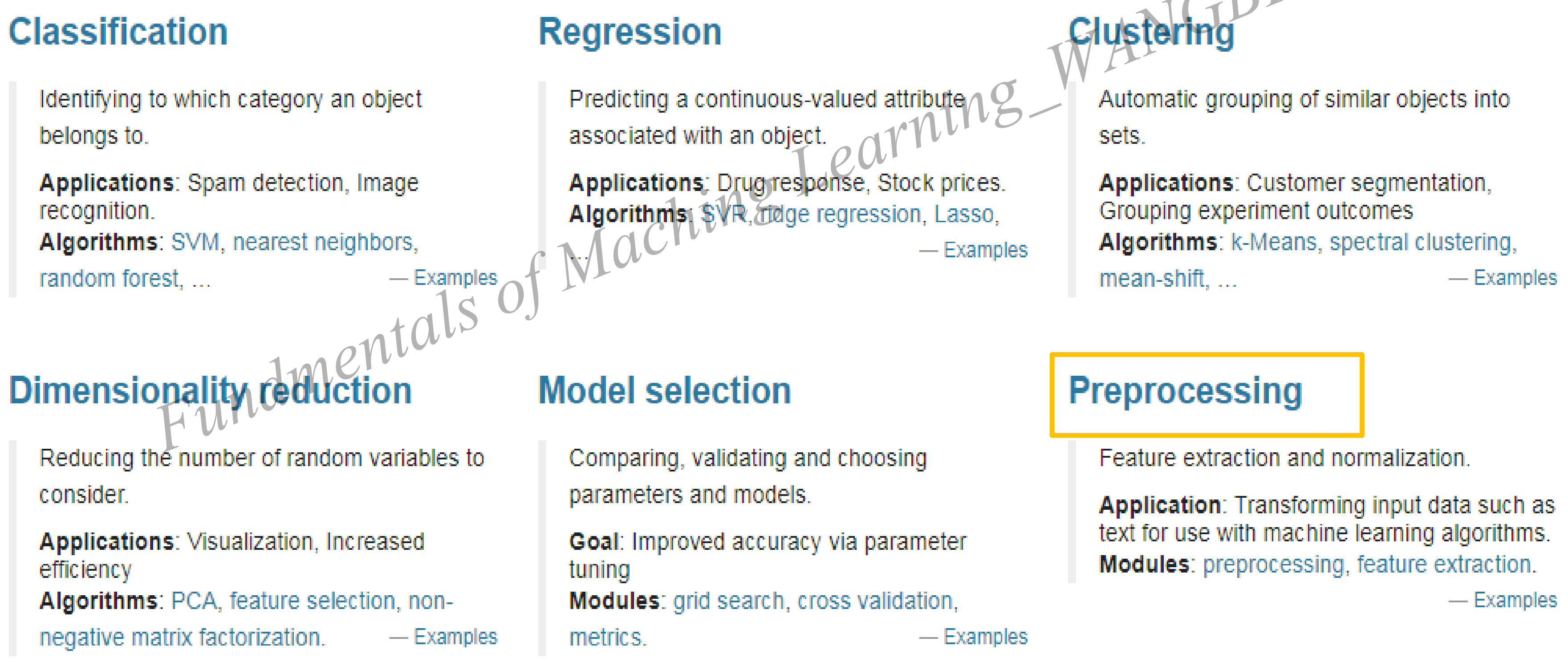
- 特征工程(Feature Engineering)：指对于特定应用，如何找到最佳数据表示。

“数据决定了机器学习的上限，而算法只是尽可能逼近这个上限而已”

- 特征处理：数据预处理
- 特征选择：从特征集中选择特征子集
- 特征降维：减少特征数量，并保留大部分有效信息

数据预处理

- sklearn中的数据预处理和特征工程



Classification

Identifying to which category an object belongs to.

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

Regression

Predicting a continuous-valued attribute associated with an object.

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

Clustering

Automatic grouping of similar objects into sets.

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

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization. — Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics. — Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction. — Examples

数据预处理

- sklearn中的数据预处理和特征工程
  - sklearn.preprocessing：几乎包含数据预处理的所有内容
  - sklearn.Impute：填补缺失值专用
  - sklearn.feature_selection：包含特征选择的各种方法的实践
  - sklearn.decomposition：包含降维算法

数据预处理

- sklearn.preprocessing

  https://scikit-learn.org/stable/modules/classes.html#module-sklearn.preprocessing

  - 范围缩放（Feature scaling）：将特征的取值区间缩放到某个特定范围，如最大最小缩放（min max scaling）
  - 标准化（standardization）：均值为零，标准差为1
  - 归一化（normalization）：把每个特征值都归到0-1范围
  - 二值化（binarization）：将数值特征向量转换为布尔型向量
  - 独热编码：特征向量的每个特征与特征的非重复总数相对应，通过one-of-k的形式对每个值进行编码
  - 标签编码：将标签编码转换成数字形式

数据预处理

- 不同特征常具有不同单位或量纲：

  数量级的差异将导致量级较大的特征占主导

  数量级的差异将导致迭代收敛速度减慢

  有些距离算法对于数据的数量级非常敏感

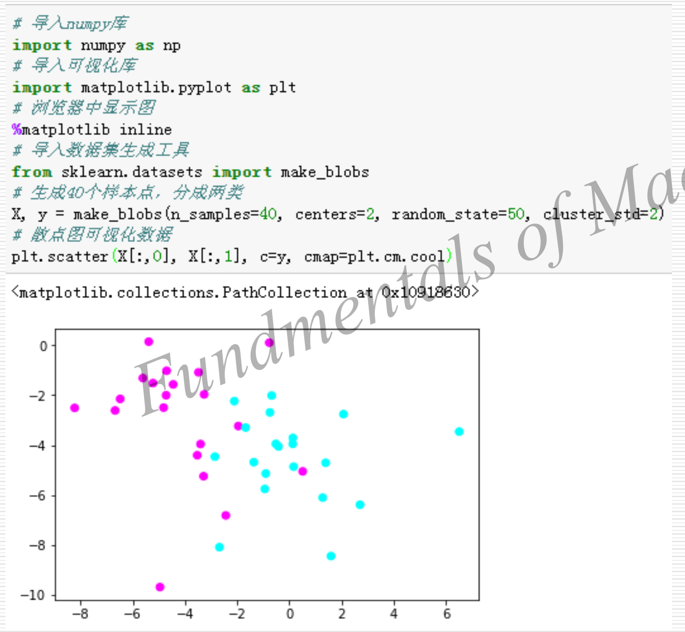
- 无量纲化：将不同规格的数据转换到同一规格，

  或不同分布的数据转换到某个特定分布的需求

数据预处理

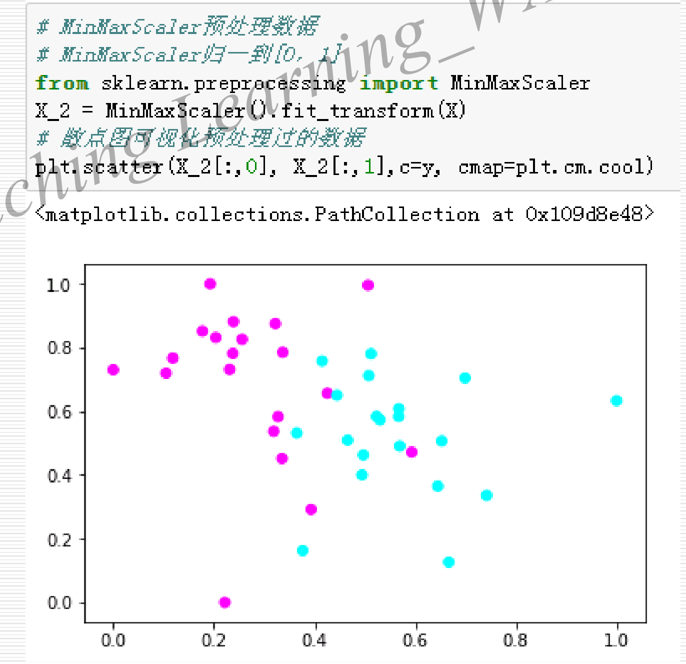
◆ 示例：class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)

$$x_{new} = \frac{x - \min}{\max - \min}$$



```
# 导入numpy库
import numpy as np
# 导入可视化库
import matplotlib.pyplot as plt
# 浏览器中显示图
%matplotlib inline
# 导入数据集生成工具
from sklearn.datasets import make_blobs
# 生成40个样本点，分成两类
X, y = make_blobs(n_samples=40, centers=2, random_state=50, cluster_std=2)
# 散点图可视化数据
plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.cool)
```

```
<matplotlib.collections.PathCollection at 0x10918630>
```



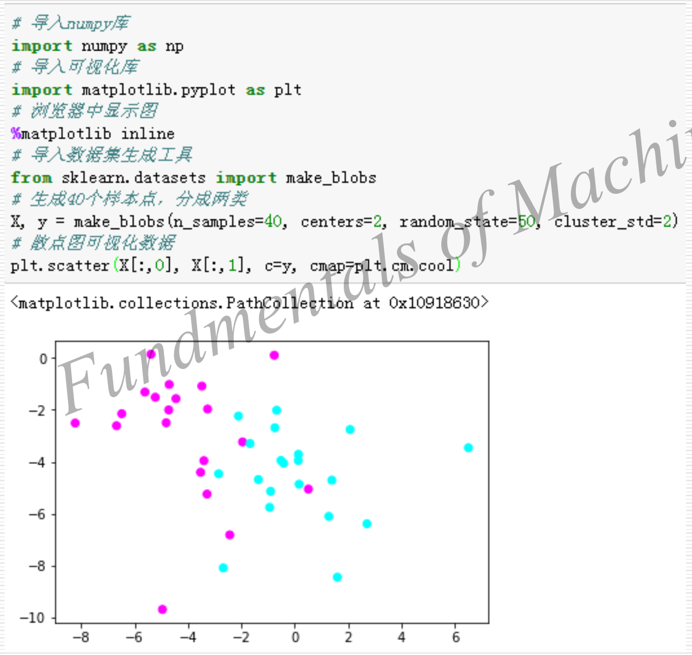
```
# MinMaxScaler预处理数据
# MinMaxScaler归一到[0, 1]
from sklearn.preprocessing import MinMaxScaler
X_2 = MinMaxScaler().fit_transform(X)
# 散点图可视化预处理过的数据
plt.scatter(X_2[:,0], X_2[:,1],c=y, cmap=plt.cm.cool)
```

```
<matplotlib.collections.PathCollection at 0x109d8e48>
```

数据预处理

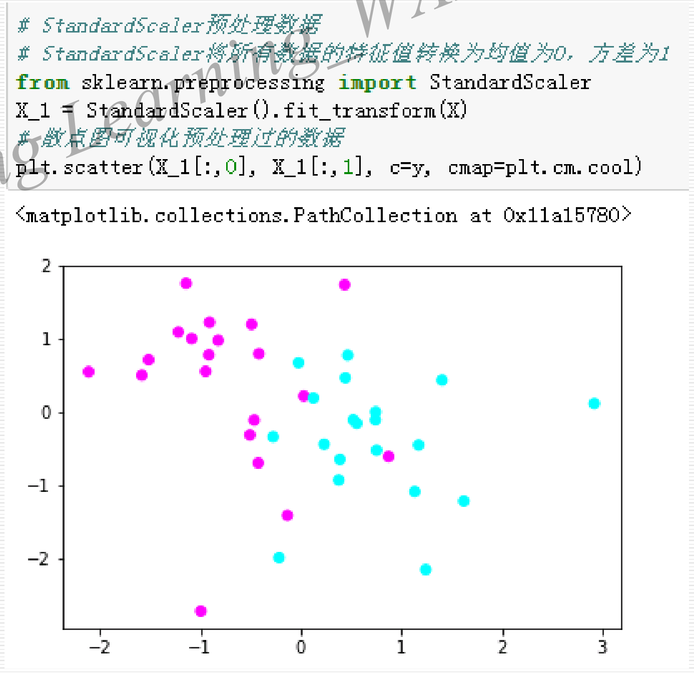
- 示例：class sklearn.preprocessing.StandardScaler(*, copy=True, with_mean=True, with_std=True)

$$x_{new} = \frac{x-u}{\sigma}$$



```
# 导入numpy库
import numpy as np
# 导入可视化库
import matplotlib.pyplot as plt
# 浏览器中显示图
%matplotlib inline
# 导入数据集生成工具
from sklearn.datasets import make_blobs
# 生成40个样本点，分成两类
X, y = make_blobs(n_samples=40, centers=2, random_state=50, cluster_std=2)
# 散点图可视化数据
plt.scatter(X[:,0], X[:,1], c=y, cmap=plt.cm.cool)
```

```
<matplotlib.collections.PathCollection at 0x10918630>
```

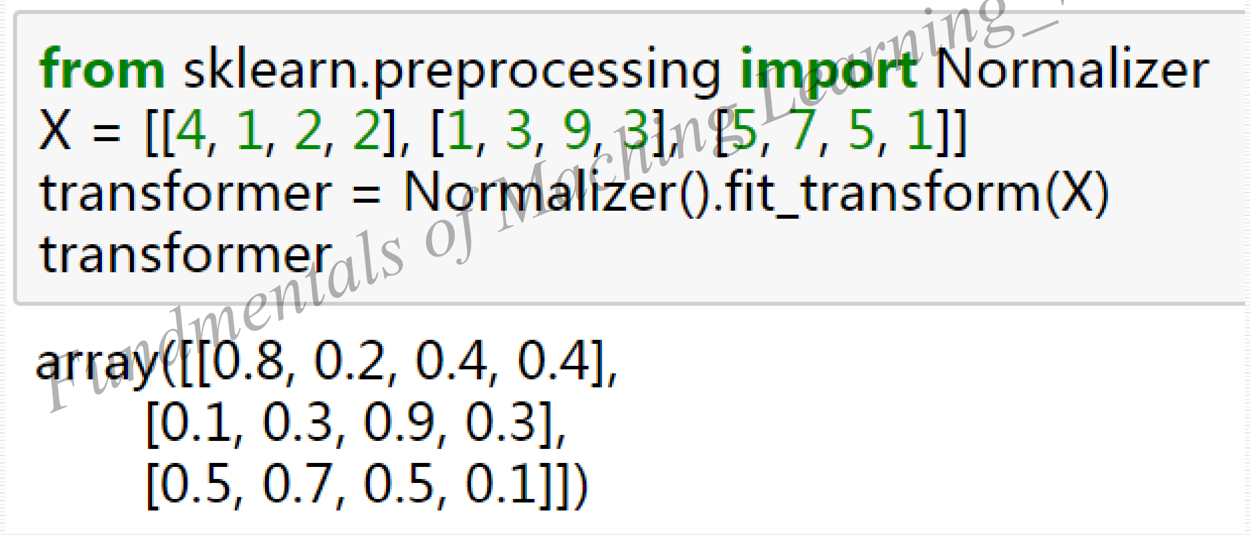


```
# StandardScaler预处理数据
# StandardScaler将所有数据的特征值转换为均值为0，方差为1
from sklearn.preprocessing import StandardScaler
X_1 = StandardScaler().fit_transform(X)
#散点图可视化预处理过的数据
plt.scatter(X_1[:,0], X_1[:,1], c=y, cmap=plt.cm.cool)
```

```
<matplotlib.collections.PathCollection at 0x11a15780>
```

数据预处理

- 示例：class sklearn.preprocessing.normalize(X, norm='l2', *, axis=1, copy=True, return_norm=False)



```
from sklearn.preprocessing import Normalizer
X = [[4, 1, 2, 2], [1, 3, 9, 3], [5, 7, 5, 1]]
transformer = Normalizer().fit_transform(X)
transformer
```

```
array([[0.8, 0.2, 0.4, 0.4],
       [0.1, 0.3, 0.9, 0.3],
       [0.5, 0.7, 0.5, 0.1]])
```

kNN算法

- 向量的范数(norm)：表示向量的长度或大小

向量 $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ 的范数计算

一范数： $l_1 = |x_1| + |x_2| + \ldots + |x_d|$

二范数： $l_2 = \sqrt{x_1^2 + x_2^2 + \ldots + x_d^2}$

无穷范数： $l_\infty = \max(|x_k|),\ \ k = 1, 2, \ldots, d$