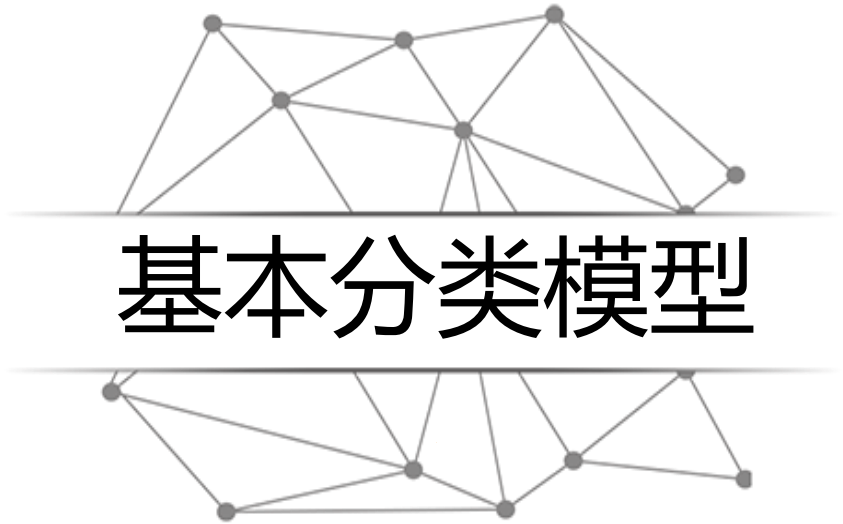
# 监督学习

*ML13*

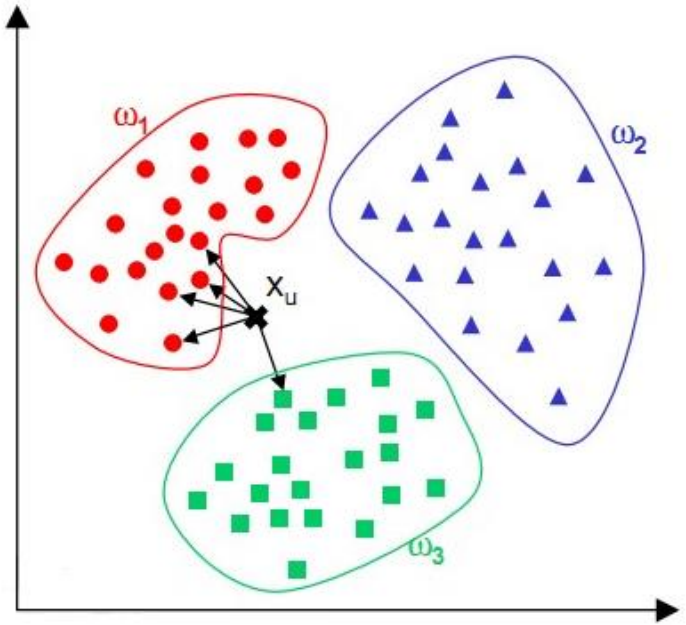

礼欣
www.python123.org



# 基本分类模型

# K近邻分类器(KNN)

KNN：通过计算待分类数据点，与已有数据集中的所有数据点的距离。取距离最小的前K个点，根据“少数服从多数“的原则，将这个数据点划分为出现次数最多的那个类别。



# sklearn中的K近邻分类器

在sklearn库中，可以使用sklearn.neighbors.KNeighborsClassifier创建一个K近邻分类器，主要参数有：

- n_neighbors：用于指定分类器中K的大小(默认值为5，注意与kmeans的区别)
- weights：设置选中的K个点对分类结果影响的权重（默认值为平均权重“uniform”，可以选择“distance”代表越近的点权重越高，或者传入自己编写的以距离为参数的权重计算函数）

# sklearn中的K近邻分类器

它的主要参数还有：

- algorithm：设置用于计算临近点的方法，因为当数据量很大的情况下计算当前点和所有点的距离再选出最近的k各点，这个计算量是很费时的，所以（选项中有ball_tree、kd_tree和brute，分别代表不同的寻找邻居的优化算法，默认值为auto，根据训练数据自动选择）

# K近邻分类器的使用

创建一组数据 X 和它对应的标签 y：

```
>>> X = [[0], [1], [2], [3]]
>>> y = [0, 0, 1, 1]
```

使用 import 语句导入 K 近邻分类器。

```
>>> from sklearn.neighbors import KNeighborsClassifier
```

# K近邻分类器的使用

参数 n_neighbors 设置为 3，即使用最近的3个邻居作为分类的依据，其他参数保持默认值，并将创建好的实例赋给变量 neigh。

```
>>> neigh = KNeighborsClassifier(n_neighbors=3)
```

调用 fit() 函数，将训练数据 X 和 标签 y 送入分类器进行学习。

```
>>> neigh.fit(X, y)
```

# K近邻分类器的使用

调用 predict() 函数，对未知分类样本 [1.1] 分类，可以直接并将需要分类的数据构造为数组形式作为参数传入，得到分类标签作为返回值。

```
>>> print(neigh.predict([[1.1]]))
[0]
```

样例输出值是 0，表示K近邻分类器通过计算样本 [1.1] 与训练数据的距离，取 0,1,2 这 3 个邻居作为依据，根据“投票法”最终将样本分为类别 0。

# KNN的使用经验

在实际使用时，我们可以使用所有训练数据构成特征 X 和标签 y，使用 fit() 函数进行训练。在正式分类时，通过一次性构造测试集或者一个一个输入样本的方式，得到样本对应的分类结果。有关K 的取值：

- 如果较大，相当于使用较大邻域中的训练实例进行预测，可以减小估计误差，但是距离较远的样本也会对预测起作用，导致预测错误。
- 相反地，如果 K 较小，相当于使用较小的邻域进行预测，如果邻居恰好是噪声点，会导致过拟合。
- 一般情况下，K 会倾向选取较小的值，并使用交叉验证法选取最优 K 值。

# 决策树

**决策树**是一种树形结构的分类器，通过顺序询问分类点的属性决定分类点最终的类别。通常根据特征的信息增益或其他指标，构建一颗决策树。在分类时，只需要按照决策树中的结点依次进行判断，即可得到样本所属类别。

例如，根据右图这个构造好的分类决策树，一个无房产，单身，年收入55K的人的会被归入无法偿还信用卡这个类别。

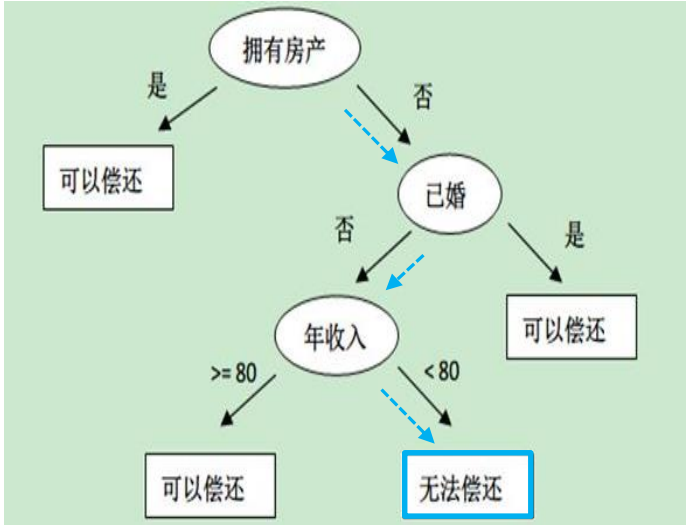



图. 信用卡偿还能力分类决策树

# sklearn中的决策树

在sklearn库中，可以使用sklearn.tree.DecisionTreeClassifier创建一个决策树用于分类，其主要参数有：

- criterion ：用于选择属性的准则，可以传入“gini”代表基尼系数，或者“entropy”代表信息增益。
- max_features ：表示在决策树结点进行分裂时，从多少个特征中选择最优特征。可以设定固定数目、百分比或其他标准。它的默认值是使用所有特征个数。

# 决策树的使用

首先，我们导入 sklearn 内嵌的鸢尾花数据集：

```
>>> from sklearn.datasets import load_iris
```

接下来，我们使用 import 语句导入决策树分类器，同时导入计算交叉验证值的函数 cross_val_score。

```
>>> from sklearn.tree import DecisionTreeClassifier
>>> from sklearn.model_selection import cross_val_score
```

# 决策树的使用

我们使用默认参数，创建一颗基于基尼系数的决策树，并将该决策树分类器赋值给变量 clf。

```
>>> clf = DecisionTreeClassifier()
```

将鸢尾花数据赋值给变量 iris。

```
>>> iris = load_iris()
```

# 决策树的使用

这里我们将决策树分类器做为待评估的模型，iris.data鸢尾花数据做为特征，iris.target鸢尾花分类标签做为目标结果，通过设定cv为10，使用10折交叉验证。得到最终的交叉验证得分。

```
>>> cross_val_score(clf, iris.data, iris.target, cv=10)
...
array([ 1. , 0.93..., 0.86..., 0.93..., 0.93...,
        0.93..., 0.93..., 1. , 0.93..., 1. ])
```

# 决策树的使用

以仿照之前 K近邻分类器的使用方法，利用 fit() 函数训练模型并使用 predict() 函数预测：

```
>>> clf.fit(X, y)
>>> clf.predict(x)
```

# 决策树

- 决策树本质上是寻找一种对特征空间上的划分，旨在构建一个训练数据拟合的好，并且复杂度小的决策树。
- 在实际使用中，需要根据数据情况，调整DecisionTreeClassifier类中传入的参数，比如选择合适的criterion，设置随机变量等。

# 朴素贝叶斯

朴素贝叶斯分类器是一个以贝叶斯定理为基础的多分类的分类器。

对于给定数据，首先基于特征的条件独立性假设，学习输入输出的联合概率分布，然后基于此模型，对给定的输入x，利用贝叶斯定理求出后验概率最大的输出y。

$$\boldsymbol{p(A|B) = \frac{p(B|A) \cdot p(A)}{p(B)}}$$

# sklearn中的朴素贝叶斯

在sklearn库中，实现了三个朴素贝叶斯分类器，如下表所示：

| 分类器 | 描述 |
| --- | --- |
| naive_bayes.GussianNB | 高斯朴素贝叶斯 |
| naive_bayes.MultinomialNB | 针对多项式模型的朴素贝叶斯分类器 |
| naive_bayes.BernoulliNB | 针对多元伯努利模型的朴素贝叶斯分类器 |

区别在于假设某一特征的所有属于某个类别的观测值符合特定分布，如，分类问题的特征包括人的身高，身高符合高斯分布，这类问题适合高斯朴素贝叶斯

# sklearn中的朴素贝叶斯

在sklearn库中，可以使用sklearn.naive_bayes.GaussianNB创建一个高斯朴素贝叶斯分类器，其参数有：

- priors ：给定各个类别的先验概率。如果为空，则按训练数据的实际情况进行统计；如果给定先验概率，则在训练过程中不能更改。

# 朴素贝叶斯的使用

例1：导入 numpy 库，并构造训练数据 X 和 y。

```
>>> import numpy as np
>>> X = np.array([[-1, -1], [-2, -1], [-3, -2], [1, 1], [2, 1], [3, 2]])
>>> Y = np.array([1, 1, 1, 2, 2, 2])
```

使用 import 语句导入朴素贝叶斯分类器。

```
>>> from sklearn.naive_bayes import GaussianNB
```

# 朴素贝叶斯的使用

使用默认参数，创建一个高斯朴素贝叶斯分类器，并将该分类器赋给变量 clf。

```
>>> clf = GaussianNB(priors=None)
```

类似的，使用 fit() 函数进行训练，并使用 predict() 函数进行预测，得到预测结果为 1。（测试时可以构造二维数组达到同时预测多个样本的目的）

```
>>> clf.fit(X, Y)
>>> print(clf.predict([[-0.8, -1]]))
[1]
```

# 朴素贝叶斯

朴素贝叶斯是典型的生成学习方法，由训练数据学习联合概率分布，并求得后验概率分布。

朴素贝叶斯一般在小规模数据上的表现很好，适合进行多分类任务。