

Python编程与人工智能实践

算法篇：算法篇小结

numpy简介

于泓

鲁东大学

信息与电气工程学院

2021.4.6

算法篇小结 分类模型

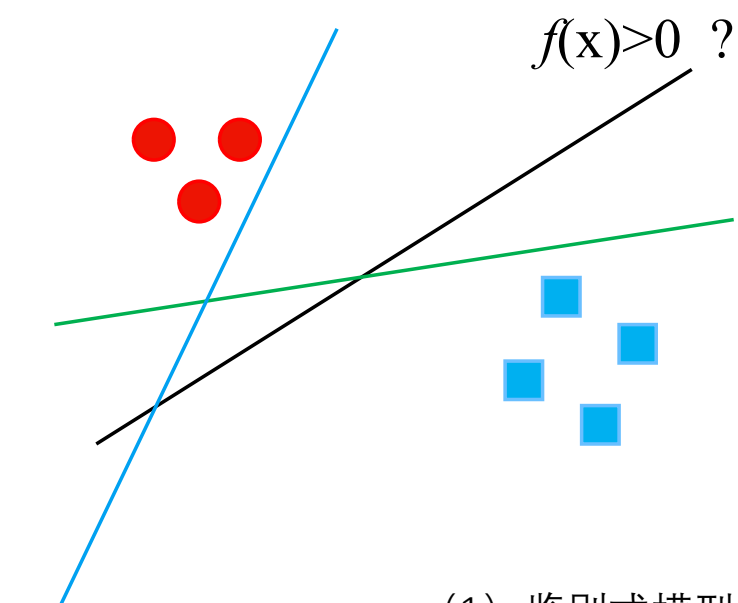
- KNN (K近邻) : 利用训练集直接作为分类模型
- 决策树 : 根据**熵最小化原则**, 划分数据, 构建分类树
- 朴素贝叶斯: 利用**计数统计**的方法, 构建概率模型, 拟合数据分布
(每类一个模型)
- GMM (高斯混合模型) : 利用概率假设+最大似然, 构建概率模型, 拟合每类数据的分布
(每类一个模型)
- 逻辑回归: 利用一条线将数据分成两堆 (一个鉴别函数)。利用梯度下降, 进行参数寻优
- BP神经网络: 利用多节点的复杂网络进行数据分类, 利用梯度下降进行网络参数调优

算法篇小结 聚类模型

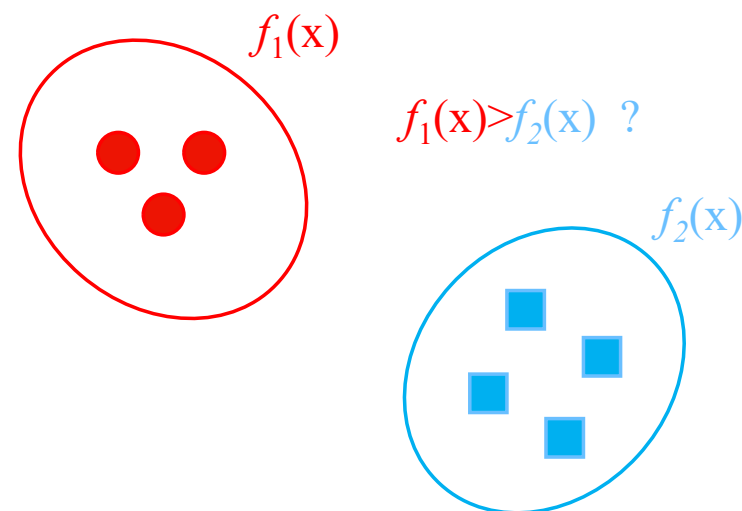
- K-Means (K均值) : 利用数据的均值作为聚类中心。
- GMM聚类 : 利用数据的均值与方差构造高斯模型, 作为聚类中心。
- 密度峰值聚类 (DP) : 计算每个点的局部密度, 以及寻找距离其最近且密度比其大的点的距离

鉴别式模型：基于神经网络的模型

生成式模型：基于概率统计的模型



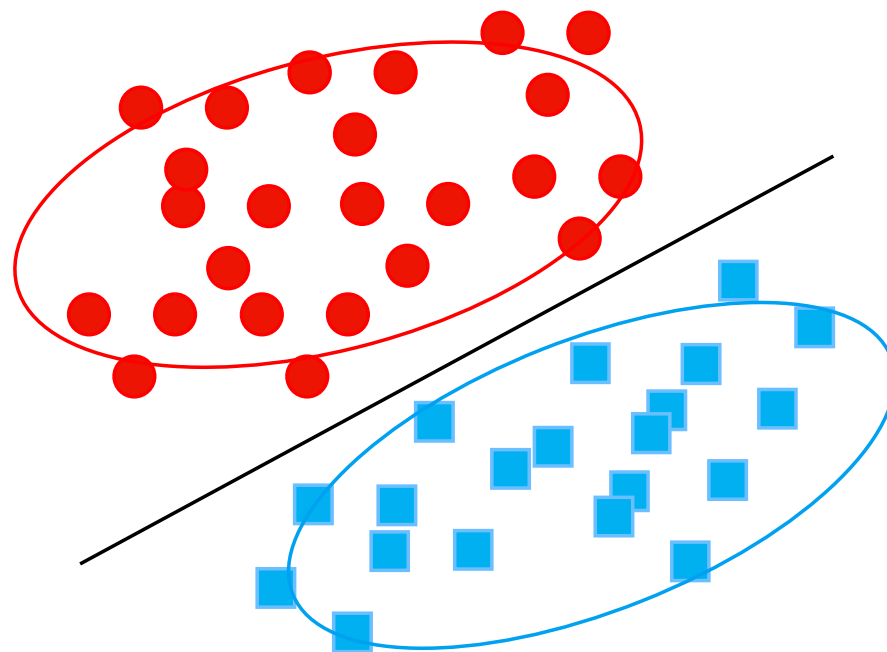
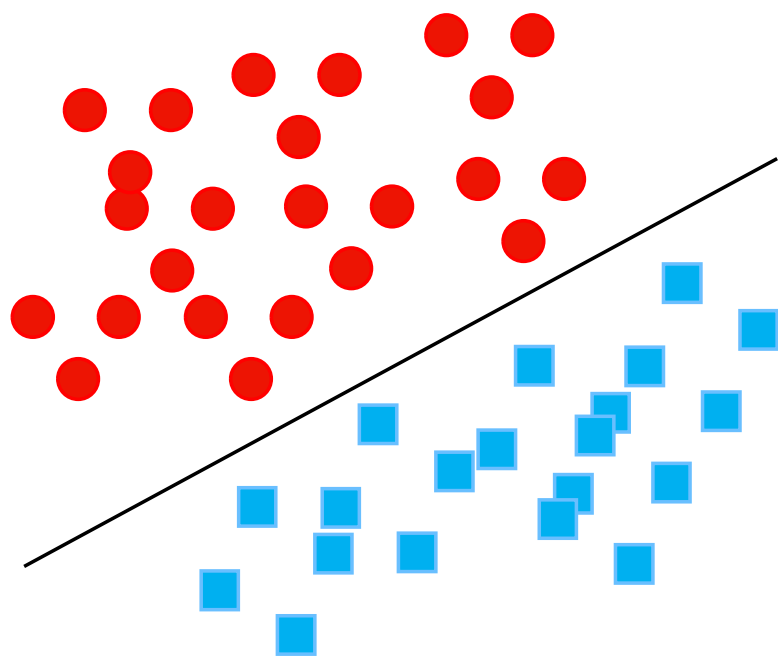
两者区别

训练数据
较少时

- (1) 鉴别式模型分类线位置可能不确定
生成式模型采用合理的概率假设，设定数据的分布
- (2) 鉴别式模型分类器采用梯度下降，进行参数优化，无法知道是否达到最优值
生成式模型采用EM算法可以保证每步迭代都使最大似然值减少

鉴别式模型：基于神经网络的模型

生成式模型：基于概率统计的模型



两者区别

训练数据
较多时

- (1) 鉴别式模型分类线位置可以较为精确
人工设计的概率模型，无法准确拟合复杂的数据分布
- (2) 鉴别式模型分类器采用随机梯度下降，训练速度提高，局部优值也可以接受
生成式模型需要利用全部数据进行参数更新，速度较慢，内存消耗过大

Numpy 简单介绍: 它是一个Python库, 提供多维数组对象, 各种派生对象 (如掩码数组和矩阵), 以及用于数组快速操作的各种API, 有包括数学、逻辑、形状操作、排序、选择、输入输出、离散傅立叶变换、基本线性代数, 基本统计运算和随机模拟等等。

Numpy 的安装: `pip install numpy`

Numpy的使用: 在代码的开端加入
`import numpy as np`

Numpy的对象: `ndarray`

Numpy的常用api:

- (1) 创建一个numpy变量:
查看变量的尺寸
查看变量的数据类型
修改变量的类型

```
>>> a = np.array([1,2,3,4,5,6])
>>> a.shape
(6,)
>>> a.dtype
dtype('int64')
```

```
>>> a = np.array([1,2,3,4,5,6]).astype(np.float32)
>>> a
array([1., 2., 3., 4., 5., 6.], dtype=float32)
```

```
>>> a = np.array([[1,2],[3,4],[5,6]])
>>> a.shape
(3, 2)
>>> np.shape(a)
(3, 2)
```

- (2) numpy变量和list之间的转换

```
>>> a = [1,2,3,4,5,6]
>>> b = np.array(a)
>>> b
array([1, 2, 3, 4, 5, 6])
```

```
>>> a= np.array([1,2,3,4,5,7])
>>> b = a.tolist()
>>> b
[1, 2, 3, 4, 5, 7]
```

```
>>> a = np.array([[1,2],[3,4],[5,6]])
>>> b = a.tolist()
>>> b
[[1, 2], [3, 4], [5, 6]]
>>> len(b)
3
```

(3) 创建全0、
全1的对象
Np.zeros() np.ones([])

```
>>> a = np.zeros([3,4])
>>> a
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

```
>>> a
array([[1., 1., 1., 1.],
       [1., 1., 1., 1.],
       [1., 1., 1., 1.]])
```

(4) numpy数据求和
np.sum()

```
>>> a= np.array([[1,2],[3,4],[5,6]])
>>>
>>> b = np.sum(a)
>>> b
21
```

```
>>> b = np.sum(a,axis=0)
>>> b
array([ 9, 12])
>>> b = np.sum(a,axis=1)
>>> b
array([ 3,  7, 11])
```

```
>>> b = np.sum(a,axis=1,keepdims=True)
>>> b
array([[ 3],
       [ 7],
       [11]])
>>> b.shape
(3, 1)
```

```
>>> b = np.sum(a,axis=0,keepdims=True)
>>> b
array([[ 9, 12]])
>>> b.shape
(1, 2)
```

Numpy数据求均值、标准差、均方方差 与np.sum ()类似

np.mean()

np.std() : std = sqrt(mean(x)) x = abs(a - a.mean())**2

np.var() : var = mean(x) x = abs(a - a.mean())**2

(5) 进行数据查找

np.where()

与.shape 配合可以获取数组中某个元素的数目

```
>>> a = np.array([[1,1],[2,3],[4,5],[1,7],[1,3]])
>>> b = np.where(a==1)
>>> b
(array([0, 0, 3, 4]), array([0, 1, 0, 0]))
```

```
>>> N_b = np.where(a==1)[0].shape[0]
>>> N_b
4
```

(6) 数据排序 np.sort()

```
>>> c = np.sort(a,axis=1)
>>> c
array([[1, 1],
       [2, 3],
       [4, 5],
       [1, 7],
       [1, 3]])
```

```
>>> c = np.sort(a,axis=0)
>>> c
array([[1, 1],
       [1, 3],
       [1, 3],
       [2, 5],
       [4, 7]])
```

```
>>> c = np.sort(a,axis=None)
>>> c
array([1, 1, 1, 1, 2, 3, 3, 4, 5, 7])
```

(7) 得到排序后的位置信息

np.arg(sort)

```
>>> a = np.array([4,4,2,2,5,6,7,1])
>>> b = np.argsort(a)
>>> b
array([7, 2, 3, 0, 1, 4, 5, 6])
```

```
>>> b = np.argsort(-a)
>>> b
array([6, 5, 4, 0, 1, 2, 3, 7])
```

从小到大

(8) 获取不重复的元素

np.unique()

```
>>> a = np.array([4,4,2,2,5,6,7,1])
>>> b = np.unique(a)
>>> b
array([1, 2, 4, 5, 6, 7])
>>> a = np.array(['cat','cat','dog','dog'])
>>> b = np.unique(a)
>>> b
array(['cat', 'dog'], dtype='<U3')
```

(9) 取最大, 最小值
np.max np.min()

```
>>> a = np.array([[1,1],[2,3],[4,5],[1,7],[1,3]])
>>> np.max(a)
7
>>> np.max(a,axis=0)
array([4, 7])
>>> np.max(a,axis=1)
array([1, 3, 5, 7, 3])
>>> np.max(a,axis=1,keepdims=True)
array([[1],
       [3],
       [5],
       [7],
       [3]])
```

(10) 取最大, 最小值的位置
np.argmin() np.argmax()

[1,1]
[2,3]
[4,5]
[1,-1]
[1,3]

```
>>> a = np.array([1,4,5,8,-2,0,0])
>>> np.argmin(a)
4
```

```
>>> a = np.array([[1,1],[2,3],[4,5],[1,-1],[1,3]])
>>> np.argmin(a)
7
>>> np.argmin(a,axis=0)
array([0, 3])
```

```
>>> a
array([[0.2, 0.8],
       [0.3, 0.6],
       [0.7, 0.3],
       [0.5, 0.6]])
>>> np.argmax(a,axis=1)
array([1, 1, 0, 1])
```

(11) 对一个区间进行均匀采样
np.linspace()

```
>>> a = np.linspace(0,10,5)
>>> a
array([ 0. ,  2.5,  5. ,  7.5, 10. ])
```

```
>>> a = np.linspace([0,10],[10,20],5)
>>> a
array([[ 0. , 10. ],
       [ 2.5, 12.5],
       [ 5. , 15. ],
       [ 7.5, 17.5],
       [10. , 20. ]])
```

(12) 维度扩展
np.newaxis
np.expand_dims

```
>>> a = np.array([0,1,2,3])
>>> a.shape
(4,)
```

```
>>> b = a[:,np.newaxis]
>>> b
array([[0],
       [1],
       [2],
       [3]])
>>> b.shape
(4, 1)
```

```
>>> b = a[np.newaxis,:]
>>> b
array([[0, 1, 2, 3]])
>>> b.shape
(1, 4)
```

```
>>> a = np.array([1,2,3])
>>> b= np.expand_dims(a,axis=0)
>>> b.shape
(1, 3)
```

(13) 矩阵拼接
np.r_
np.c_
np.concatenate

```
>>> a = np.ones([3,2])
>>> b = 2*np.ones([1,2])
>>> c = np.r_[a,b]
>>> c
array([[1., 1.],
       [1., 1.],
       [1., 1.],
       [2., 2.]])
```

```
>>> a = np.ones([3,2])
>>> b = 2*np.ones([3,1])
>>> c = np.c_[a,b]
>>> c
array([[1., 1., 2.],
       [1., 1., 2.],
       [1., 1., 2.]])
```

```
>>> c = np.concatenate((a,b,a),axis=1)
>>> c
array([[1., 1., 2., 1., 1.],
       [1., 1., 2., 1., 1.],
       [1., 1., 2., 1., 1.]])
```

(14) 矩阵的复制
np.tile

```
>>> a = np.array([1,2,3,4,5])
>>> np.tile(a,[3,1])
array([[1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5]])
>>> np.tile(a,[3,2])
array([[1, 2, 3, 4, 5, 1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5, 1, 2, 3, 4, 5],
       [1, 2, 3, 4, 5, 1, 2, 3, 4, 5]])
```

```
>>> a = np.array([[1,2,2],[3,4,5]])
>>> np.tile(a,[3,1])
array([[1, 2, 2],
       [3, 4, 5],
       [1, 2, 2],
       [3, 4, 5],
       [1, 2, 2],
       [3, 4, 5]])
>>> np.tile(a,[3,2])
array([[1, 2, 2, 1, 2, 2],
       [3, 4, 5, 3, 4, 5],
       [1, 2, 2, 1, 2, 2],
       [3, 4, 5, 3, 4, 5],
       [1, 2, 2, 1, 2, 2],
       [3, 4, 5, 3, 4, 5]])
```

14 numpy 矩阵的+ - * /

1 当参与计算的2个矩阵维度
相同时，实现逐元素操作

```
>>> a = np.ones([3,2])
>>> b = 2*np.ones([3,2])
>>> a+b
array([[3., 3.],
       [3., 3.],
       [3., 3.]])
>>> a-b
array([[-1., -1.],
       [-1., -1.],
       [-1., -1.]])
>>> a*b
array([[2., 2.],
       [2., 2.],
       [2., 2.]])
>>> a/b
array([[0.5, 0.5],
       [0.5, 0.5],
       [0.5, 0.5]])
```

2 当参与计算的一个是矩阵
一个是标量时
矩阵中的每个元素和标量进行操作

```
>>> a = 3*np.ones([3,2])
>>> a
array([[3., 3.],
       [3., 3.],
       [3., 3.]])
```

```
>>> a+1
array([[4., 4.],
       [4., 4.],
       [4., 4.]])
```

```
>>> a-1
array([[2., 2.],
       [2., 2.],
       [2., 2.]])
```

```
>>> a*2
array([[6., 6.],
       [6., 6.],
       [6., 6.]])
```

```
>>> a/2
array([[1.5, 1.5],
       [1.5, 1.5],
       [1.5, 1.5]])
```

3 当参与运算的 a 是一个是矩阵

b 是一个是矢量

矩阵的最后一个维度，要与矢量的维度相同

a 每行和b操作

```
>>> a = 3*np.ones([5,3])
>>> a
array([[3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.],
       [3., 3., 3.]])
```

```
>>> b = np.array([1,2,3])
```

```
>>> a+b
array([[4., 5., 6.],
       [4., 5., 6.],
       [4., 5., 6.],
       [4., 5., 6.],
       [4., 5., 6.]])
```

```
>>> a-b
array([[2., 1., 0.],
       [2., 1., 0.],
       [2., 1., 0.],
       [2., 1., 0.],
       [2., 1., 0.]])
```

```
>>> a*b
array([[3., 6., 9.],
       [3., 6., 9.],
       [3., 6., 9.],
       [3., 6., 9.],
       [3., 6., 9.]])
```

```
>>> a/b
array([[3. , 1.5, 1. ],
       [3. , 1.5, 1. ],
       [3. , 1.5, 1. ],
       [3. , 1.5, 1. ],
       [3. , 1.5, 1. ]])
```

4 当参与运算的 a 是一个是矩阵

b 是一个是矩阵且有1个维度为1

其他维度与a相同，沿着为1的维度
进行操作

例如：对a的每一列进行操作

```
>>> a
array([[3., 3., 3., 3., 3.],
       [3., 3., 3., 3., 3.],
       [3., 3., 3., 3., 3.]])
>>> a.shape
(3, 5)
>>> b
array([[1],
       [2],
       [3]])
>>> b.shape
(3, 1)
```

```
>>> a+b
array([[4., 4., 4., 4., 4.],
       [5., 5., 5., 5., 5.],
       [6., 6., 6., 6., 6.]])
>>> a-b
array([[2., 2., 2., 2., 2.],
       [1., 1., 1., 1., 1.],
       [0., 0., 0., 0., 0.]])
```

```
>>> a*b
array([[3., 3., 3., 3., 3.],
       [6., 6., 6., 6., 6.],
       [9., 9., 9., 9., 9.]])
>>> a/b
array([[3. , 3. , 3. , 3. , 3. ],
       [1.5, 1.5, 1.5, 1.5, 1.5],
       [1. , 1. , 1. , 1. , 1. ]])
```

15 np.dot(a,b)

- 1、a,b都是矢量：a和b 内积
(逐元素相乘、相加)

```
>>> a = np.array([1,2,3])
>>> b = np.array([4,5,6])
>>> np.dot(a,b)
32
```

- 2、a,b都是矩阵：a [M,N]
 b [N,K]
 矩阵乘 输出 [M,K]

```
>>> a = np.ones([3,2])
>>> a
array([[1., 1.],
       [1., 1.],
       [1., 1.]])
>>> b = np.ones([2,5])
>>> b
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
>>> np.dot(a,b)
array([[2., 2., 2., 2., 2.],
       [2., 2., 2., 2., 2.],
       [2., 2., 2., 2., 2.]])
```

- 3、a是矩阵 [M,N]
 b 是矢量 且与a最后一个维度相同
 [N,]
 b与a的每行进行内积

```
>>> b = [1,3]
>>> a
array([[3., 3.],
       [3., 3.],
       [3., 3.]])
>>> b = np.array([1,3])
>>> np.dot(a,b)
array([12., 12., 12.])
>>> np.dot(a,b).shape
(3,)
```

16 生成随机数

(1) 数据乱序

`np.random.permutation(N)`

```
>>> np.random.permutation(10)
array([9, 8, 4, 3, 7, 0, 1, 6, 2, 5])
>>> np.random.permutation(10)
array([6, 0, 4, 8, 1, 9, 2, 7, 3, 5])
```

(2) 正态分布的随机数

`np.random.randn(d1,d2,...)`

```
>>> np.random.randn(3,4,2)
array([[ 1.21515718, -1.26898201],
       [ 0.28892898, -0.16788155],
       [-0.46162184,  0.29647557],
       [-2.16995313,  0.98486189]],

      [[-0.45777392,  0.55592016],
       [-1.26882785, -0.57486763],
       [-0.10352497, -0.58018936],
       [-1.4595391 , -0.23485386]],

      [[ 1.55617945,  0.75623237],
       [-1.26219014, -0.05866999],
       [ 0.32611175, -0.05491684],
       [ 1.6667987 , -0.47386148]]])
```

(3) 多维高斯分布随机数

`np.random.multivariate_normal([2,2], [[.5,0],[0,.5]], 100)`

均值 方差 数据

17 数据的保存和加载

```
np.save(file,data)
np.load(file)
```

```
>>> a = np.array([1,2,3,4])
>>> np.save('a.npy',a)
>>> b = np.load("a.npy")
>>> b
array([1, 2, 3, 4])
```

也可以保存非np.array 格式的非正规化的数据
在load时需要加入参数 allow_pickle=True

```
>>> a2 = [{1:1},2,3,4]
>>> np.save("a2.npy",a2)
>>> b = np.load("a2.npy",allow_pickle=True)
>>> b
array([[{1: 1}], 2, 3, 4], dtype=object)
>>>
```

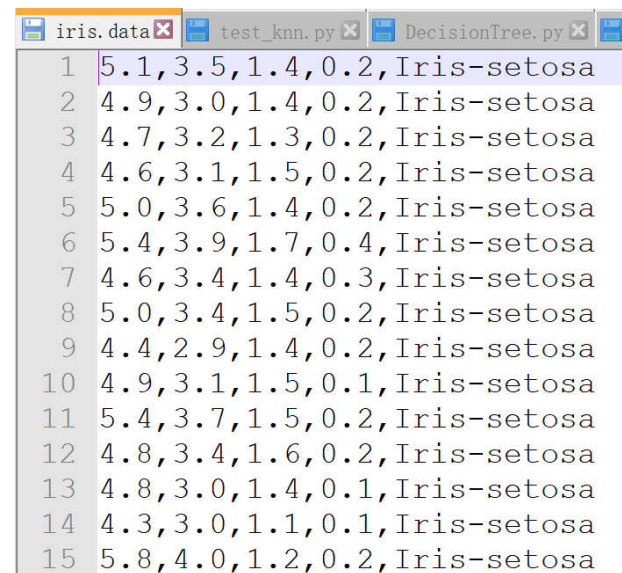
18 加载文本数据

```
file_data = 'iris.data'
```

```
# 数据读取
```

```
data = np.loadtxt(file_data,dtype = np.float, delimiter = ',',usecols=(0,1,2,3))
```

```
lab = np.loadtxt(file_data,dtype = str, delimiter = ',',usecols=(4))
```



```
iris.data x test_knn.py x DecisionTree.py x
1 5.1,3.5,1.4,0.2,Iris-setosa
2 4.9,3.0,1.4,0.2,Iris-setosa
3 4.7,3.2,1.3,0.2,Iris-setosa
4 4.6,3.1,1.5,0.2,Iris-setosa
5 5.0,3.6,1.4,0.2,Iris-setosa
6 5.4,3.9,1.7,0.4,Iris-setosa
7 4.6,3.4,1.4,0.3,Iris-setosa
8 5.0,3.4,1.5,0.2,Iris-setosa
9 4.4,2.9,1.4,0.2,Iris-setosa
10 4.9,3.1,1.5,0.1,Iris-setosa
11 5.4,3.7,1.5,0.2,Iris-setosa
12 4.8,3.4,1.6,0.2,Iris-setosa
13 4.8,3.0,1.4,0.1,Iris-setosa
14 4.3,3.0,1.1,0.1,Iris-setosa
15 5.8,4.0,1.2,0.2,Iris-setosa
```


一般数据文件的读取方法

```
with open(file,"r",encoding="utf-8") as f:
    lines = f.read().splitlines()

# 取 lab 维度为 N x 1
labs = [line.split("\t")[-1] for line in lines]
labs = np.array(labs).astype(np.float32)

# 取数据 增加 一维全是1的特征
datas = [line.split("\t")[:-1] for line in lines]
datas = np.array(datas).astype(np.float32)
```

简单绘图

	<code>import matplotlib.pyplot as plt</code>
画散点	<code>plt.scatter(sub_datas[:,1],sub_datas[:,2],s=16.,color=dic_colors[i])</code>
	横坐标 纵坐标 图标大小 颜色
画线	<code>plt.plot(x,y)</code>
保存	<code>plt.savefig(name)</code>