- 机器学习算法及其实践

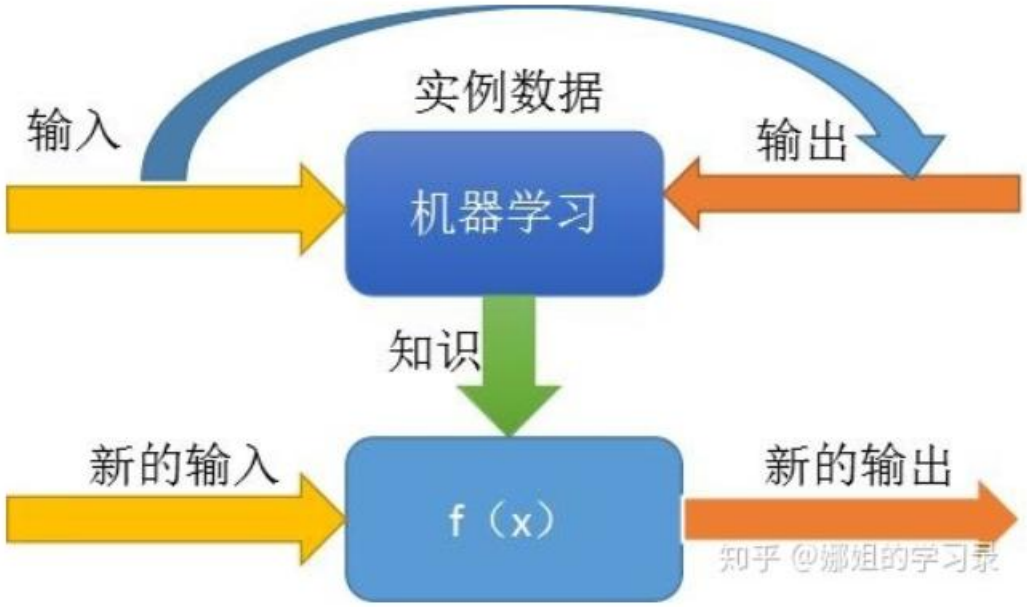
课程要求：
1）python基础
2）已经实现了特征提取、特征筛选
3）进行了缺省值、异常值处理

此时，所有的数据全部变为了特征，而不是fMRI信号了。

机器学习

机器学习就是给定一定的输入，通过施加一定的算法，得到输出，然后通过学到的知识，输入新的数据，获得新的输出。

1）提出问题
2）理解数据
3）特征提取
4）构建模型
5）模型解释



Regression（回归）

Output：离散的值

股票预测：

f（之前的股票信息） = 之后的股票信息

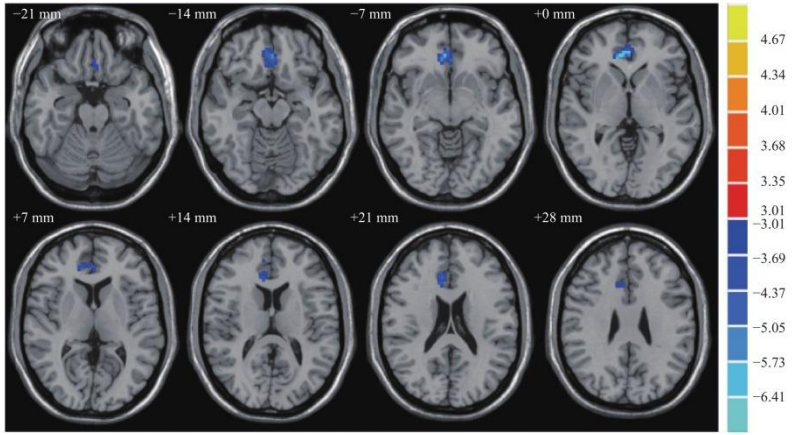
自动驾驶

f（传感器信息） = 方向盘的角度

PM2.5预测

f（其他天气信息） = PM2.5

有态度、有深度、有温度

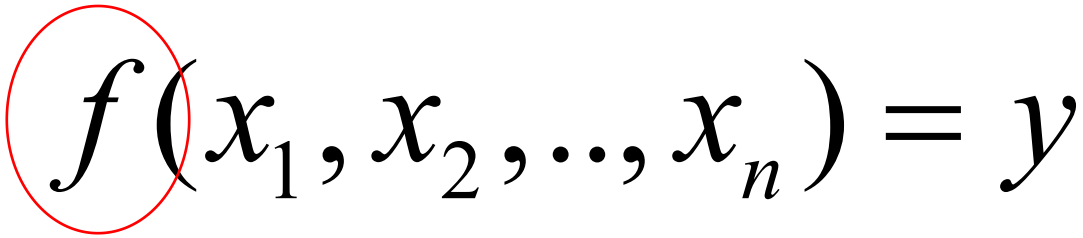
f(

)= Value

Value可以是量化被试的情绪状态或者抑郁水平

抽象：
已知X1,X2,…,Xn和y，求解f

$$f(x_1, x_2, .., x_n) = y$$



Machine learning的三大步骤：
第一步：定义一个函数集合(define a function set)
第二步：判断函数的好坏(goodness of a function)
第三步：选择最好的函数(pick the best one)

Step1: 定义一个函数集合(define a function set)

定义函数集合为：

$$y = \sum \omega_i x_i + b$$

$$y = \sum \omega_i x_i + b = w \bullet x$$

W和b都是模型的参数，可以为任意值。
不同的w和不同b会构成不同的模型。

Y = 0.2x1+0.4x2+0.4
Y = 0.1x1-0.14x2+3.87
Y = 14x1-27x2+0.4
……

显然y是一个线性模型。Wi称为模型的权重（weight），b称为偏置(bias)。
Xi是特征。有多少个特征就有多少个x和w，但只有一个b。

Step2: 判断函数的好坏(goodness of a function)

1）收集Training data （x:特征提取和特征筛选, y:label）

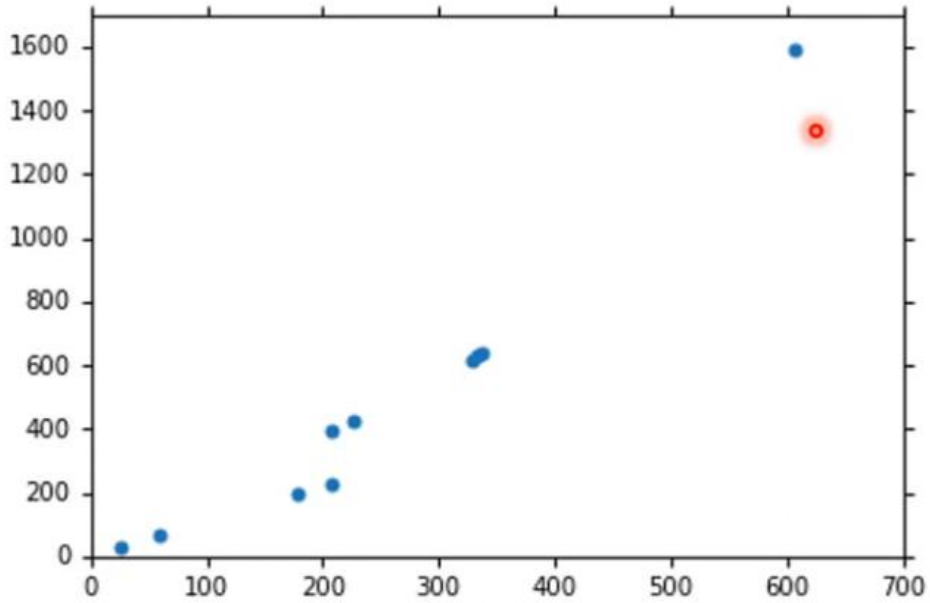
训练集:
X（实际的特征）
$\hat{y}$（实际的label）

$$f(x_1, x_2, .., x_n)$$

Y （模型输出的label）

许多个训练数据

$$(x^1, \hat{y}^1)$$
$$(x^2, \hat{y}^2)$$
$$\vdots$$
$$(x^{10}, \hat{y}^{10})$$



This is real data.

Step2: 判断函数的好坏(goodness of a function)

2)构建损失函数(Loss function)

Loss function是一个特殊的函数， Input: a function, output: how bad it is

$(x^1, \hat{y}^1)$

$(x^2, \hat{y}^2)$

$\vdots$

$(x^{10}, \hat{y}^{10})$

This is real data.

针对： $y = \sum \omega_i x_i + b = w \cdot x + \mathrm{b}$

$$L(f) = \sum_{n=1}^{10} (\hat{y}_n - f(x_n))^2$$

$$= \sum_{n=1}^{10} (\hat{y}_n - (w \cdot x_n + b))^2$$

$$= L(w, b)$$

训练样本的x

遍历所有训练样本　　真实的y　　预测的y

该损失函数被称为“均方误差”。MSE（Mean square error）

第三步：选择最好的函数(pick the best one)

损失函数：$$L(w,b)=\sum_{n}(\hat{y}_n-(w\bullet x_n+b))^2$$

Q1：损失函数是用来量化什么的?
Q2：什么样的f是最好的f?

优化目标：$$\min L(w,b) \xrightarrow{\text{什么样的w和b会导致L最小}} \operatorname{argmin} L(w,b)$$

最终需要求解：$$w^*,b^*=\arg\min L(w,b)$$

$$=\arg\min\sum_{n=1}^{10}(\hat{y}_n-(w\bullet x_n+b))^2$$

第三步：选择最好的函数(pick the best one)

$$w^*, b^* = \arg\min L(w,b)$$

$$= \arg\min \sum_{n=1}^{10} (\hat{y}_n - (w \bullet x_n + b))^2$$

如果求解w,b?
不同模型方法是不同的，但机器学习中最常用的一种方法叫梯度下降。

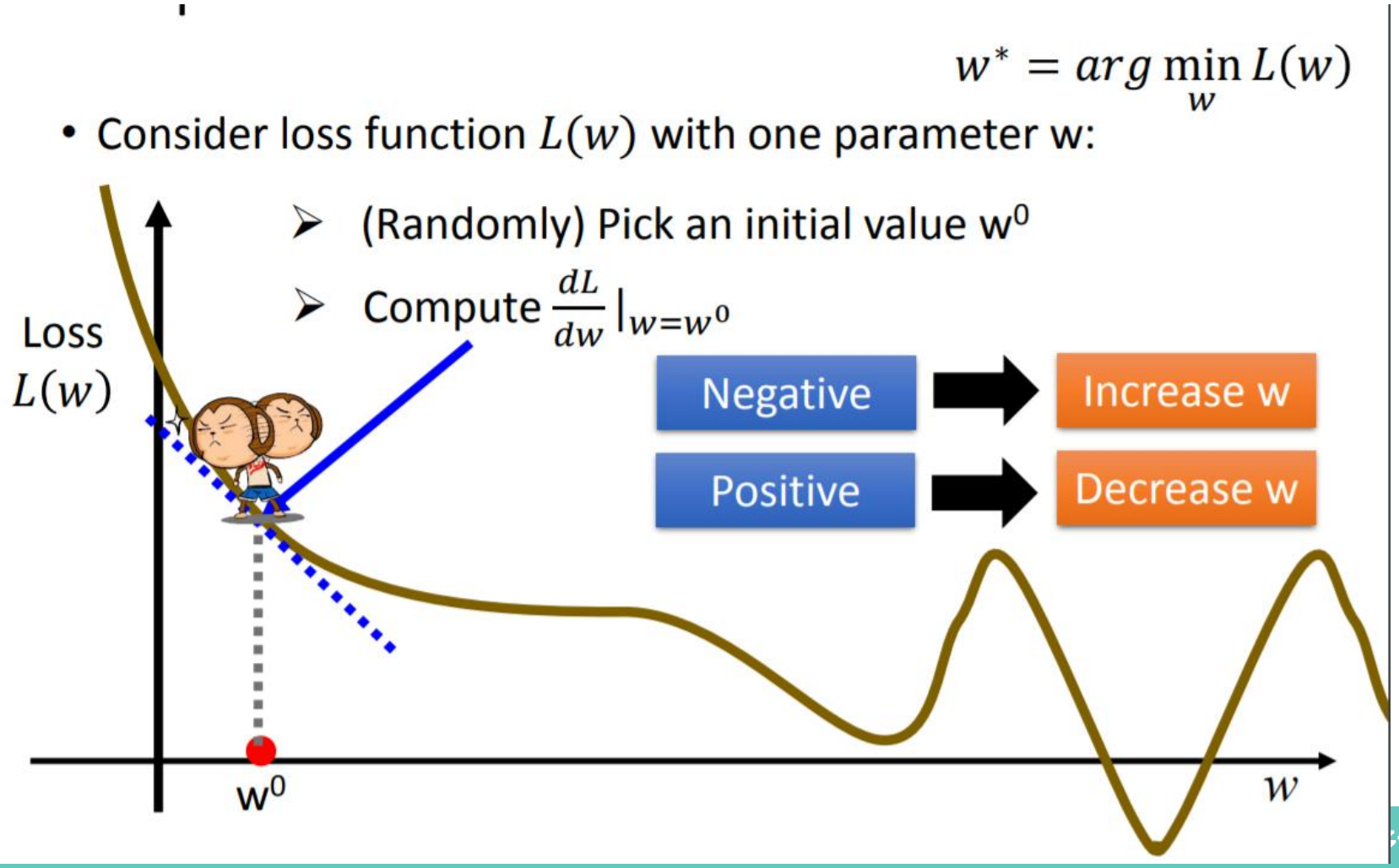
每个模型都有自己的损失函数，不管是监督式学习还是非监督式学习。损失函数包含了若干个位置的模型参数，比如在多元线性回归中，损失函数均方误差，我们就是要找到使损失函数尽可能小的参数未知模型参数。

在简单线性回归时，我们使用最小二乘法来求损失函数的最小值，但是这只是一个特例。在绝大多数的情况下，损失函数是很复杂的(比如逻辑回归)，根本无法得到参数估计值的表达式。因此需要一种对大多数函数都适用的方法。这就引出了“梯度算法”。

第三步：选择最好的函数(pick the best one)

梯度下降

多元函数的导数(derivative)就是梯度(gradient)。



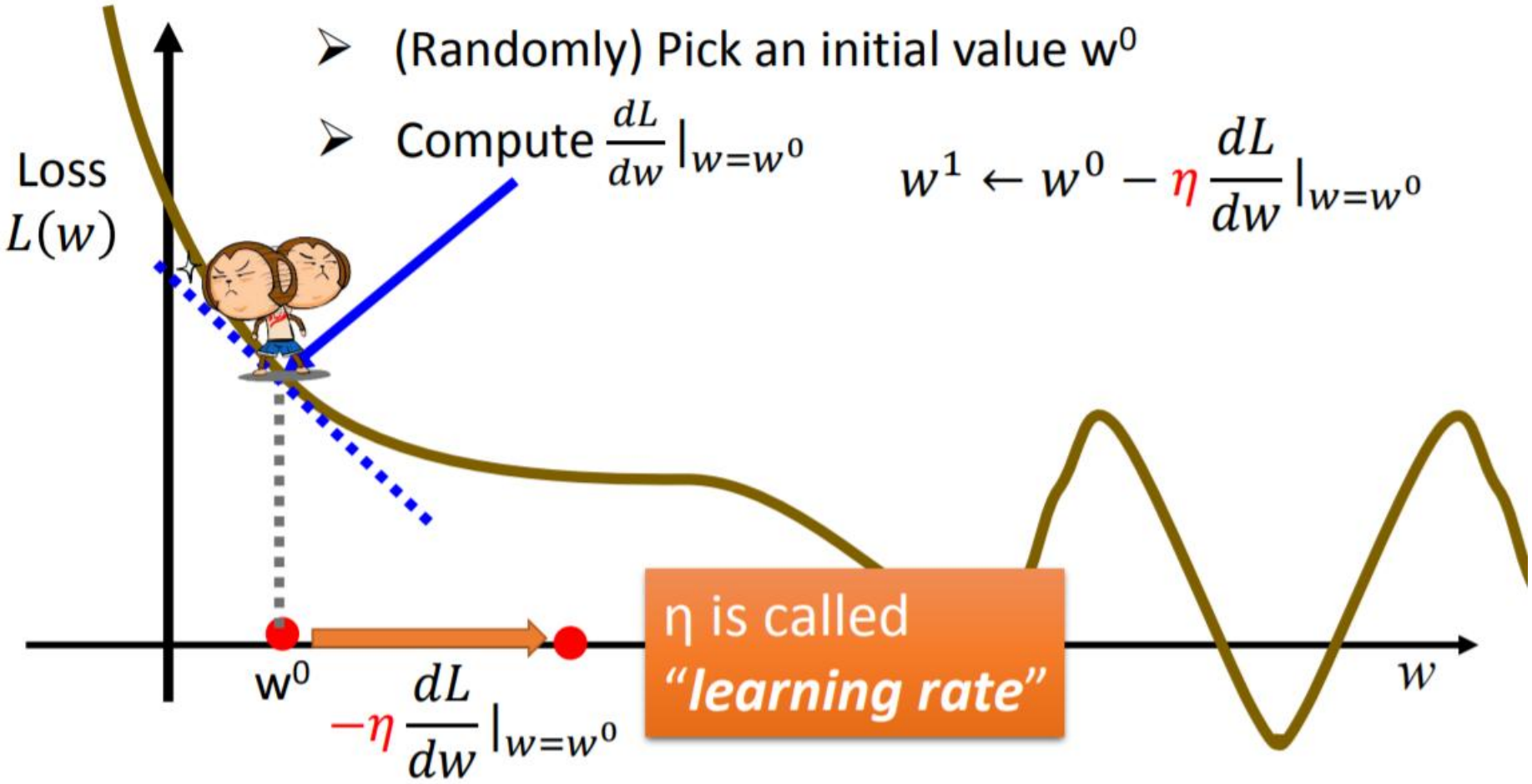
第三步：选择最好的函数(pick the best one)

$$w^* = arg\min_w L(w)$$

梯度下降

- Consider loss function $L(w)$ with one parameter w:
  - (Randomly) Pick an initial value w^0
  - Compute $\frac{dL}{dw}|_{w=w^0}$

$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$$

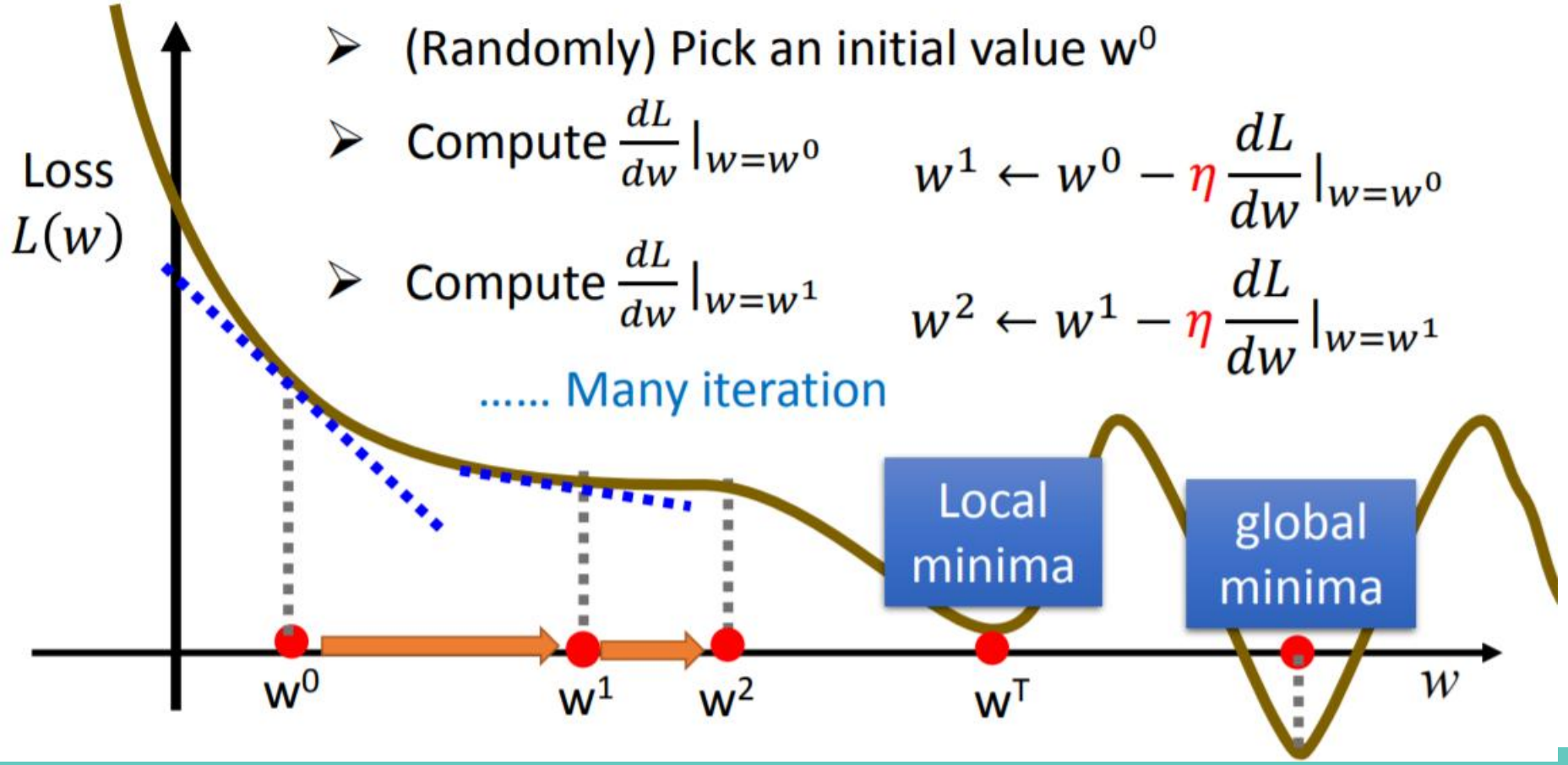


第三步：选择最好的函数(pick the best one)

梯度下降

$$w^* = arg\min_w L(w)$$

- Consider loss function $L(w)$ with one parameter w:



$$\begin{bmatrix} \frac{\partial L}{\partial w} \\ \frac{\partial L}{\partial b} \end{bmatrix}$$ gradient

第三步：选择最好的函数(pick the best one)

- How about two parameters? $w^*, b^* = arg \min_{w,b} L(w,b)$
  - (Randomly) Pick an initial value w^0, b^0
  - Compute $\frac{\partial L}{\partial w}|_{w=w^0,b=b^0}, \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$

$$w^1 \leftarrow w^0 - \eta \frac{\partial L}{\partial w}|_{w=w^0,b=b^0} \qquad b^1 \leftarrow b^0 - \eta \frac{\partial L}{\partial b}|_{w=w^0,b=b^0}$$

  - Compute $\frac{\partial L}{\partial w}|_{w=w^1,b=b^1}, \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$

$$w^2 \leftarrow w^1 - \eta \frac{\partial L}{\partial w}|_{w=w^1,b=b^1} \qquad b^2 \leftarrow b^1 - \eta \frac{\partial L}{\partial b}|_{w=w^1,b=b^1}$$

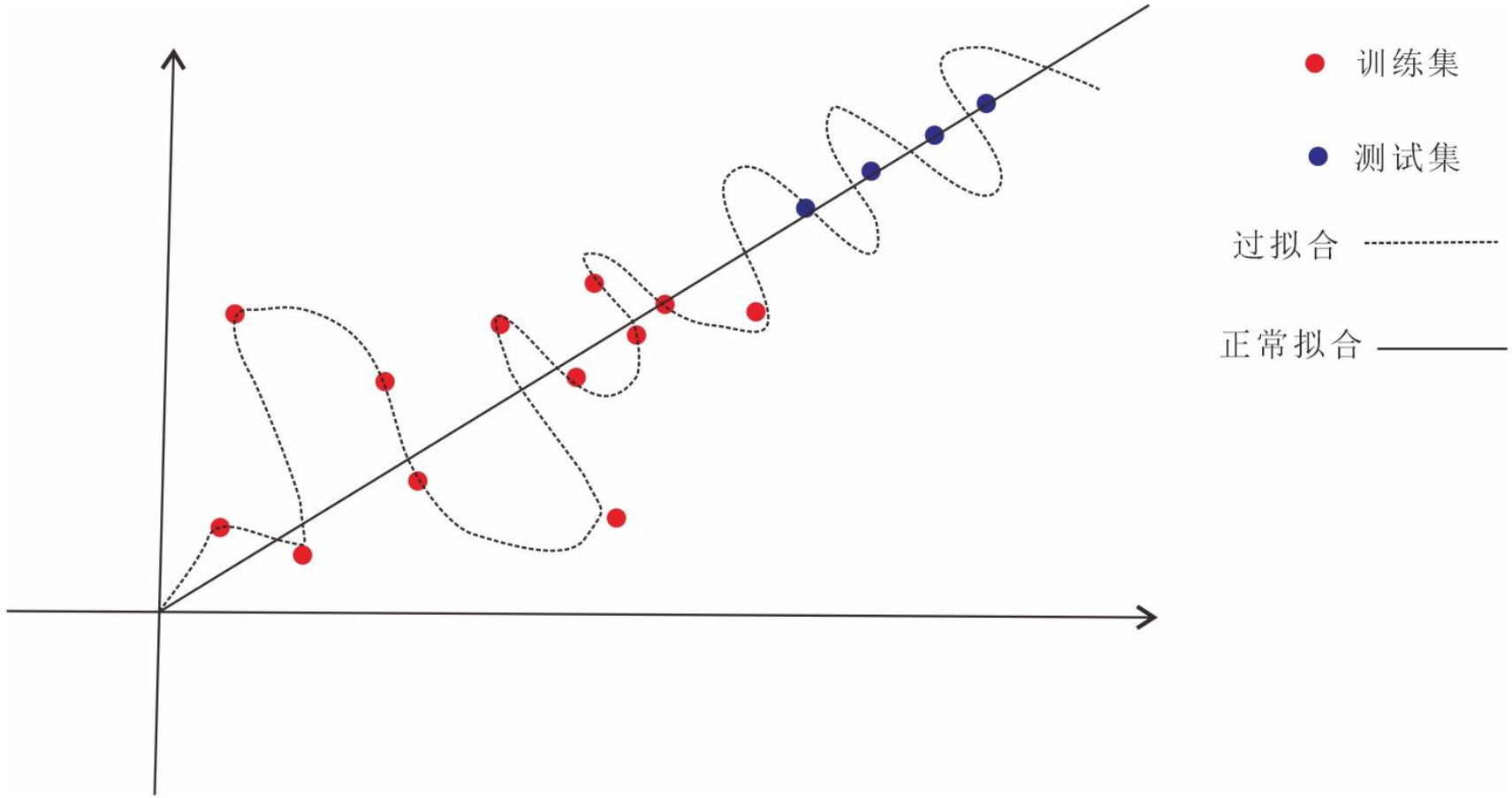
第三步：选择最好的函数(pick the best one)

此时，已经找到了好的w，b。就构成了一个线性模型：

$$y = \sum \omega_i x_i + b = w \bullet x + b$$

接着，在测试集上测试，如果测试集上的准确率较好，该模型就被训练好了。

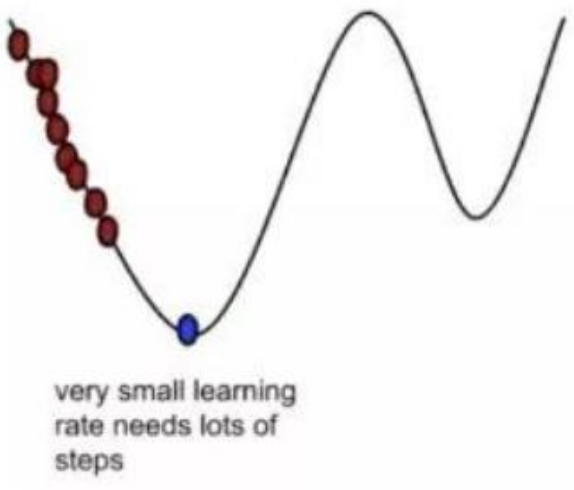
一个重要问题：过拟合（在训练集上表现好，在测试集上表现差）



$$w^1 \leftarrow w^0 - \eta \frac{dL}{dw}|_{w=w^0}$$

另一个重要问题：局部最小

从理论上，它只能保证达到局部最低点，而非全局最低点。在很多复杂函数中有很多极小值点，我们使用梯度下降法只能得到局部最优解，而不能得到全局最优解。
对于梯度下降来说，初始点的位置，也是一个超参数。



线性回归

线性回归是回归问题中的一种，线性回归假设目标值与特征之间线性相关，即满足一个多元一次方程。
通过构建损失函数，来求解损失函数最小时的参数w和b。

目标函数：$$y = \sum \omega_i x_i + b$$

损失函数：$$L(f) = \frac{1}{2n} \sum_n (\hat{y_n} - f(x_n))^2$$

优化目标：

$$(w^*, b^*) = \arg\min_{(w,b)} \sum_{i=1}^{n} (wx_i + b - y_i)^2$$

$$\frac{\partial L}{\partial w} = \left(w \sum_{i=1}^{n} x^2 - \sum_{i=1}^{n} x_i (y_i - b) \right)$$

$$\frac{\partial L}{\partial b} = \left(nb - \sum_{i=1}^{n} (y_i - w x_i) \right)$$

$$w \leftarrow w - \alpha \frac{\partial L}{\partial w}$$

$$b \leftarrow b - \alpha \frac{\partial L}{\partial b}$$

每次把n个训练数据代入，计算一次导入，更新一次权重。

- 请大家完成下列软件库的软件：
- conda install scikit-learn
- conda install matplotlib
- conda install numpy

线性回归Python函数调用：sklearn.linear_model.LinearRegression

Doc： https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

训练过程：

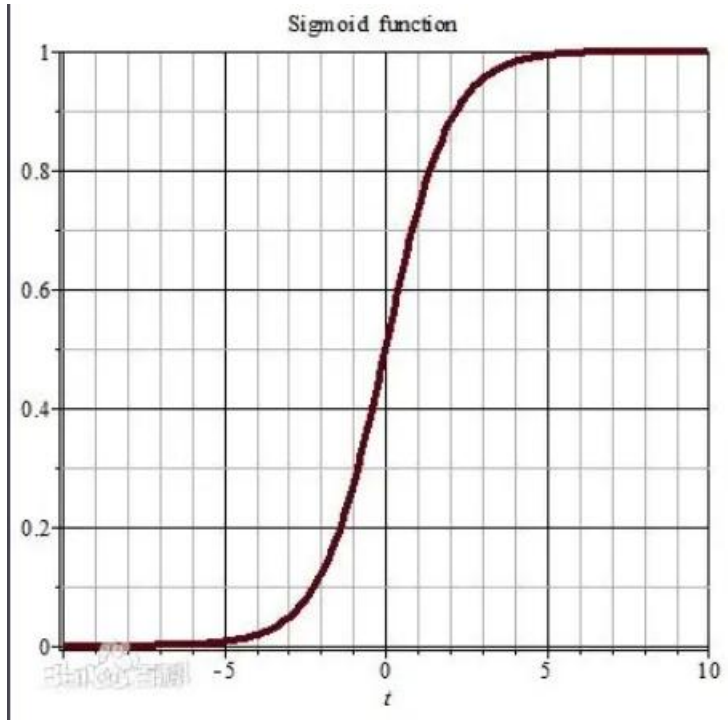
```
import numpy as np
from sklearn.linear_model import LinearRegression
x = np.array([[1, 1], [1, 2], [2, 2], [2, 3]])
# y = 1 * x_0 + 2 * x_1 + 3
y = np.dot(x, np.array([1, 2])) + 3
linear_regression_model = LinearRegression()
linear_regression_model.fit(x,y)
linear_regression_model.score(x, y)
```

测试过程：

```
x_test = ([[5, 5], [10, 10]])
y_test = np.dot(x_test, np.array([1, 2])) + 3
linear_regression_model.predict(x_test)
test_score = linear_regression_model.score(x_test, y_test)
```

逻辑回归（logistic回归，一种分类算法，并常用于二分类）

逻辑回归是在线性回归的基础上加了一个 Sigmoid 函数（非线形）映射。



把输出（Y）从实数范围拉回了[0,1]，有时候这个输出也被称为概率P。

设定阈值后，即可得到分类结果。

假定阈值为0.5
如果P=0.6，则分类为正样本。
如果P=0.2，则分类为负样本。

逻辑回归能分析分析**非线性问题。**

有态度、有深度、有温度

线性回归的损失函数：

$$L(f)=\frac{1}{2n}\sum_{n}(\hat{y_n}-f(x_n))^2$$

均方误差：
回归问题

逻辑回归的损失函数：

$$L(w)=\prod[p(x_i)]^{y_i}[1-p(x_i)]^{1-y_i}$$

交叉熵： 如果相同就为0.
分类问题

为了更方便求解，我们对等式两边同取对数，写成对数似然函数：

$$\begin{aligned}L(w)&=\sum[y_i ln p(x_i)+(1-y_i)ln(1-p(x_i))]\\&=\sum[y_i ln\frac{p(x_i)}{1-p(x_i)}+ln(1-p(x_i))]\\&=\sum[y_i(w\cdot x_i)-ln(1+e^{w\cdot x_i})]\end{aligned}$$

逻辑回归的优点

- 线性回归是在**实数域范围内**进行预测，而**分类范围则需要在 [0,1]，逻辑回归减少了预测范围；**
- 线性回归在实数域上敏感度一致，而逻辑回归在 0 附近敏感，在远离 0 点位置不敏感，这个的好处就是模型更加关注分类边界，可以增加模型的鲁棒性。

为什么逻辑回归不使用均方误差?
主要是梯度消失

假设目标函数是 MSE，即：

$$L = \frac{(y-\hat{y})^2}{2}$$

$$\frac{\partial L}{\partial w} = (\hat{y}-y)\sigma'(w\cdot x)x$$

这里 Sigmoid 的导数项为：

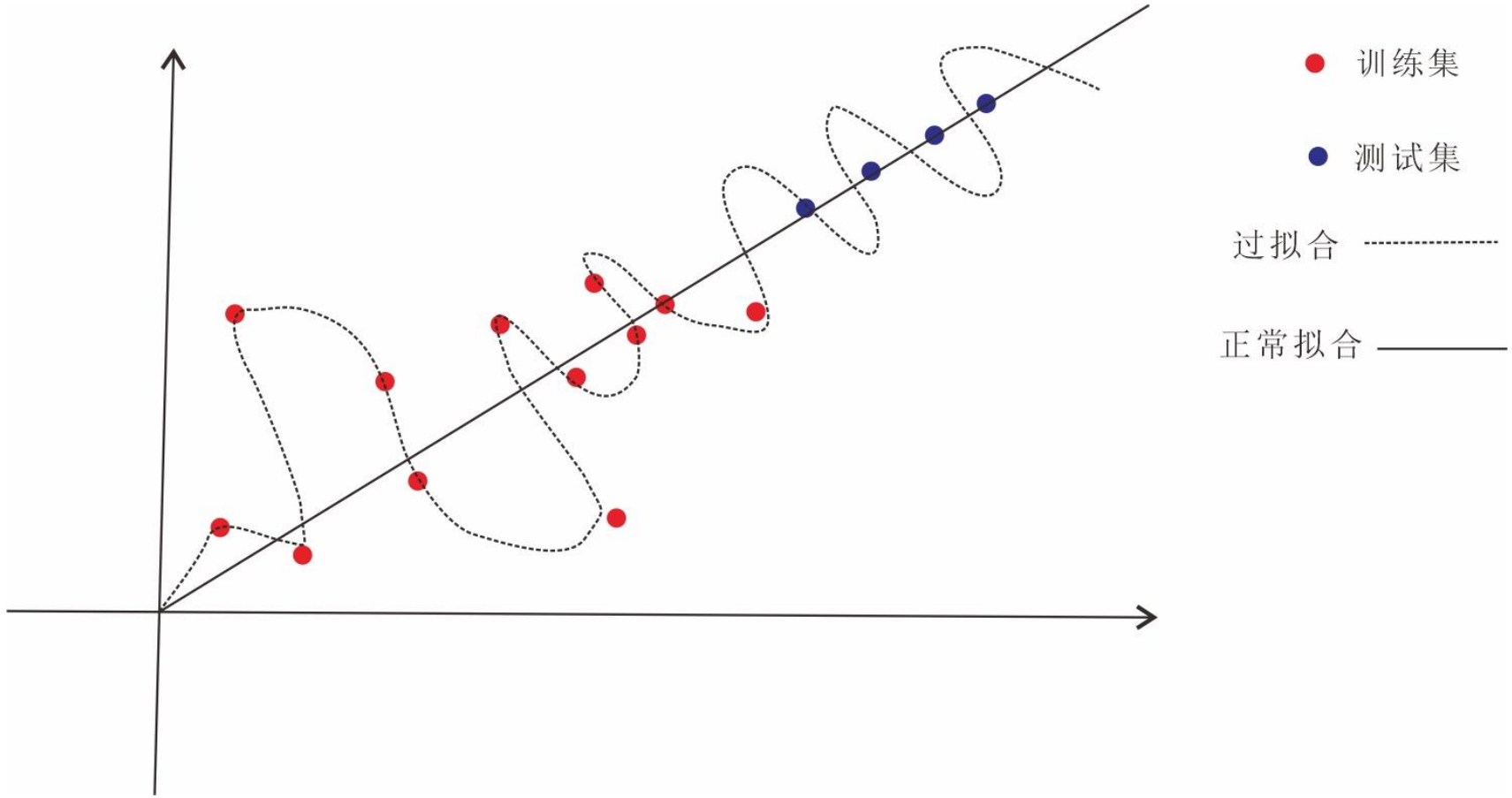
$$\sigma'(w\cdot x) = w\cdot x(1-w\cdot x)$$

根据 w 的初始化，导数值可能很小（想象一下 Sigmoid 函数在输入较大时的梯度）而导致收敛变慢，而训练途中也可能因为该值过小而提早终止训练（梯度消失）。

另一方面，交叉熵的梯度如下，当模型输出概率偏离于真实概率时，梯度较大，加快训练速度，当拟合值接近于真实概率时训练速度变缓慢，没有 MSE 的问题。

$$g' = \sum_{i=1}^{N} x_i(y_i - p(x_i))$$

一个重要问题：过拟合（在训练集上表现好，在测试集上表现差）



正则化方法：防止模型过拟合

从逻辑回归开始，提到的分类器就不是线性分类器了，就可能存在过拟合。**通过正则化方法来防止过拟合。**

改变模型的Loss function：

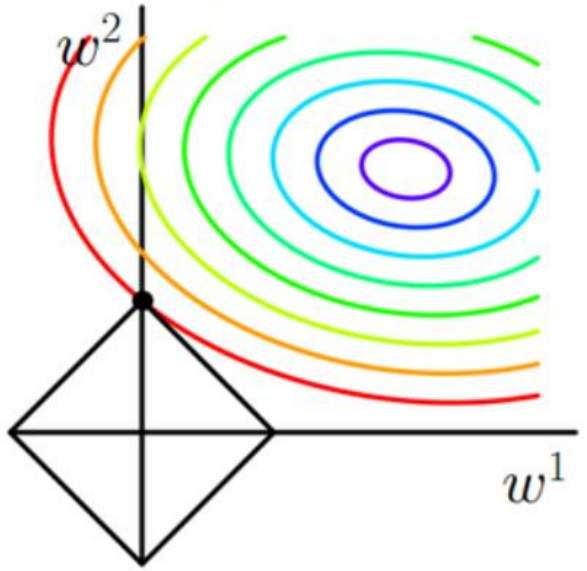
$$L = L + \lambda \cdot f(w_i + b)$$

原始的损失函数　　正则化方法

L1正则化

$$L = L + \lambda |w|$$

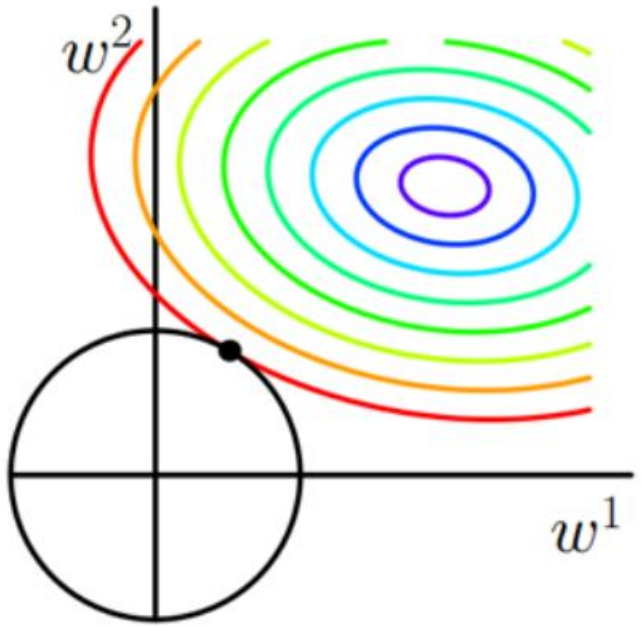
L1正则化可以产生稀疏模型，即让部分权重等于0，方便提取无用特征。



L2正则化

$$L = L + \lambda |w|^2$$

L2正则化可以使权重平滑，防止过拟合。



逻辑回归Python函数调用：
sklearn.linear_model.LogisticRegression
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html

```python
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
data = load_iris()
#数据集划分
x = data['data']
y = data['target']
x_train, x_test ,y_train, y_test = train_test_split(x,y,test_size = 0.2)
clf = LogisticRegression().fit(x_train,y_train)
clf.score(x_test,y_test)
```

绘制ROC


```python
#绘制ROC曲线
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

y_label = ([0, 0, 0, 1, 1, 1])
y_pre = ([0.3, 0.5, 0.9, 0.8, 0.4, 0.6])

# y_label = y_test
# y_pre_proba = clf.predict_proba(x_test)
# y_pre = y_pre_proba[:,1]
fpr, tpr, thersholds = roc_curve(y_label, y_pre)
for i, value in enumerate(thersholds):
    print("%f %f %f" % (fpr[i], tpr[i], value))
roc_auc = auc(fpr, tpr)
plt.plot(fpr, tpr, 'k--', label='ROC (area = {0:.2f})'.format(roc_auc), lw=2)
plt.xlim([-0.05, 1.05])  # 设置x、y轴的上下限，以免和边缘重合，更好的观察图像的整体
plt.ylim([-0.05, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')  # 可以使用中文，但需要导入一些库即字体
plt.title('ROC Curve')
plt.legend(loc="lower right")
```

思考：如何写代码计算准确率?

```
y_label = y_test
y_pre = clf.predict(x_test)
```

```
# acc = np.sum(y_pre == y_label)/len(y_pre)
```

总结：线性回归和逻辑回归

模型主要做什么?	回归问题	分类问题
损失函数?	均方误差	交叉熵
求解方法?	最小二乘法/梯度下降	梯度下降

朴素贝叶斯（Naive Bayes，NB）-- 基于贝叶斯定理分类算法

1）算法数学原理清晰，应用广泛。
2）算法稳定。
3）容易解释。

为什么“朴素”？

它假定所有的特征在数据集中的作用是同样重要和独立的,正如我们所知,这个假设在现实世界中是很不真实的,因此,说是很“朴素的”。

贝叶斯公式：

$$P(B|A) = \frac{P(A|B)P(B)}{P(A)}$$

$$\text{p(类别|特征)} = \frac{p(\text{特征|类别})\text{p(类别)}}{\text{p(特征)}}$$

需要求解的目标

朴素贝叶斯（Naive Bayes，NB）-- 基于贝叶斯定理

优点：

1）朴素贝叶斯模型发源于古典数学理论，**有稳定的分类效率。**

2）对**小规模的数据表现很好**，能个处理多分类任务，适合增量式训练，尤其是数据量超出内存时，我们可以一批批的去增量训练。

3）对**缺失数据不太敏感**，算法也比较简单，常用于文本分类。

缺点：

1）理论上，朴素贝叶斯模型与其他分类方法相比具有最小的误差率。但是实际上并非总是如此，这是**因为朴素贝叶斯模型假设特征之间相互独立**，这个假设在实际应用中往往是不成立的，在属性个数比较多或者属性之间相关性较大时，分类效果不好。

2）需要知道**先验概率**，且先验概率很多时候取决于假设，假设的模型可以有很多种，因此在某些时候会由于假设的先验模型的原因导致预测效果不佳。

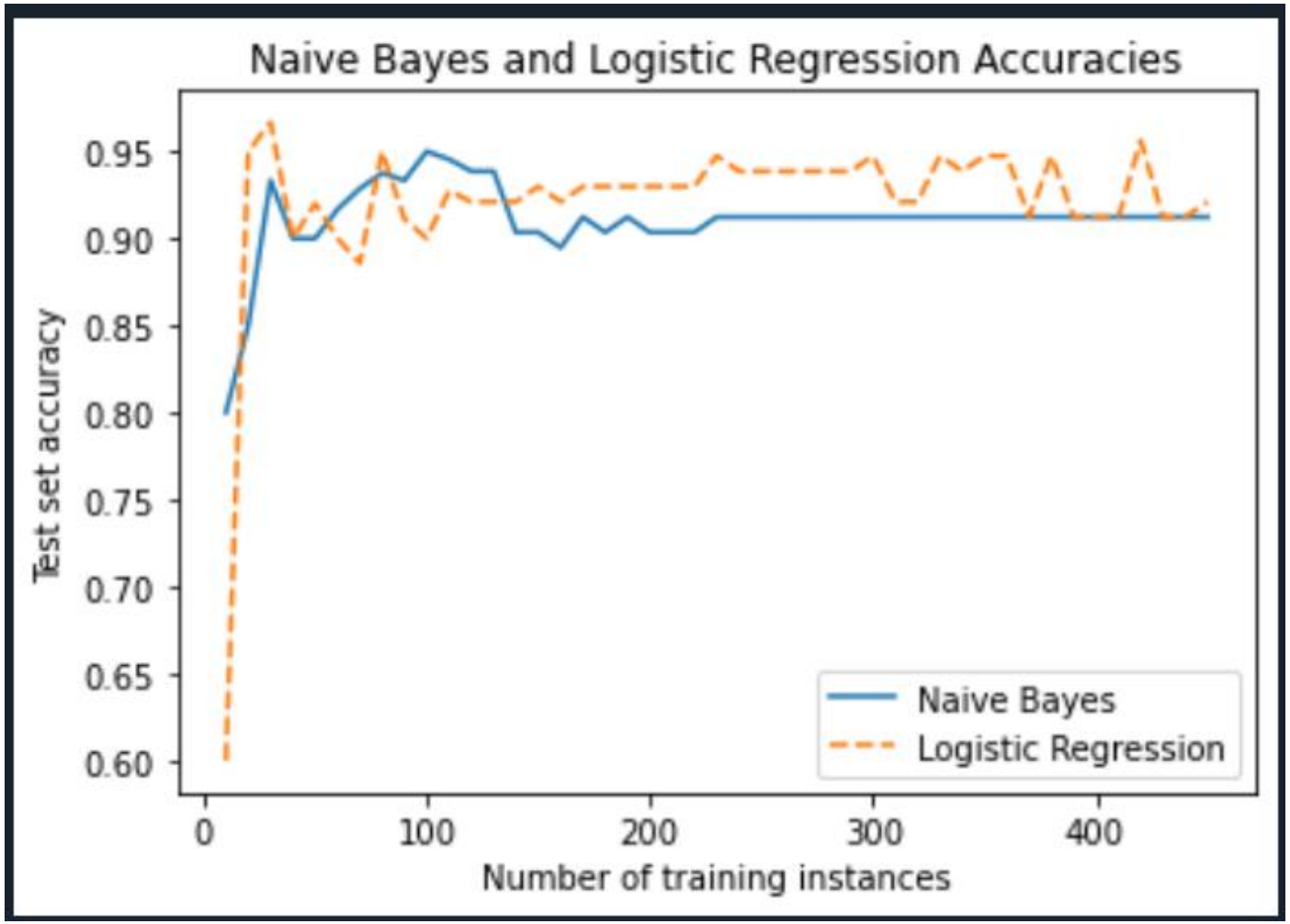
Python调用：sklearn.naive_bayes.GaussianNB

观察不同样本量下，贝叶斯和逻辑回归的效果

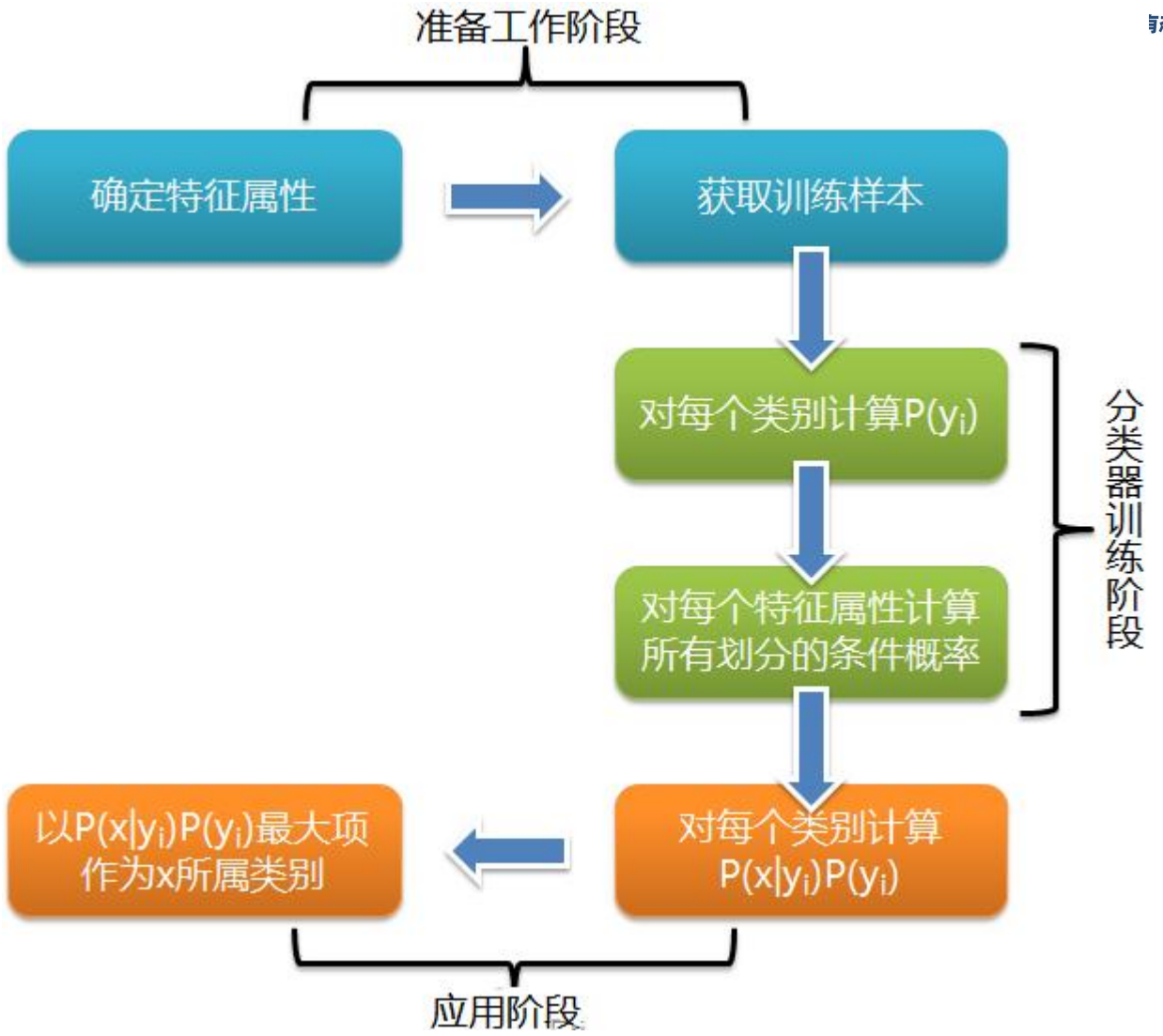

```python
#逻辑回归与贝叶斯分类器对比
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
import matplotlib.pyplot as plt

data = load_breast_cancer()
x = data['data']
y = data['target']
x_train, x_test ,y_train, y_test = train_test_split(x,y,test_size = 0.2)

lr = LogisticRegression()
nb = GaussianNB()

print('========不断提高训练的样本数据量=======')
train_sizes = range(10, len(x_train), 10)
lr_scores_list = []
nb_scores_list = []
```

```python
for train_size in train_sizes:
    train_x_slice = x_train[:train_size]
    train_y_slice = y_train[:train_size]
    test_x_slice = x_test[:train_size]
    test_y_slice = y_test[:train_size]

    nb.fit(train_x_slice, train_y_slice)
    nb_socer = nb.score(test_x_slice,test_y_slice)
    nb_scores_list.append(nb_socer)

    lr.fit(train_x_slice, train_y_slice)
    lr_socer = lr.score(test_x_slice,test_y_slice)
    lr_scores_list.append(lr_socer)


plt.plot(train_sizes, nb_scores_list, label='Naive Bayes')
plt.plot(train_sizes, lr_scores_list, linestyle='--', label='Logistic Regression')
plt.title("Naive Bayes and Logistic Regression Accuracies")
plt.xlabel("Number of training instances")
plt.ylabel("Test set accuracy")
plt.legend()
```

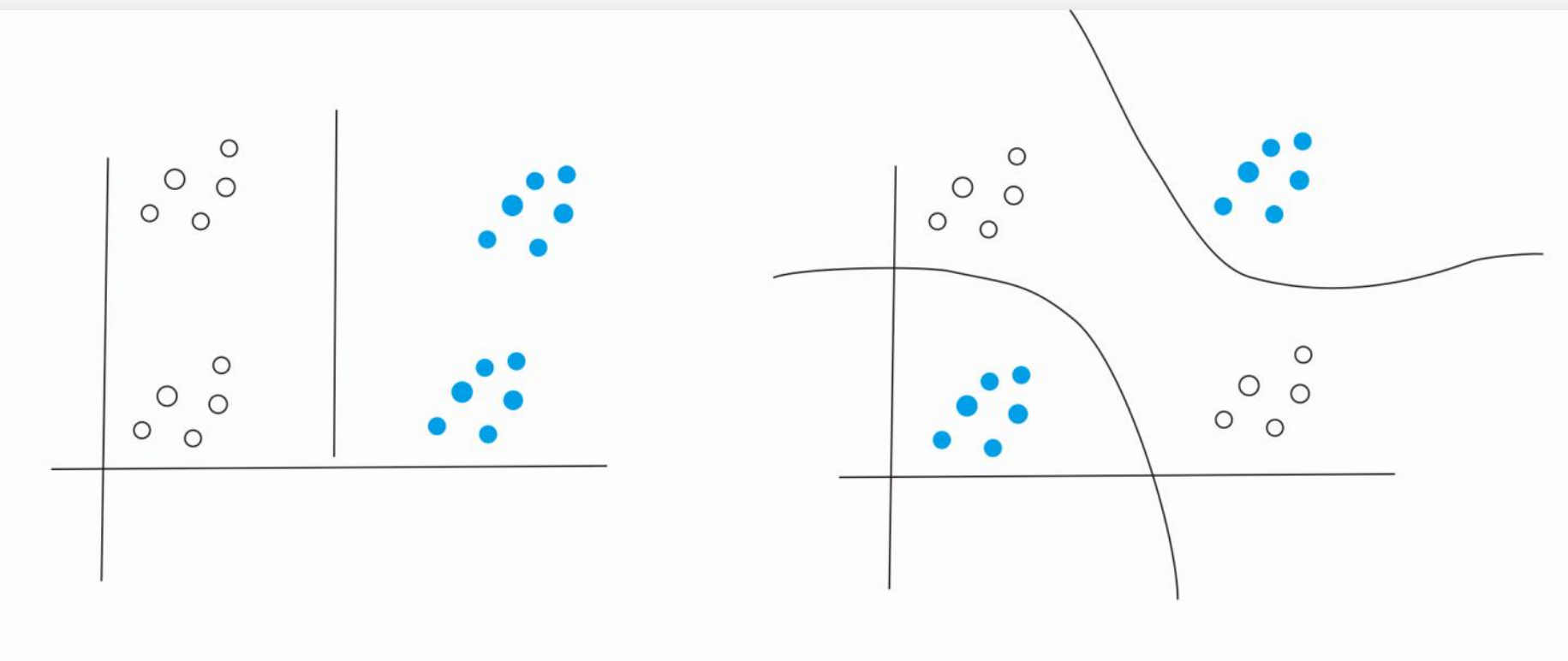
https://scikit-learn.org/0.15/modules/generated/sklearn.naive_bayes.GaussianNB.html#sklearn.naive_bayes.GaussianNB


Naive Bayes and Logistic Regression Accuracies
Test set accuracy
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0
100
200
300
400
Number of training instances
Naive Bayes
Logistic Regression

有态度、有深度、有温度



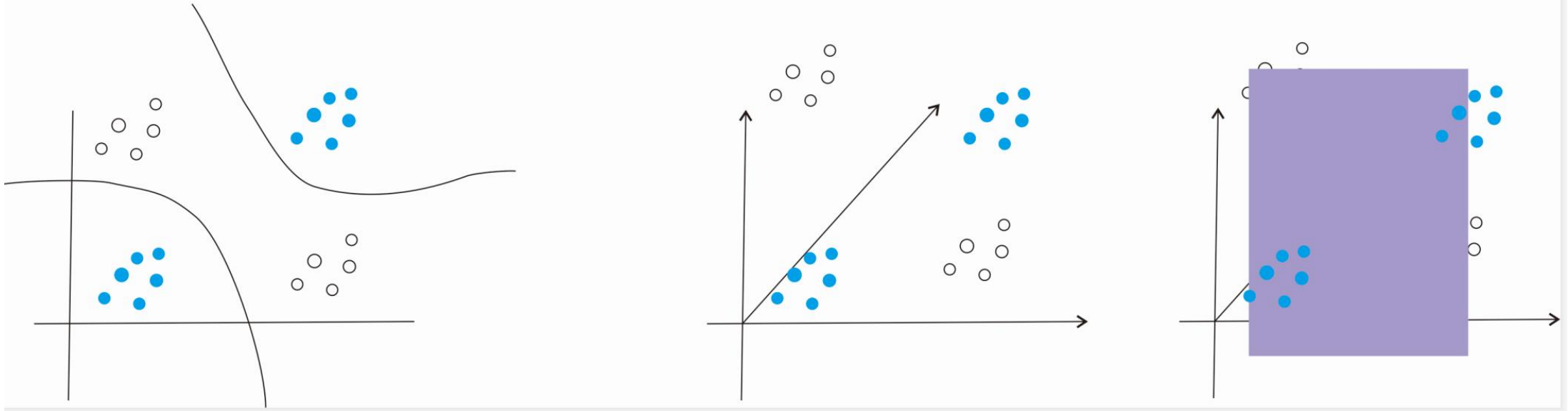
支持向量机（Support Vector Machine, SVM）---- **重点模型**



线性可分

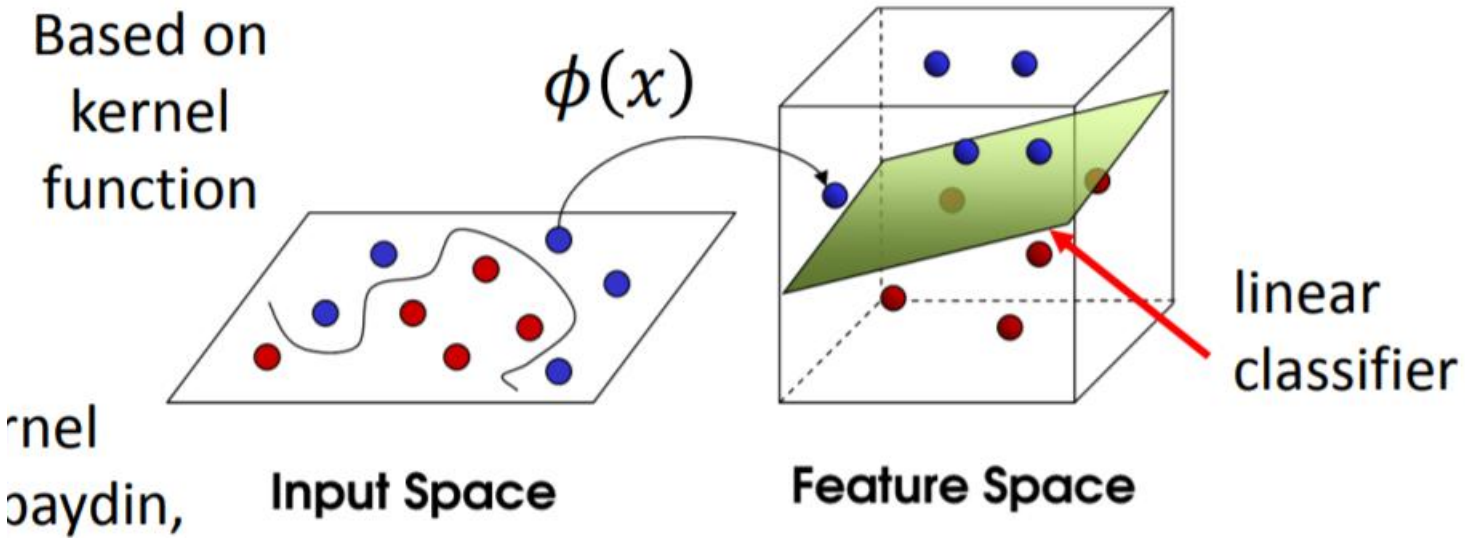
线性不可分

Kernel trick



把低纬数据映射到高维，从而解决数据的线性不可分问题。
SVM的一大核心： Kernel trick

Kernel trick



把低纬数据映射到高维，从而解决数据的线性不可分问题。
SVM的一大核心：Kernel trick

Kernel trick

线性核（Linear Kernel）

$$k(x,y)=x^{T}y+c$$

多项式核（Polynomial Kernel）

$$k(x,y)=(ax^{T}y+c)^{d}$$

径向基核函数（Radial Basis Function）

$$k(x,y)=\exp\left(-\gamma\|x-y\|^{2}\right)$$

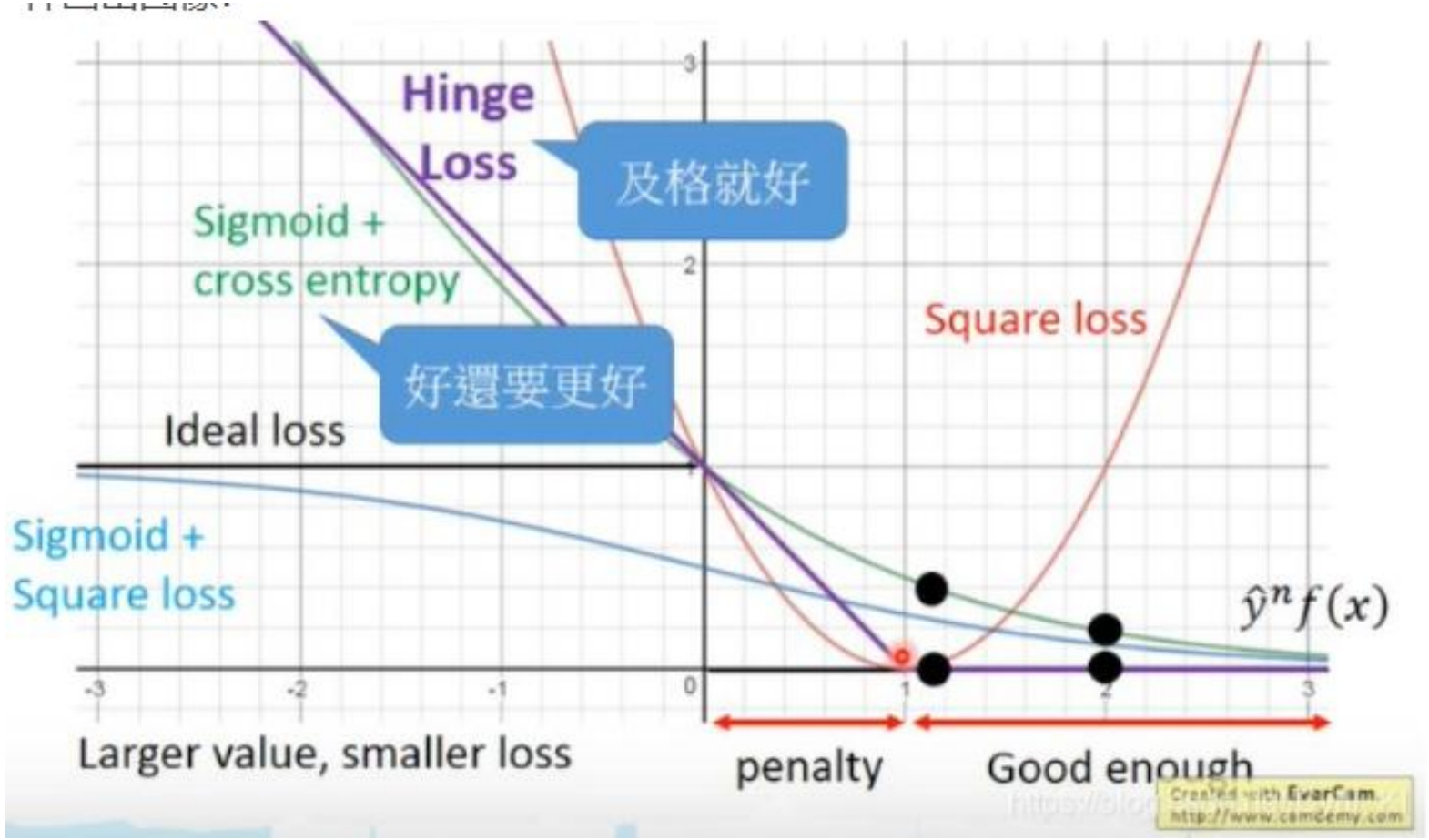
也叫**高斯核（Gaussian Kernel）**，因为可以看成如下核函数的领一个种形式：

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

Kernel trick把数据从低纬映射到高维，从而实现数据可分。

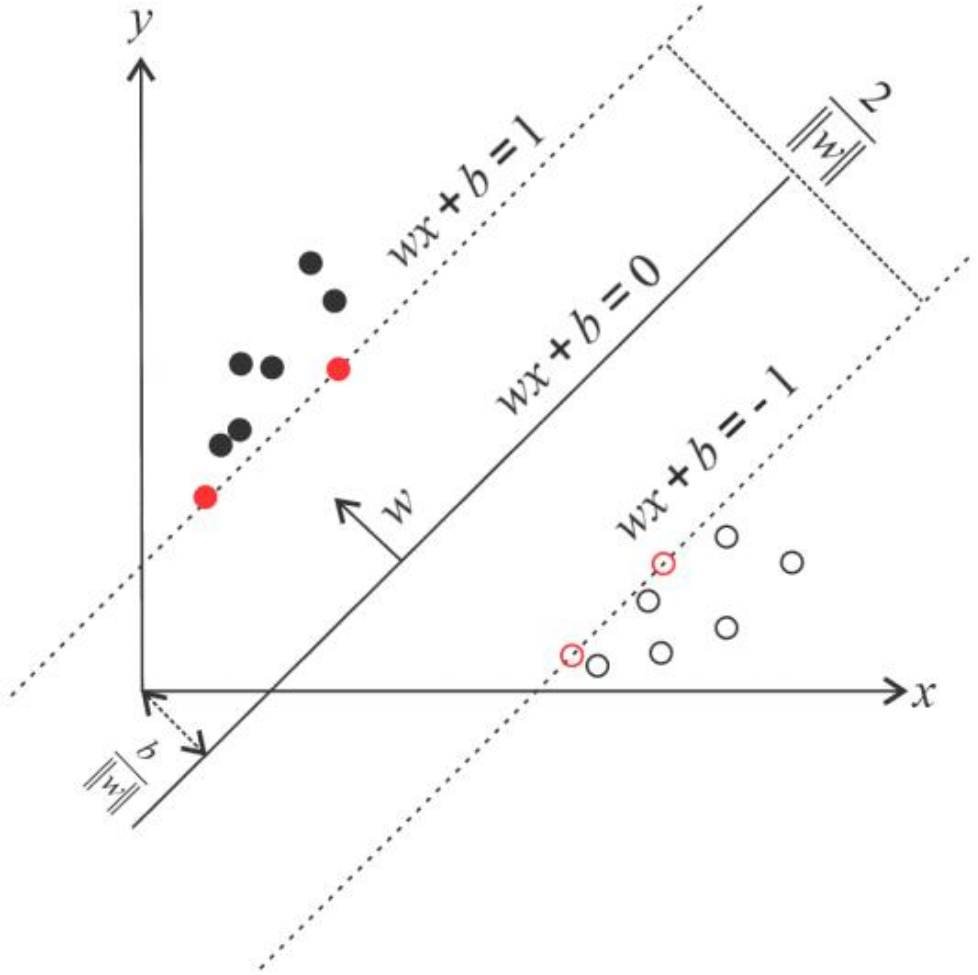
一般建议采用线性核的SVM，比较容易解释。

Hinge Loss（合页损失函数）



及格就好，不会追求所有数据点都拟合完美，SVM对异常值不敏感。

SVM



支持向量机原理图

SVM的python调用：sklearn.svm.svc

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

调用：

sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)

sklearn.svm.SVC（C=10，kernel=“”）

重要参数：

C：float参数 默认值为1.0 错误项的惩罚系数。C越大，即对分错样本的惩罚程度越大，因此在训练样本中准确率越高，但是泛化能力降低，也就是对测试数据的分类准确率降低。相反，减小C的话，容许训练样本中有一些误分类错误样本，泛化能力强。对于训练样本带有噪声的情况，一般采用后者，把训练样本集中错误分类的样本作为噪声。

{'linear', 'poly', 'rbf', 'sigmoid', 'precomputed'} or callable, default='rbf'

kernel: str参数 默认为 ‘rbf’。

‘linear’: 线性核函数

‘poly’： 多项式核函数

‘rbf’： 径像核函数/高斯核

‘sigmod’:sigmod核函数

‘precomputed’:核矩阵

max_iter： 最大迭代次数，默认为100。

SVM的python调用：sklearn.svm.svc

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

调用：

sklearn.svm.SVC(*, C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None)

svm= sklearn.svm.SVC()

svm.XXX

重要属性：

svc.n_support_：各类各有多少个支持向量

svc.support_：各类的支持向量在训练样本中的索引

svc.support_vectors_：各类所有的支持向量

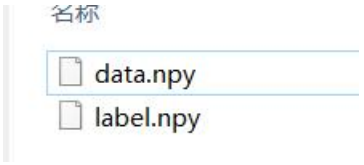
coef_ : ndarray of shape (n_classes * (n_classes - 1) / 2, n_features)

Weights assigned to the features when kernel="linear".

SVM的python调用：sklearn.svm.svc

https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html#sklearn.svm.SVC

1)数据读取：np.load()



2)数据集划分

```
#整体划分
train_data,test_data,train_label,test_label = train_test_split(data,label,test_size=0.2)
#验证集
train_data,val_data,train_label,val_label = train_test_split(train_data,train_label,test_size=0.2)
```

3）通过验证集来调节SVM的参数（主要调整*C=1.0*、*kernel='rbf'*）每次调整完后，要重新创建模型，重新训练 sklearn.svm.svc(c=1.0,kernel='*rbf*')
sklearn.svm.svc(c=10,kernel='*linear*')

```
#模型评估
ACC = Logstic_reg.score(val_data,val_label)
```

4）在测试集上测试结果

```
#计算测试集上的准确率
ACC_test = Logstic_reg.score(test_data,test_label)
```

五折校验验证参数调整sklearn.model_selection.GridSearchCV

https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV

sklearn.model_selection.GridSearchCV(*estimator*, *param_grid*, *, *scoring=None*, *n_jobs=None*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score=nan*, *return_train_score=False*)

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
iris = datasets.load_iris()

parameters = {'kernel':('linear', 'rbf'), 'C':[1,2,3,4,5,6,7,8,9,10]}
svc = svm.SVC()
clf = GridSearchCV(svc, parameters)
clf.fit(iris.data, iris.target)
cv_result = clf.cv_results_
clf.best_params_
```

SVM实现多分类

SVM算法最初是为二值分类问题设计的，当处理多类问题时，就需要构造合适的多类分类器。目前，构造SVM多类分类器的方法主要有两类：
一类是直接法，直接在目标函数上进行修改，将多个分类面的参数求解合并到一个最优化问题中，通过求解该最优化问题“一次性”实现多类分类。这种方法看似简单，但其计算复杂度比较高，实现起来比较困难，只适合用于小型问题中；
另一类是间接法，主要是通过组合多个二分类器来实现多分类器的构造，常见的方法有one-against-one和one-against-all两种。

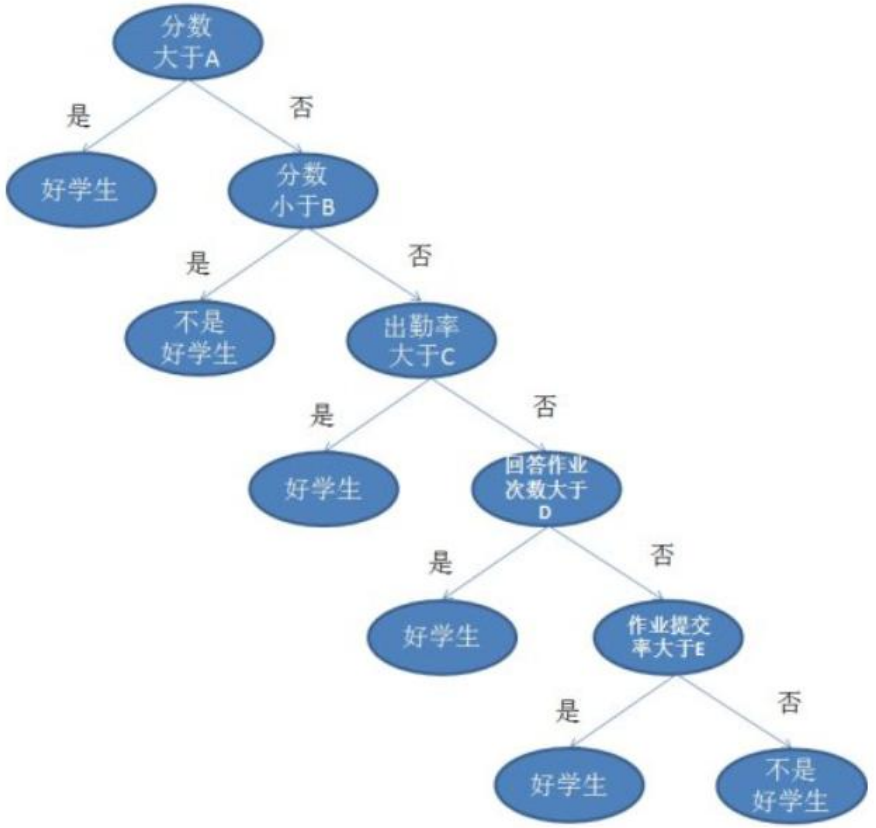
a.一对多法（one-versus-rest,简称1-v-r SVMs）。训练时依次把某个类别的样本归为一类,其他剩余的样本归为另一类，这样k个类别的样本就构造出了k个SVM。分类时将未知样本分类为具有最大分类函数值的那类。
b.一对一法（one-versus-one,简称1-v-1 SVMs）。其做法是在任意两类样本之间设计一个SVM，因此k个类别的样本就需要设计k(k-1)/2个SVM。当对一个未知样本进行分类时，最后得票最多的类别即为该未知样本的类别。Libsvm中的多类分类就是根据这个方法实现的。

```
clf = svm.SVC(decision_function_shape='ovo')
```

决策树（重点模型）：分类和回归都行。分类就叫分类树，回归就叫回归树

决策树是一种基本的分类和回归方法，用于分类主要是借助每一个叶子节点对应一种属性判定，通过不断的判定导出最终的决策；用于回归则是用均值函数进行多次二分，用子树中数据的均值进行回归。

学生编号	分数	出勤率	回答问题次数	作业提交率	分类:是否好学生
1	99	80%	5	90%	是
2	89	100%	6	100%	是
3	69	100%	7	100%	否
4	50	60%	8	70%	否
5	95	70%	9	80%	否
6	98	60%	10	80%	是
7	92	65%	11	100%	是
8	91	80%	12	85%	是
9	85	80%	13	95%	是
10	85	91%	14	98%	是



决策树的生成主要分以下两步，这两步通常通过学习已经知道分类结果的样本来实现。

1） 节点的分裂：一般当一个节点所代表的属性无法给出判断时，则选择将这一节点分成2个子节点（如不是二叉树的情况会分成n个子节点）

2）阈值的确定：选择适当的阈值使得分类错误率最小 （Training Error）
比较常用的决策树算法有ID3、C4.5和CART（分类和回归树）。CART的效果一般最好。

ID3：信息增益 ---分类树
C4.5：信息增益率---分类树
CART：基尼系数（从样本中随机抽取两个样本，其类别不一致的概率，越小越小）---分类回归树

信息增益：**代表了在一个条件下，信息复杂度（不确定性）减少的程度。**

熵：表示随机变量的不确定性。

条件熵：在一个条件下，随机变量的不确定性。

信息增益：熵 - 条件熵。表示在一个条件下，信息不确定性减少的程度。

通俗地讲，X(明天下雨)是一个随机变量，X的熵可以算出来， Y(明天阴天)也是随机变量，在阴天情况下下雨的信息熵我们如果也知道的话（此处需要知道其联合概率分布或是通过数据估计）即是条件熵。

X的熵减去Y条件下X的熵，就是信息增益。具体解释：原本明天下雨的信息熵是2，条件熵是0.01（因为如果知道明天是阴天，那么下雨的概率很大，信息量少），这样相减后为1.99。在获得阴天这个信息后，下雨信息不确定性减少了1.99，不确定减少了很多，所以信息增益大。也就是说，阴天这个信息对明天下午这一推断来说非常重要。

所以在特征选择的时候常常用信息增益，如果IG（信息增益大）的话那么这个特征对于分类来说很关键，决策树就是这样来找特征的。

ID3：信息增益 ---分类树

ID3分类树的问题：信息增益会偏向那些数据量多个类别。

因此才有了C4.5（信息增益率）：克服了ID3的不足。

C4.5:能够分析不衡平的数据集。

CART树：回归和分类树----基尼系数，可以进行分类也可以回归

决策树的收敛条件：
1）达到最大迭代深度
2）不纯度的减少小于阈值

Q：为什么不能无限的分下去？这样会出现过拟合吗？

其他问题：
决策树是容易发生过拟合的。如何解决？剪枝

前剪枝：在树的生成过程中，用过一些阈值来限制一些分支
后剪枝：通过极小化决策树的整体损失来进行剪枝

思考？决策树需不需要数据归一化？不需要的

一般的机器学习模型的特征都是需要进行数据归一化的。
NB，假设数据独立的、同分布的。
X1 [0,1]
X2 [0,10000]
Y = w1x1+w2x2 + b
L2正则化

Python

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

tree.DecisionTreeClassifier(*[, criterion, ...])	A decision tree classifier.
tree.DecisionTreeRegressor(*[, criterion, ...])	A decision tree regressor.
tree.ExtraTreeClassifier(*[, criterion, ...])	An extremely randomized tree classifier.
tree.ExtraTreeRegressor(*[, criterion, ...])	An extremely randomized tree regressor.

load_iris数据集的决策树分类。五折CV进行参数选择（不用自己划分验证集）

Criterion

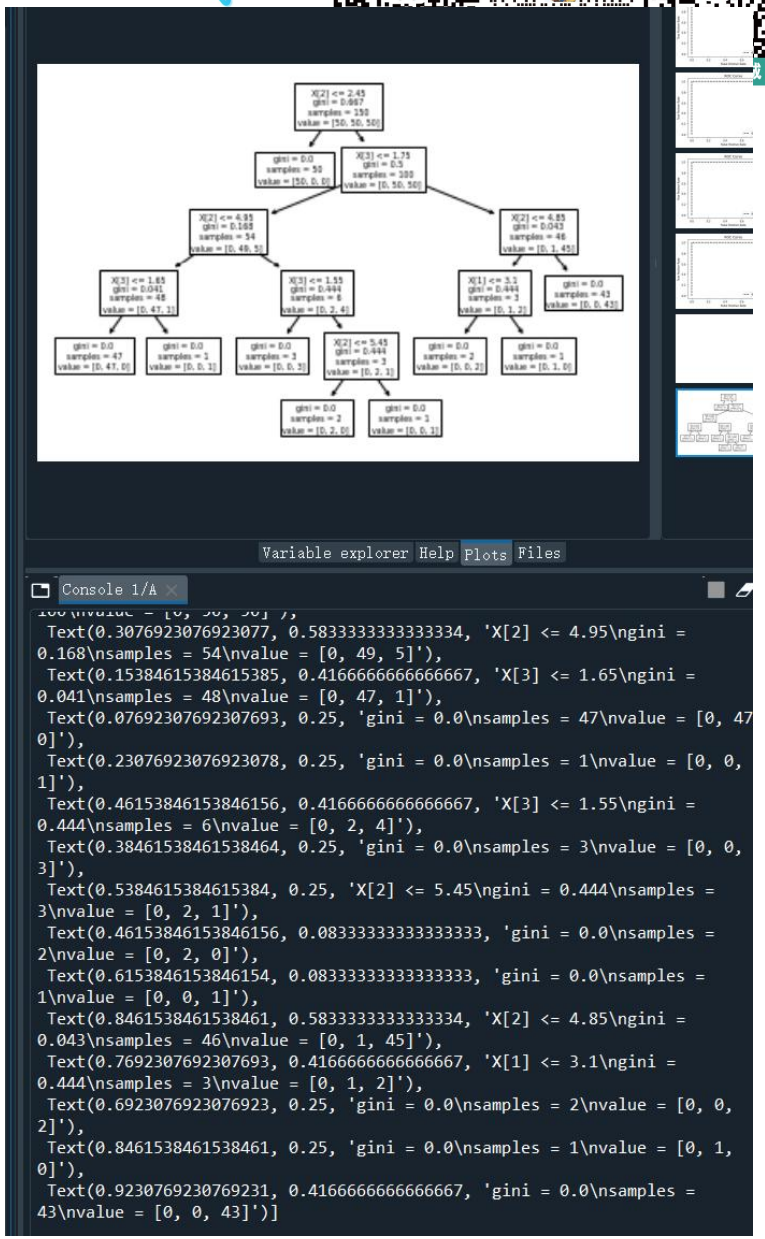
max_depth

Splitter （可以不进行参数选择）

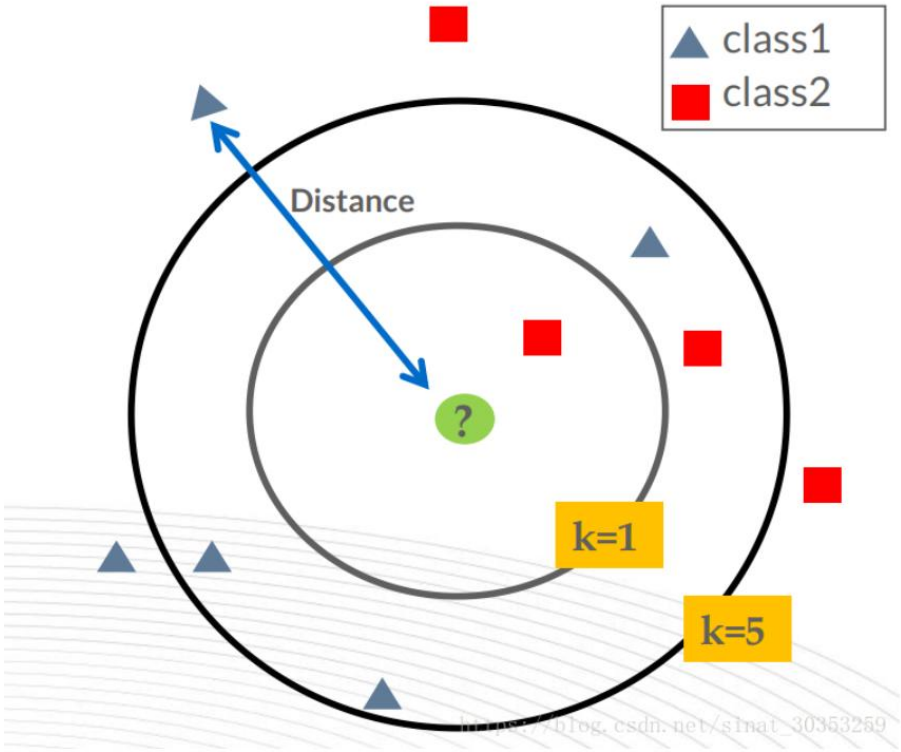
Test acc

parameters = {'**Criterion**':('***gini***', '***entropy***'), '**max_depth**':[5， 10]}

树的可视化：
https://scikit-learn.org/stable/modules/generated/sklearn.tree.plot_tree.html#sklearn.tree.plot_tree



KNN（K邻近）：k个最近的邻居，即每个样本都可以用它最接近的k个邻居来代表。



一个样本与数据集中的k个样本最相似， 如果这k个样本中的大多数属于某一个类别， 则该样本也属于这个类别。

如何度量两个样本是否相似？

在选择两个实例相似性时，一般使用的欧式距离

Lp距离定义：

$$L_p(x_i, x_j) = \left(\sum_{l=1}^{n} |x_i^{(l)} - x_j^{(l)}|^p \right)^{\frac{1}{p}}$$

K值的确定：

K值过小：k值小，特征空间被划分为更多子空间（模型的项越多），整体模型变复杂，容易发生过拟合，k值越小，选择的范围就比较小，训练的时候命中率较高，近似误差小，而用test的时候就容易出错，估计误差大，容易过拟合。

K值=N：无论输入实例是什么，都将简单的预测他属于训练实例中最多的类。

态度、有深度、有温度

$$d = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

《唐人街探案》VS《伦敦陷落》

$$d = \sqrt{(23-2)^2 + (3-3)^2 + (17-55)^2} = 43.42$$

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型
1	功夫熊猫	39	0	31	喜剧片
2	叶问3	3	2	65	动作片
3	伦敦陷落	2	3	55	动作片
4	代理情人	9	38	2	爱情片
5	新步步惊心	8	34	17	爱情片
6	谍影重重	5	2	57	动作片
7	功夫熊猫	39	0	31	喜剧片
8	美人鱼	21	17	5	喜剧片
9	宝贝当家	45	2	9	喜剧片
10	唐人街探案	23	3	17	?

序号	电影名称	搞笑镜头	拥抱镜头	打斗镜头	电影类型	距离	K=5时
1	功夫熊猫	39	0	31	喜剧片	21.47	√
2	叶问3	3	2	65	动作片	52.01	
3	伦敦陷落	2	3	55	动作片	43.42	
4	代理情人	9	38	2	爱情片	40.57	
5	新步步惊心	8	34	17	爱情片	34.44	√
6	谍影重重	5	2	57	动作片	43.87	
7	功夫熊猫	39	0	31	喜剧片	21.47	√
8	美人鱼	21	17	5	喜剧片	18.55	√
9	宝贝当家	45	2	9	喜剧片	23.43	√
10	唐人街探案	23	3	17	?	——	?

KNN（k邻近算法）

参考：

https://www.cnblogs.com/listenfwind/p/10685192.html

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

sklearn.neighbors: Nearest Neighbors

The sklearn.neighbors module implements the k-nearest neighbors algorithm.

User guide: See the Nearest Neighbors section for further details.

neighbors.BallTree(X[, leaf_size, metric])	BallTree for fast generalized N-point problems
neighbors.KDTree(X[, leaf_size, metric])	KDTree for fast generalized N-point problems
neighbors.KernelDensity(*[, bandwidth, ...])	Kernel Density Estimation.
neighbors.KNeighborsClassifier([...])	Classifier implementing the k-nearest neighbors vote.
neighbors.KNeighborsRegressor([n_neighbors, ...])	Regression based on k-nearest neighbors.
neighbors.KNeighborsTransformer(*[, mode, ...])	Transform X into a (weighted) graph of k nearest neighbors.
neighbors.LocalOutlierFactor([n_neighbors, ...])	Unsupervised Outlier Detection using the Local Outlier Factor (LOF).
neighbors.RadiusNeighborsClassifier([...])	Classifier implementing a vote among neighbors within a given radius.
neighbors.RadiusNeighborsRegressor([radius, ...])	Regression based on neighbors within a fixed radius.
neighbors.RadiusNeighborsTransformer(*[, ...])	Transform X into a (weighted) graph of neighbors nearer than a radius.
neighbors.NearestCentroid([metric, ...])	Nearest centroid classifier.
neighbors.NearestNeighbors(*[, n_neighbors, ...])	Unsupervised learner for implementing neighbor searches.
neighbors.NeighborhoodComponentsAnalysis([...])	Neighborhood Components Analysis.

neighbors.kneighbors_graph(X, n_neighbors, *)	Computes the (weighted) graph of k-Neighbors for points in X
neighbors.radius_neighbors_graph(X, radius, *)	Computes the (weighted) graph of Neighbors for points in X

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

请自行代码书写

模型保存和加载

from sklearn.externals import joblib
# 保存模型 joblib.dump(lr, "./ML/test.pkl") # lr是训练好的模型， "./ML/test.pkl"是模型要保存的路径及保存模型的文件名，其中，'pkl' 是sklearn中默认的保存格式gai

lr = joblib.load("./ML/test.pkl") # 进行模型的预测 y_pred = lr.predict(x_test) # 加载出来的模型跟我们训练出来的模型一样，有相同的参数

https://blog.csdn.net/weixin_45252110/article/details/98883571

- pip install joblib

总结：