# XGBoost

官网：http://xgboost.readthedocs.io/en/latest/
Github 地址：https://github.com/dmlc/xgboost

# 1.XGBoost 安装

http://xgboost.readthedocs.io/en/latest/build.html　XBOOST 安装文档

## 1.1 采用 whl 包安装方式

Windows 平台支持 python2 和 3（推荐 3.6）版本的 xgboost 版本，这里选用 python3.6 版本的 whl 文件安装。

（1）首先下载 xgboost 编译好的 whl 包

下载路径为：http://www.lfd.uci.edu/~gohlke/pythonlibs/#xgboost

这里采用：xgboost-0.6-cp36-cp36m-win_amd64.whl　这个是 python3.6，windows 64 位的包

（2）安装 xgboost

下载好后进入 whl 包的目录下执行，可以将下面的文件名替换为你自己下载的文件名，安装即可。在 anaconda 的 prompt 下安装。

命令：pip install xgboost-0.6-cp36-cp36m-win_amd64.whl

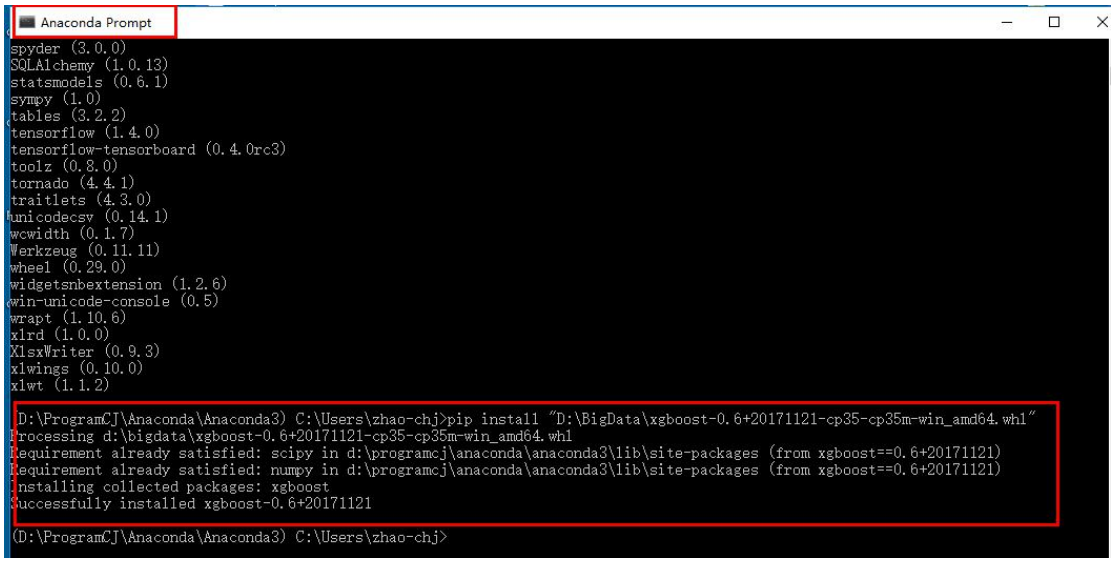
```
(D:\ProgramCJ\Anaconda\Anaconda3) C:\Users\zhao-chj>pip install xgboost-0.6-cp36-cp36m-win_amd64.whl
Processing c:\users\zhao-chj\xgboost-0.6-cp36-cp36m-win_amd64.whl
Requirement already satisfied: numpy in d:\programcj\anaconda\anaconda3\lib\site-packages (from xgboost==0.6)
Requirement already satisfied: scipy in d:\programcj\anaconda\anaconda3\lib\site-packages (from xgboost==0.6)
Installing collected packages: xgboost
Successfully installed xgboost-0.6
```

测试 XGBoost 的安装：


注意：如果环境变量配置不是 anaconda 的 python35 或 python36，两种解决方法

（1）修改环境变量为 anaconda 的 python35

（2）打开 anaconda 的 prompt，如下图输入命令安装




# 1.2 Graphviz 的安装

http://pypi.python.org/pypi/graphviz，可视化工具包，可以借助该工具显示树结构。安装运行：pip install graphviz

# 1.3 XGBOOST 的 API

DMatrix 是 XGBoost 创建的基本的数据结构

***class*** `xgboost.` `DMatrix` (*data*, *label=None*, *missing=None*, *weight=None*, *silent=False*, *feature_names=None*, *feature_types=None*)

Bases: `object`

Data Matrix used in XGBoost.

DMatrix is a internal data structure that used by XGBoost which is optimized for both memory efficiency and training speed. You can construct DMatrix from numpy.arrays

`feature_names`

Get feature names (column labels).

**Returns:** **feature_names**
**Return type:** list or None

`feature_types`

Get feature types (column types).

**Returns:** **feature_types**
**Return type:** list or None

`xgboost.train` (*params, dtrain, num_boost_round=10, evals=(), obj=None, feval=None, maximize=False, early_stopping_rounds=None, evals_result=None, verbose_eval=True, learning_rates=None, xgb_model=None, callbacks=None*)

Train a booster with given parameters.

**Parameters:**

- **params** (*dict*) – Booster params.
- **dtrain** (*DMatrix*) – Data to be trained.
- **num_boost_round** (*int*) – Number of boosting iterations.
- **evals** (*list of pairs (DMatrix, string)*) – List of items to be evaluated during training, this allows user to watch performance on the validation set.
- **obj** (*function*) – Customized objective function.
- **feval** (*function*) – Customized evaluation function.
- **maximize** (*bool*) – Whether to maximize feval.
- **early_stopping_rounds** (*int*) – Activates early stopping. Validation error needs to decrease at least every <early_stopping_rounds> round(s) to continue training. Requires at least one item in evals. If there's more than one, will use the last. Returns the model from the last iteration (not the best one). If early stopping occurs, the model will have three additional fields: bst.best_score, bst.best_iteration and bst.best_ntree_limit. (Use bst.best_ntree_limit to get the correct value if num_parallel_tree and/or num_class appears in the parameters)
- **evals_result** (*dict*) –
  This dictionary stores the evaluation results of all the items in watchlist. Example: with a watchlist containing [(dtest,'eval'), (dtrain,'train')] and a paramater containing ('eval_metric', 'logloss') Returns: {'train': {'logloss': ['0.48253', '0.35953']},

  'eval': {'logloss': ['0.480385', '0.357756']}}

- **verbose_eval** (*bool or int*) – Requires at least one item in evals. If *verbose_eval* is True then the evaluation metric on the validation set is printed at each boosting stage. If *verbose_eval* is an integer then the evaluation metric on the validation set is printed at every given *verbose_eval* boosting stage. The last boosting stage / the boosting stage found by using *early_stopping_rounds* is also printed. Example: with verbose_eval=4 and at least one item in evals, an evaluation metric is printed every 4 boosting stages, instead of every boosting stage.
- **learning_rates** (*list or function*) – List of learning rate for each boosting round or a customized function that calculates eta in terms of current number of round and the total number of boosting round (e.g. yields learning rate decay) - list l: eta = l[boosting round] - function f: eta = f(boosting round, num_boost_round)
- **xgb_model** (*file name of stored xgb model or 'Booster' instance*) – Xgb model to be loaded before training (allows training continuation).
- **callbacks** (*list of callback functions*) – List of callback functions that are applied at end of each iteration.

**Returns:** **booster**

**Return** a trained booster model

predict (*data*, *output_margin=False*, *ntree_limit=0*, *pred_leaf=False*)

Predict with data.

**NOTE: This function is not thread safe.**

For each booster object, predict can only be called from one thread. If you want to run prediction using multiple thread, call bst.copy() to make copies of model object and then call predict

**Parameters:**
- **data** (*DMatrix*) – The dmatrix storing the input.
- **output_margin** (*bool*) – Whether to output the raw untransformed margin value.
- **ntree_limit** (*int*) – Limit number of trees in the prediction; defaults to 0 (use all trees).
- **pred_leaf** (*bool*) – When this option is on, the output will be a matrix of (nsample, ntrees) with each record indicating the predicted leaf index of each sample in each tree. Note that the leaf index of a tree is unique per tree, so you may find leaf 1 in both tree 1 and tree 0.

**Returns:** **prediction**

**Return** numpy array

# 2.XGBoost 入门

## 2.1XGBoost 介绍

XGBoost 是"极端梯度提升"(eXtreme Gradient Boosting)的简称。XGBoost 源于梯度提升框架，但是能并行计算、近似建树、对稀疏数据的有效处理以及内存使用优化，这使得 XGBoost 至少比现有梯度提升实现有至少 10 倍的速度提升。XGBoost 可以处理回归、分类和排序等多种任务。由于它在预测性能上的强大且训练速度快，XGBoost 已屡屡斩获 Kaggle 各大竞赛的冠军宝座。

XGBoost：在机器学习的应用中一个强有力的工具

Gradient Boosting Maching(GBM)的优化实现，快速有效

Scikitlearn 有 XGboost 的实现，有 18 个参数需要调

XGBOOST 更多的是一种工程上优化，内存使用优化，并行计算等。

因此，XGBoost 参数调优需要了解模型基本原理。建议“给了一个问题如果找不到好的模型的话，可以试一下 XGBoost 模型，不会让你太失望。

我们首先回顾之前学习的两个知识点：

### （1）一般机器学习的处理步骤回归：

机器学习模型就是建立数据产生规律，每个模型我们都要建立真实值和预测值之间的联系，因此我们需要构建目标函数用于评估模型的准确程度，包括了损失函数和正则项两个部分。损失函数一般是与任务有关的，我们经常选择与训练数据匹配最好的模型，如回归模型通常使用残差平方的损失函数，分类问题我们选择 0-1 损失、logistic 损失及合叶损失（SVM）。

正则项一般是与模型复杂度有关的，这里我们遵循奥卡姆剃刀原则，通常有 L2 正则和 L1 正则，使用的时候常用的是 L2 正则。这里的 XGBoost 也使用了 L2

正则。

**Occam 奥卡姆剃刀原则：**模型尽量要简单有效

## （2）回顾一下 Boosting Machines

Gradient 梯度（用梯度的方法）

Boosting 增强（也就是组合一些弱分类器）

根据前面的 Boosting 算法：我们一般是将弱学习器组合为强学习器。而弱学习器定义为比随机猜测性能好的学习器。常用的弱学习器有决策树或分类回归树（CART 树）。

决策树：每个叶子结点对应决策中的一种。

分类回归树:每个叶子结点有个预测分数，比决策树更更加灵活。树的结构为二叉树。

AdaBoost 是 Boosting 算法的核心代表，通过串行组合各个弱分类器，对当前分类器不能处理的样本或处理效果比较差的样本增加其权重。不断的加入新的弱学习器，直到达到终止条件。一般把 AdaBoost 称为强学习器，强学习器是弱学习器的加权线性组合，权重与正确性相关。

AdaBoost 在 1995 年提出，2001 年人脸检测中被广泛使用

Firedman 将 AdaBoost 推广到一般的 Gradient Boosting 框架，得到了 GBM（Gradient Boosting Machine）：将 boosting 视作一个数值优化问题，采用类似梯度下降的方式优化求解。这种优化方法使得我们可以使用任何可微的损失函数（因为有了梯度），从而支持的任务从两类分类扩展到了多分类分类问题。

注意：

弱学习器一般都是由树模型组合。原则上弱学习器用哪个都行，因为树模型是非线性的，在很多方面的应用也表现的很好。因为 XGBOOST 只是一个框架。

# 2.2 XGBoost 的特别之处：

特点：

（1）正则化：以“正则化提升（reularized boosting）著称”

-标准 GBM 的实现没有显示的正则化步骤

-正则化对减少过拟合有帮助

（2）并行处理，相比 GBM 有了速度的飞跃

-借助 OpenMP，自动利用单机 CPU 的多核进行并行计算。

-支持 GPU 加速

-支持分布式

（3）高度的灵活性：允许用户自定义优化目标和评价标准

-只需损失函数的一阶导数和二阶导数。

（4）提供多语言接口

-命令行 CLI，C++，python,R,Julia,Java 和 JVM 语言（如 Scala）

XGBoost 称为近几年应用机器学习领域内的一个强有力的武器：

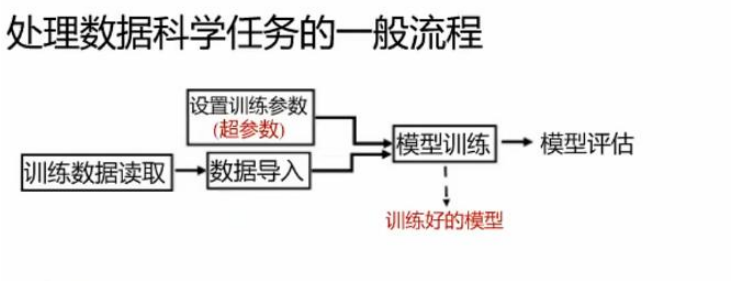
（1）执行速度上确实比其他的 Gradient Boosting 实现快

（2）模型性能：在结构数据集上，在分类/回归/排序预测建模上表现突出。

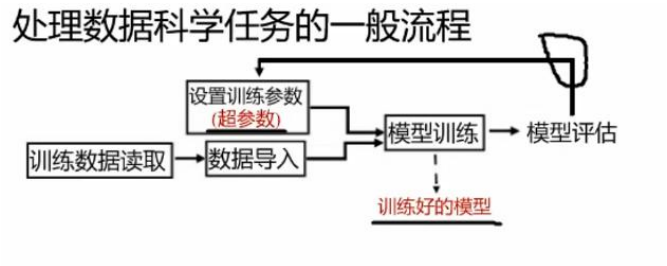
（3）执行速度：在 Kaggle 的 Higgs Boson 竞赛中，单线程的 XGBoost 比 GBM 的 R 语言包实现和 Python 的 sklearn 实现快将近一倍。并且多线程有接近线性程度的提升。

# 2.3 处理数据科学任务流程

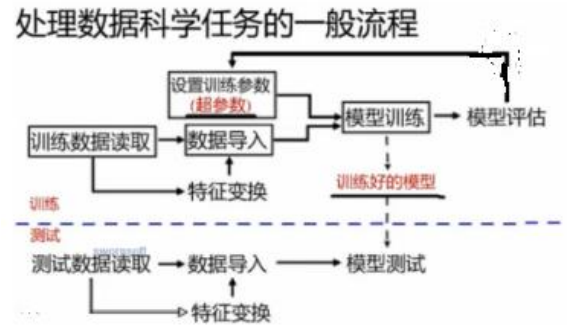
（1）根据训练数据进行模型训练




（2）根据模型评估情况调整超参数。

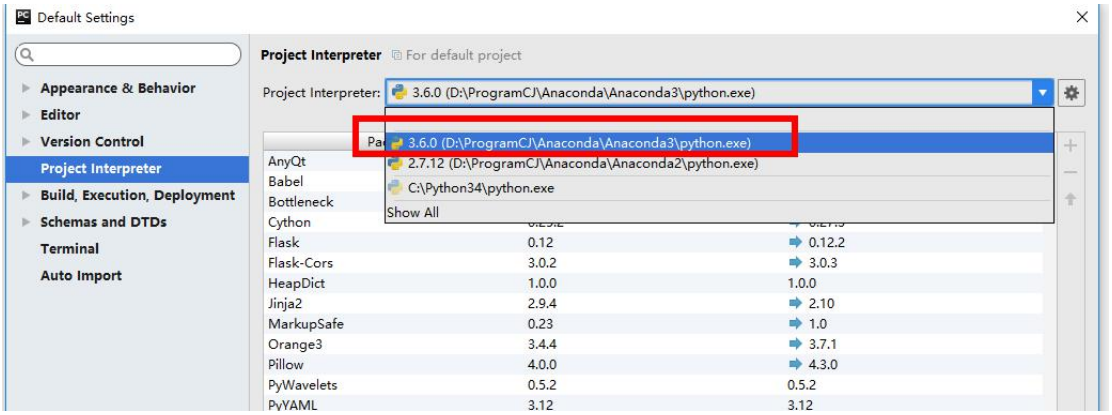



（3）测试数据运用到训练好好的模型中




注意：

**如何更改 pycharm 的 Anaconda3 的环境问题？**




# 2.4 XGBoost 的第一个小任务

（1）确定任务：

数据集：UCI 机器学习库的 Mushroom 数据集，是 XGBoost 中的 Demo 数据。

任务：根据蘑菇的 22 特征判断蘑菇是否有毒。

总样本数：8124，可食用的 4208,51.8% 有毒的 3916,48.2%

训练样本：6513，测试样本：1611

（2）特征处理

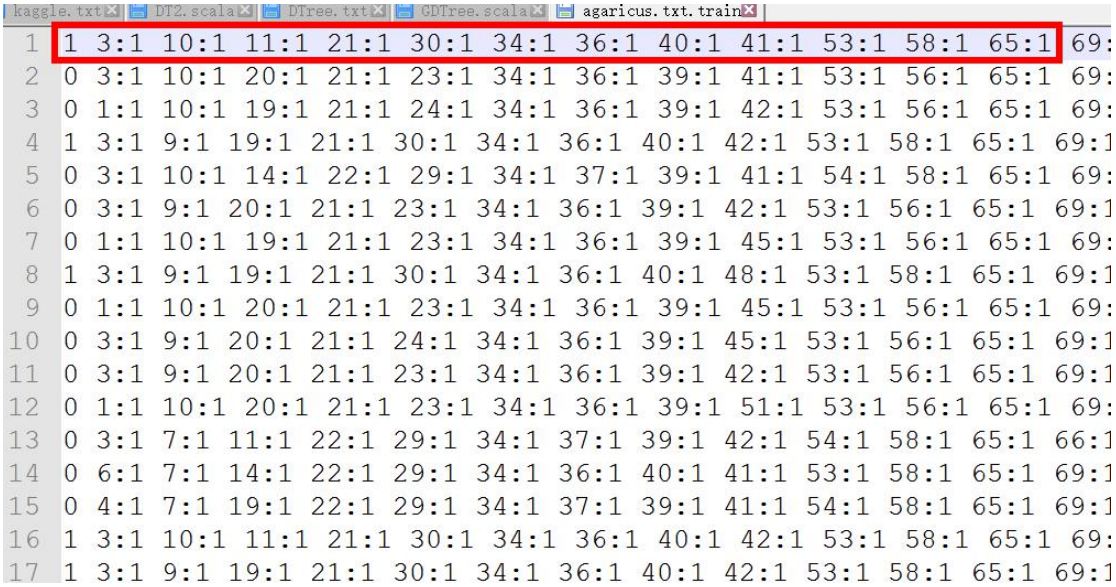
数据集中的 22 维度特征经过处理，变成了 126 维特征量。（One-hot 编码为 126 维度特征），参考以下链接：

http://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.names

```
7. Attribute Information: (classes: edible=e, poisonous=p)
     1. cap-shape:                bell=b, conical=c, convex=x, flat=f,
                                  knobbed=k, sunken=s
     2. cap-surface:              fibrous=f, grooves=g, scaly=y, smooth=s
     3. cap-color:                brown=n, buff=b, cinnamon=c, gray=g, green=r,
                                  pink=p, purple=u, red=e, white=w, yellow=y
     4. bruises?:                 bruises=t, no=f
     5. odor:                     almond=a, anise=l, creosote=c, fishy=y, foul=f,
                                  musty=m, none=n, pungent=p, spicy=s
     6. gill-attachment:          attached=a, descending=d, free=f, notched=n
     7. gill-spacing:             close=c, crowded=w, distant=d
     8. gill-size:                broad=b, narrow=n
     9. gill-color:               black=k, brown=n, buff=b, chocolate=h, gray=g,
                                  green=r, orange=o, pink=p, purple=u, red=e,
                                  white=w, yellow=y
    10. stalk-shape:              enlarging=e, tapering=t
    11. stalk-root:               bulbous=b, club=c, cup=u, equal=e,
                                  rhizomorphs=z, rooted=r, missing=?
    12. stalk-surface-above-ring: fibrous=f, scaly=y, silky=k, smooth=s
    13. stalk-surface-below-ring: fibrous=f, scaly=y, silky=k, smooth=s
    14. stalk-color-above-ring:   brown=n, buff=b, cinnamon=c, gray=g, orange=o,
                                  pink=p, red=e, white=w, yellow=y
    15. stalk-color-below-ring:   brown=n, buff=b, cinnamon=c, gray=g, orange=o,
                                  pink=p, red=e, white=w, yellow=y
    16. veil-type:                partial=p, universal=u
    17. veil-color:               brown=n, orange=o, white=w, yellow=y
    18. ring-number:              none=n, one=o, two=t
    19. ring-type:                cobwebby=c, evanescent=e, flaring=f, large=l,
                                  none=n, pendant=p, sheathing=s, zone=z
    20. spore-print-color:        black=k, brown=n, buff=b, chocolate=h, green=r,
                                  orange=o, purple=u, white=w, yellow=y
    21. population:               abundant=a, clustered=c, numerous=n,
                                  scattered=s, several=v, solitary=y
    22. habitat:                  grasses=g, leaves=l, meadows=m, paths=p,
                                  urban=u, waste=w, woods=d

8. Missing Attribute Values: 2480 of them (denoted by "?"), all for
   attribute #11.

9. Class Distribution:
    --    edible: 4208 (51.8%)
    -- poisonous: 3916 (48.2%)
    --     total: 8124 instances
```

（3）XGBoost有多少参数？这些参数怎么设定？怎样使用Scikit-learn框架使用熟练，怎么样快速集成新的XGBoost工具？

## 2.4.1 数据集解释




Libsvm 使用的数据格式是一个存储稀疏特征的数据格式。

该数据集为 libsvm 的文本数据格式，libsvm 的文件格式主要用于稀疏特征的表示中，每一行为一个样本，如 1 101:1.2 的 1 表示样本类标签，101 表示特征索引，1.2 表示特征的值，前 100 维度特征均为 0，因为组成的矩阵是稀疏矩阵。

通常 XGBoost 加载数据存储在对相关 Dmatrix 中。

备注：

（1）LIBSVM 是台湾大学林智仁(Lin Chih-Jen)教授等开发设计的一个简单、易于使用和快速有效的 SVM 模式识别与回归的软件包，他不但提供了编译好的可在 Windows 系列系统的执行文件，还提供了源代码，方便改进、修改以及在其它操作系统上应用；该软件对 SVM 所涉及的参数调节相对比较少，提供了很多的默认参数，利用这些默认参数可以解决很多问题；并提供了交互检验(Cross Validation)的功能。该软件可以解决 C-SVM、v-SVM、ε-SVR 和v-SVR 等问题，包括基于一对一算法的多类模式识别问题。

（2）使用的训练数据和检验数据文件格式如下：

<label> <index1>:<value1> <index2>:<value2> ...

其中<label> 是训练数据集的目标值，对于分类，它是标识某类的整数(支持多个类)；对于回归，是任意实数。<index> 是以 1 开始的整数，可以是不连续的；<value>为实数，也就是我们常说的自变量。检验数据文件中的 label 只用于计算准确度或误差，如果它是未知的，只需用一个数填写这一栏，也可以空着不填。在程序包中，还包括有一个训练数据实例。

**代码片段 1：**

```
import xgboost as xgb
```

```
from sklearn.metrics import accuracy_score
```

```
#读入数据
```

```
dtrain=xgb.DMatrix("D:\\BigData\\xgboost_Model\\First_Lesson\\Lec1Code\\data\\agaricus.txt.train")
```

```
dtest=xgb.DMatrix("D:\\BigData\\xgboost_Model\\First_Lesson\\Lec1Code\\data\\agaricus.txt.test")
```

注 1：Fit 和 fittransform 区别：fit 只做训练不做编码，fittransform 既做训练又做编码。

注 2：XGBoost 的数据接口：

（1）XGBoost 加载的数据存储在对象 Dmatrix 中，做了存储效率和运行速度的优化。

（2）支持三种接口：1.libsvm txt 格式的文件，2.常见的矩阵（numpy2D array），3.xgboost inary buffer file 二进制文件

代码片段 2：查看数据的情况

注意 python3 中使用 print(dtrain.num_rows)来打印

```
In [19]: dtrain.num_col()#查看训练集维度
Out[19]: 127

In [20]: dtrain.num_row()#查看训练集行
Out[20]: 6513

In [21]: dtest.num_col()#查看测试集维度
Out[21]: 127

In [22]: dtest.num_row()#测试集的行数
Out[22]: 1611
```

## 2.4.2 设置训练参数

代码片段 2:

```
In [8]: param={'max_depth':2,'eta':1,'silent':0,'objective':'binary:logistic'}
```

（1）max_depth:树的最大深度，缺省值为 6，取值范围是[1,+无穷]

（2）Eta:为了防止过拟合，更新过程中用到的收缩步长。Eta 通过缩减特征的权重使 t 提升计算过程更加保守。缺省值为 0.3，取值范围为：[0,1]

（3）Silent：0 表示打印出运行时信息，取 1 时表示以缄默方式运行，不打印运行时信息。缺省值为 0。

（4）Objective：定义学习任务及相应的学习目标，“binary：logistic”表示

二分类的逻辑回归问题，输出为概率。

设置 Boosting 迭代次数,最后会用到多少棵树：num_round=2

代码片段 3：

```
In [9]: num_round=2

In [10]: bst=xgb.train(param, dtrain, num_round)
```

代码片段 4：

```
In [ ]: ##打印训练模型所用的时间

In [23]: import time
         starttime=time.clock()
         bst=xgb.train(param, dtrain, num_round)
         endtime=time.clock()
         print(endtime-starttime)

         0.027760482487621963
```

## 2.4.3 预测（训练数据上的评估）

模型训练好后，可以用训练好的模型对测试数据尽心预测。

--XGBoost 预测的输出是概率，输出值是样本为第一类的概率，在将概率转化为 0 或 1。

代码片段 4：

```
In [13]: #预测（训练数据上的预测）

In [14]: train_preds=bst.predict(dtrain)
         train_predictions=[round(value) for value in train_preds]
         y_train=dtrain.get_label()
         train_accuracy=accuracy_score(y_train, train_predictions)
         print("Train Accuary:%.2f%%"%(train_accuracy*100.0))

         Train Accuary:97.77%
```

代码片段 5：

对测试数据的预测：可以达到 97.83%的准确率。

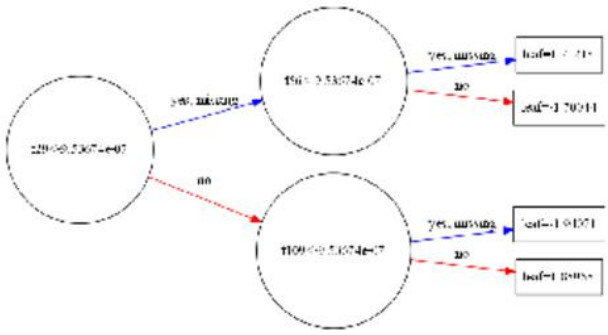
```
In [15]: #对测试数据的预测

In [17]: preds=bst.predict(dtest)
         predictions=[round(value) for value in preds]
         y_test=dtest.get_label()
         test_accuracy=accuracy_score(y_test, predictions)
         print("Train Accuary:%.2f%%"%(test_accuracy*100.0))

         Train Accuary:97.83%
```

## 2.4.4 模型可视化

调用 XGBoost 工具包中的 plot_tree，在显示要可视化模型需要安装 graphviz 软件包，plot_tree()的三个参数：1.训练好的模型，2.要打印的树的索引，从 0 开始，3.打印显示方向，缺省值为竖直，‘LR’是水平方向。

代码片段 6：

```
In [29]: from matplotlib import pyplot
         import graphviz  #在promot中pip install graphviz
         from xgboost import plot_tree

In [35]: xgb.plot_tree(bst,num_trees=0,rankdir='LR')
         pyplot.show()
```



## 2.4.5 训练参数调整变化

参数 1：max_depth=2,num_round=2

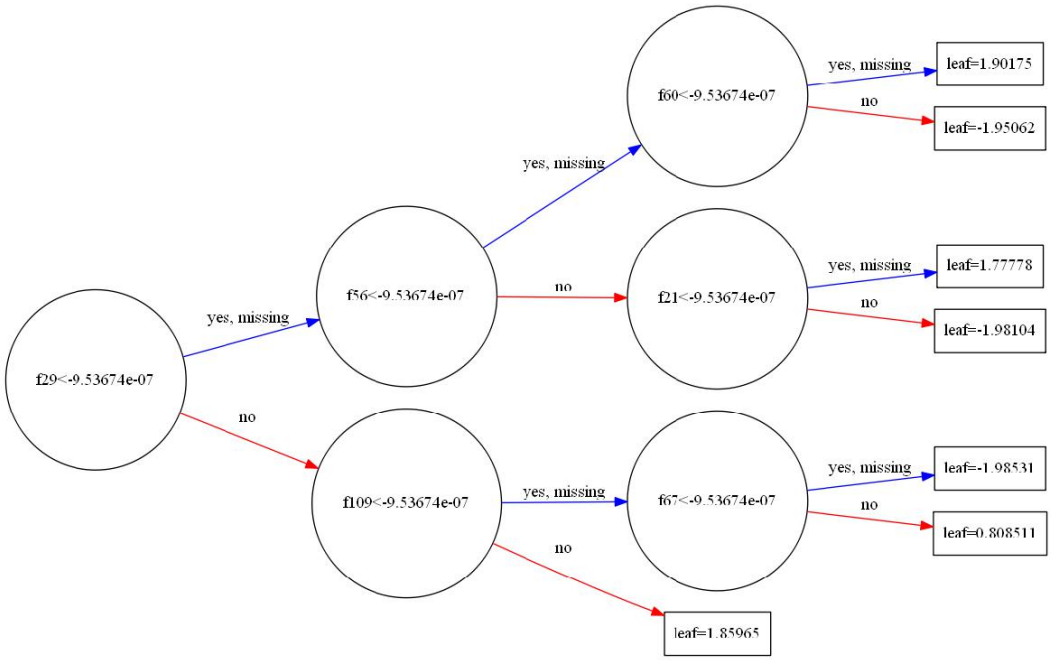
-训练集上正确率为：97.77%

-测试集上的正确率为：97.83%

参数 2：max_depth=3,num_round=2（**树的深度加深了，准确率提高了**）

-训练集上正确率为：99.88%

-测试集上的正确率为：100%


f29<-9.53674e-07
yes, missing
no
f56<-9.53674e-07
f109<-9.53674e-07
yes, missing
no
yes, missing
no
f60<-9.53674e-07
f21<-9.53674e-07
f67<-9.53674e-07
yes, missing
leaf=1.90175
no
leaf=-1.95062
yes, missing
leaf=1.77778
no
leaf=-1.98104
yes, missing
leaf=-1.98531
no
leaf=0.808511
leaf=1.85965

# 2.5XGBoost 与 Scikit-learn 结合

以蘑菇数据 mushroom 为例子

Sklearn 框架下的机器学习模型的训练和测试流程

（1）构造学习器（设置参数）fit---predict

Fit 和 train 类似，均为训练模型的方法

Fit 和 fit_transform 区别：fit 训练相应的参数，而 fit_transform 不仅可以训练得到相应的参数，而且会进行相应编码。

（2）XGBoost 提供了一个 wrapper 类，允许模型可以和 sklearn 框架中的其他分类器或回归器一样对待。

XGBoost 中的分类模型为 XGBClassifier，模型参数在构造时传递。

## 2.5.1 导入 sklearn 和 xgboost 的包

直接调用 XGBoost 的时候是直接可以用 libsvm 作为输入的，但是在 sklearn 需要加载 libsvm 模块才可使用。

代码片段 1：

In [1]:
```
#1.导入程序需要的包
# 运行 xgboost安装包中的示例程序
from xgboost import XGBClassifier

# 加载LibSVM格式数据模块
from sklearn.datasets import load_svmlight_file
from sklearn.metrics import accuracy_score

from matplotlib import pyplot
```

## 2.5.2 sklearn 读取 libsvm 数据

代码片段 2：

```
In [3]: #2.scikit-learn中读取libsvm数据的方法，分为属性和类标签
        X_train,y_train=load_svmlight_file("D:\\BigData\\xgboost_Mode
        X_test,y_test=load_svmlight_file("D:\\BigData\\xgboost_Model\

In [5]: #3.打印测试集或训练集的维度和行数
        print(X_train.shape)
        print (X_test.shape)

        (6513, 126)
        (1611, 126)
```

## 2.5.3 训练参数设置

和上面的 XGBoost 参数大体一致。

max_depth： 树的最大深度。缺省值为 6，取值范围为：[1,∞]

eta：为了防止过拟合，更新过程中用到的收缩步长。在每次提升计算之后，算法会直接获得新特征的权重。eta 通过缩减特征的权重使提升计算过程更加保守。缺省值为 0.3，取值范围为：[0,1]

silent：取 0 时表示打印出运行时信息，取 1 时表示以缄默方式运行，不打印运行时信息。缺省值为 0

objective： 定义学习任务及相应的学习目标，“binary:logistic” 表示二分类的逻辑回归问题，输出为概率。

其他参数取默认值。

代码片段 3：

```
In [8]: # 设置boosting迭代计算次数
        num_round = 2

        #bst = XGBClassifier(**params)
        #bst = XGBClassifier()
        bst =XGBClassifier(max_depth=2, learning_rate=1, n_estimators=num_round,
                           silent=True, objective='binary:logistic')

        bst.fit(X_train, y_train)

Out[8]: XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
               gamma=0, learning_rate=1, max_delta_step=0, max_depth=2,
               min_child_weight=1, missing=None, n_estimators=2, nthread=-1,
               objective='binary:logistic', reg_alpha=0, reg_lambda=1,
               scale_pos_weight=1, seed=0, silent=True, subsample=1)
```

## 2.5.4 在训练集上的测试（预测）

代码片段 4：

```
In [ ]: ##查看模型在训练集上的分类性能
        ##XGBoost预测的输出是概率。这里蘑菇分类是一个二类分类问题，
        #输出值是样本为第一类的概率。 我们需要将概率值转换为0或1。
```

```
In [9]: #4.在训练集上的测试
        train_preds = bst.predict(X_train)
        train_predictions = [round(value) for value in train_preds]

        train_accuracy = accuracy_score(y_train, train_predictions)
        print ("Train Accuary: %.2f%%" % (train_accuracy * 100.0))
```

```
Train Accuary: 97.77%
```

### 2.5.5 在测试集上的测试

代码片段 5：

```
In [10]: #5.模型训练好后，可以用训练好的模型对测试数据进行预测
         # make prediction
         preds = bst.predict(X_test)
         predictions = [round(value) for value in preds]

         test_accuracy = accuracy_score(y_test, predictions)
         print("Test Accuracy: %.2f%%" % (test_accuracy * 100.0))
```

```
Test Accuracy: 97.83%
```

```
In [ ]:
```

## 2.6 改进

我们上面的两个实验在训练集和测试集上都检查了模型的性能。但是实际场景中测试数据是未知的，如何评估模型呢？

这里我们采用校验集。

校验集：将训练数据的一部分流出来，不参与模型参数训练。留出来的这部分数据成为校验集。即：部分数据训练模型，训练好的模型在校验集上测试。而校验集上的性能可视为模型在未知数据上性能的估计，最后我们只需要选择在校验集上表现最好的模型。

校验集切分的代码如下：

In [6]:
```
#切分测试集和训练集。1/3的训练数据作为教研数据，seed随机种子保证结果可重复性
from sklearn.model_selection import train_test_split
from sklearn.datasets import load_svmlight_file
```

In [4]:
```
seed=7
test_size=7
```

In [7]:
```
X_train,y_train=load_svmlight_file("D:\\BigData\\xgboost_Model\\First_Lesson
                                   \\Lec1Code\\data\\agaricus.txt.train")
```

In [8]:
```
X_train_part,X_validation,y_train_part,y_validation=train_test_split(
        X_train,y_train,test_size=test_size,random_state=seed)
```

完整代码步骤如下：

## 2.6.1.导包，读入数据集

In [11]:
```
# 1.运行 xgboost安装包中的示例程序
from xgboost import XGBClassifier

# 加载LibSVM格式数据模块
from sklearn.datasets import load_svmlight_file

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from matplotlib import pyplot
```

```
D:\ProgramCJ\Anaconda\Anaconda3\lib\site-packages\sklearn\cross_validation.py:44: Dep
 0.18 in favor of the model_selection module into which all the refactored classes an
 the new CV iterators are different from that of this module. This module will be rem
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [12]:
```
# 2.读取数据
X_train,y_train = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_Lesson
                                     \\Lec1Code\\data\\agaricus.txt.train")
X_test,y_test = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_Lesson
                                   \\Lec1Code\\data\\agaricus.txt.test")

X_train.shape
#X_test.shape
```

Out[12]: (6513, 126)

## 2.6.2.train_test_split 方法分离数据集

In [13]:
```
##3.训练集和校验集分离
```

In [ ]:
```
#假设我们取1/3的训练数据做为校验数据
```

In [23]:
```
# split data into train and test sets, 1/3的训练数据作为校验数据
seed = 7
test_size = 0.33
X_train_part, X_validate, y_train_part, y_validate =
    train_test_split(X_train, y_train, test_size=test_size,random_state=seed)
X_train_part.shape
#X_validate.shape
```

Out[23]: (4363, 126)

In [25]:
```
X_validate.shape
```

Out[25]: (2150, 126)

## 2.6.3 训练参数设置

In [26]:
```
#4.训练参数设置
```

In [27]:
```
# 设置boosting迭代计算次数
num_round = 2

#bst = XGBClassifier(param)
#bst = XGBClassifier()
bst =XGBClassifier(max_depth=2, learning_rate=1, n_estimators=num_round,
                   silent=True, objective='binary:logistic')

bst.fit(X_train_part, y_train_part)
```

Out[27]:
```
XGBClassifier(base_score=0.5, colsample_bylevel=1, colsample_bytree=1,
       gamma=0, learning_rate=1, max_delta_step=0, max_depth=2,
       min_child_weight=1, missing=None, n_estimators=2, nthread=-1,
       objective='binary:logistic', reg_alpha=0, reg_lambda=1,
       scale_pos_weight=1, seed=0, silent=True, subsample=1)
```

## 2.6.4.模型测试

分别对校验数据，测试数据和训练数据进行测试。

In [28]:
```
#5.模型校验集上的性能
```

In [29]:
```
validare_preds = bst.predict(X_validate)
validate_predictions = [round(value) for value in validare_preds]

train_accuracy = accuracy_score(y_validate, validate_predictions)
print ("Validation Accuary: %.2f%%" % (train_accuracy * 100.0))
```

```
Validation Accuary: 97.63%
```

In [30]:
```
#6.查看模型在训练集上的分类性能
train_preds = bst.predict(X_train)
train_predictions = [round(value) for value in train_preds]

train_accuracy = accuracy_score(y_train, train_predictions)
print ("Train Accuary: %.2f%%" % (train_accuracy * 100.0))
```

```
Train Accuary: 97.77%
```

In [31]:
```
#XGBoost预测的输出是概率。这里蘑菇分类是一个二类分类问题，
#输出值是样本为第一类的概率。我们需要将概率值转换为0或1。
```

In [32]:
```
# 对测试数据进行预测make prediction
preds = bst.predict(X_test)
predictions = [round(value) for value in preds]

test_accuracy = accuracy_score(y_test, predictions)
print("Test Accuracy: %.2f%%" % (test_accuracy * 100.0))
```

```
Test Accuracy: 97.83%
```

# 2.7 学习曲线

模型预测性能随着某个变化的学习参数（如训练样本数、迭代次数）变化的情况，可以用学习曲线表示，从曲线中可以找到最合适的参数点。包括性能变化比较用学习曲线去看。

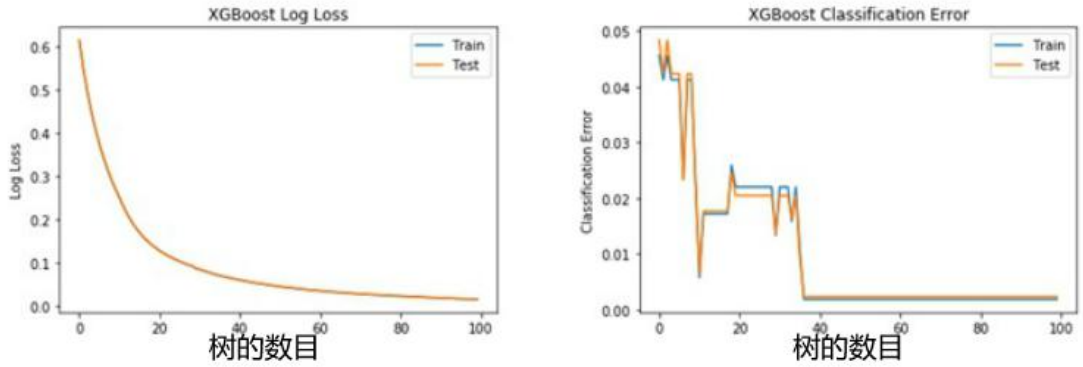
如：我们要选择的 XGBoost 的迭代次数（树的数目）

```
# 设置boosting迭代计算次数
num_round = 100
eval_set = [(X_train_part, y_train_part), (X_validation, y_validation)]
bst.fit(X_train_part, y_train_part, eval_metric=["error", "logloss"], eval_set=eval_set,
verbose=False)
```

迭代次数为 100，也就创建 100 颗树。

Eval_set 评估集，可以看一下在训练数据和校验数据上的性能。

还可以设置评价指标 eval_metrix，可设置 0-1 指标的 error 错误率，logloss 是逻辑斯损失，eval_set 是指定那些集合上评估。




由于 Train 和 Test 误差差异很小，图形的分辨率比较低，所以有的两者是重合的。

从图上可以看出，最佳的树的数目是 36 左右。

## 2.7.1 导包和读取数据

```
In [2]: # 1.运行 xgboost安装包中的示例程序
        import xgboost as xgb
        from xgboost import XGBClassifier

        # 加载LibSVM格式数据模块
        from sklearn.datasets import load_svmlight_file

        from sklearn.model_selection import train_test_split
        from sklearn.metrics import accuracy_score

        from matplotlib import pyplot
```

```
In [5]: #2.读取源数据
        #my_workpath = './data/'
        X_train,y_train = load_svmlight_file("D:\\BigData\\xgboost_M
        X_test,y_test = load_svmlight_file("D:\\BigData\\xgboost_Mod

        X_train.shape

Out[5]: (6513, 126)
```

## 2.7.2 训练集-校验集分离

```
In [6]: #3.训练集-校验集分离
        #假设我们取1/3的训练数据做为校验数据 ps: 为什么要校验?
```

```
In [7]: # split data into train and test sets, 1/3的训练数据作为校验数据
        seed = 7
        test_size = 0.33
        X_train_part, X_validate, y_train_part, y_validate=
            train_test_split(X_train, y_train, test_size=test_size,
            random_state=seed)

        X_train_part.shape
        X_validate.shape

Out[7]: (2150, 126)
```

## 2.7.3 训练模型

eval_set = [(X_train_part, y_train_part), (X_validate, y_validate)]，分别显示在训练数据和校验数据上的性能。

#validation_0-error:0.045611 中 validation_0-指的是训练集上的 error 是多少，还有逻辑斯特损失

#validation_1-error:0.048372 中 validation_1-指的是校验集上的 error 是多少，还有逻辑斯特损失

如下图所示：[0]、[1]表示树的个数为 1（[0]表示 1 棵树）、2 等等。

```
In [8]: #4训练模型--有了参数列表和数据就可以训练模型了

In [9]: # 设置boosting迭代计算次数
num_round = 100

#bst = XGBClassifier(param)
#bst = XGBClassifier()
bst =XGBClassifier(max_depth=2, learning_rate=0.1, n_estimators=num_round, silent=True, objective='binary:logistic')

eval_set = [(X_train_part, y_train_part), (X_validate, y_validate)]
bst.fit(X_train_part, y_train_part, eval_metric=["error", "logloss"], eval_set=eval_set, verbose=True)
```

```
[0]  validation_0-error:0.045611  validation_0-logloss:0.614637  validation_1-error:0.048372  validation_1-logloss:0.615401
[1]  validation_0-error:0.041256  validation_0-logloss:0.549907  validation_1-error:0.042326  validation_1-logloss:0.550696
[2]  validation_0-error:0.045611  validation_0-logloss:0.49543   validation_1-error:0.048372  validation_1-logloss:0.496777
[3]  validation_0-error:0.041256  validation_0-logloss:0.449089  validation_1-error:0.042326  validation_1-logloss:0.450412
[4]  validation_0-error:0.041256  validation_0-logloss:0.409231  validation_1-error:0.042326  validation_1-logloss:0.410717
[5]  validation_0-error:0.041256  validation_0-logloss:0.373748  validation_1-error:0.042326  validation_1-logloss:0.375653
[6]  validation_0-error:0.023378  validation_0-logloss:0.343051  validation_1-error:0.023256  validation_1-logloss:0.344738
[7]  validation_0-error:0.041256  validation_0-logloss:0.315369  validation_1-error:0.042326  validation_1-logloss:0.317409
```

我们可以看到从 36 行开始错误率恒定了，如下图。

```
[33] validation_0-error:0.015815  validation_0-logloss:0.074507  validation_1-error:0.016279  validation_1-logloss:0.074765
[34] validation_0-error:0.022003  validation_0-logloss:0.071848  validation_1-error:0.020465  validation_1-logloss:0.071811
[35] validation_0-error:0.010543  validation_0-logloss:0.069488  validation_1-error:0.009302  validation_1-logloss:0.069385
[36] validation_0-error:0.001834  validation_0-logloss:0.067147  validation_1-error:0.002326  validation_1-logloss:0.067341
[37] validation_0-error:0.001834  validation_0-logloss:0.06504   validation_1-error:0.002326  validation_1-logloss:0.065406
[38] validation_0-error:0.001834  validation_0-logloss:0.062898  validation_1-error:0.002326  validation_1-logloss:0.063381
[39] validation_0-error:0.001834  validation_0-logloss:0.060837  validation_1-error:0.002326  validation_1-logloss:0.061088
[40] validation_0-error:0.001834  validation_0-logloss:0.058894  validation_1-error:0.002326  validation_1-logloss:0.059039
[41] validation_0-error:0.001834  validation_0-logloss:0.057112  validation_1-error:0.002326  validation_1-logloss:0.057326
[42] validation_0-error:0.001834  validation_0-logloss:0.055391  validation_1-error:0.002326  validation_1-logloss:0.05543
[43] validation_0-error:0.001834  validation_0-logloss:0.053745  validation_1-error:0.002326  validation_1-logloss:0.053871
[44] validation_0-error:0.001834  validation_0-logloss:0.052198  validation_1-error:0.002326  validation_1-logloss:0.052235
[45] validation_0-error:0.001834  validation_0-logloss:0.050776  validation_1-error:0.002326  validation_1-logloss:0.051033
[46] validation_0-error:0.001834  validation_0-logloss:0.049351  validation_1-error:0.002326  validation_1-logloss:0.04973
[47] validation_0-error:0.001834  validation_0-logloss:0.047848  validation_1-error:0.002326  validation_1-logloss:0.048287
[48] validation_0-error:0.001834  validation_0-logloss:0.046406  validation_1-error:0.002326  validation_1-logloss:0.046702
[49] validation_0-error:0.001834  validation_0-logloss:0.045141  validation_1-error:0.002326  validation_1-logloss:0.045492
[50] validation_0-error:0.001834  validation_0-logloss:0.043917  validation_1-error:0.002326  validation_1-logloss:0.044133
[51] validation_0-error:0.001834  validation_0-logloss:0.042729  validation_1-error:0.002326  validation_1-logloss:0.042999
[52] validation_0-error:0.001834  validation_0-logloss:0.041608  validation_1-error:0.002326  validation_1-logloss:0.041807
[53] validation_0-error:0.001834  validation_0-logloss:0.040493  validation_1-error:0.002326  validation_1-logloss:0.040855
[54] validation_0-error:0.001834  validation_0-logloss:0.039457  validation_1-error:0.002326  validation_1-logloss:0.039871
[55] validation_0-error:0.001834  validation_0-logloss:0.038452  validation_1-error:0.002326  validation_1-logloss:0.038755
[56] validation_0-error:0.001834  validation_0-logloss:0.037478  validation_1-error:0.002326  validation_1-logloss:0.037717
[57] validation_0-error:0.001834  validation_0-logloss:0.036439  validation_1-error:0.002326  validation_1-logloss:0.036777
[58] validation_0-error:0.001834  validation_0-logloss:0.035552  validation_1-error:0.002326  validation_1-logloss:0.035936
```

但是最优值是在第 10 次出现：（后面通过 early-stop 参数停止迭代）

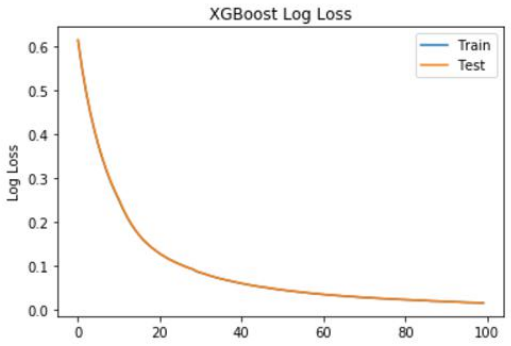
```
[0]  validation_0-error:0.045611  validation_0-logloss:0.614637  validation_1-error:0.048372  validation_1-logloss:0.615401
[1]  validation_0-error:0.041256  validation_0-logloss:0.549907  validation_1-error:0.042326  validation_1-logloss:0.550696
[2]  validation_0-error:0.045611  validation_0-logloss:0.49543   validation_1-error:0.048372  validation_1-logloss:0.496777
[3]  validation_0-error:0.041256  validation_0-logloss:0.449089  validation_1-error:0.042326  validation_1-logloss:0.450412
[4]  validation_0-error:0.041256  validation_0-logloss:0.409231  validation_1-error:0.042326  validation_1-logloss:0.410717
[5]  validation_0-error:0.041256  validation_0-logloss:0.373748  validation_1-error:0.042326  validation_1-logloss:0.375653
[6]  validation_0-error:0.023378  validation_0-logloss:0.343051  validation_1-error:0.023256  validation_1-logloss:0.344738
[7]  validation_0-error:0.041256  validation_0-logloss:0.315369  validation_1-error:0.042326  validation_1-logloss:0.317409
[8]  validation_0-error:0.041256  validation_0-logloss:0.290912  validation_1-error:0.042326  validation_1-logloss:0.292587
[9]  validation_0-error:0.023378  validation_0-logloss:0.269356  validation_1-error:0.023256  validation_1-logloss:0.271103
[10] validation_0-error:0.00573   validation_0-logloss:0.249593  validation_1-error:0.006512  validation_1-logloss:0.251354
[11] validation_0-error:0.01719   validation_0-logloss:0.228653  validation_1-error:0.017674  validation_1-logloss:0.230144
[12] validation_0-error:0.01719   validation_0-logloss:0.210442  validation_1-error:0.017674  validation_1-logloss:0.21167
[13] validation_0-error:0.01719   validation_0-logloss:0.194562  validation_1-error:0.017674  validation_1-logloss:0.19555
[14] validation_0-error:0.01719   validation_0-logloss:0.1807    validation_1-error:0.017674  validation_1-logloss:0.181463
[15] validation_0-error:0.01719   validation_0-logloss:0.168585  validation_1-error:0.017674  validation_1-logloss:0.169138
[16] validation_0-error:0.01719   validation_0-logloss:0.157988  validation_1-error:0.017674  validation_1-logloss:0.158345
[17] validation_0-error:0.01719   validation_0-logloss:0.149407  validation_1-error:0.017674  validation_1-logloss:0.149731
```

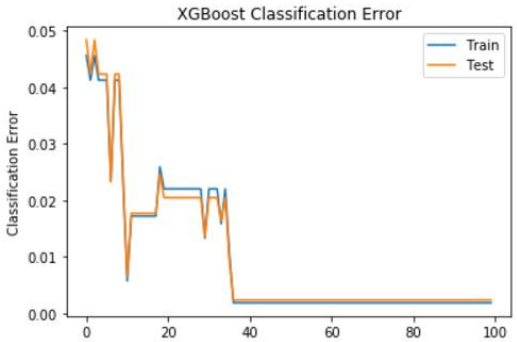
## 2.7.4 模型性能测验---绘制学习曲线

In [10]:
```
#5.模型在每次校验集上的性能存在模型中，
可用来进一步进行分析 model.evals result() 返回一个字典：评估数据集和分数
#显示学习曲线
```

In [12]:
```
# retrieve performance metrics
results = bst.evals_result()
#print(results)


epochs = len(results['validation_0']['error'])
x_axis = range(0, epochs)

# plot log loss
fig, ax = pyplot.subplots()
ax.plot(x_axis, results['validation_0']['logloss'], label='Train')
ax.plot(x_axis, results['validation_1']['logloss'], label='Test')
ax.legend()
pyplot.ylabel('Log Loss')
pyplot.title('XGBoost Log Loss')
pyplot.show()

# plot classification error
fig, ax = pyplot.subplots()
ax.plot(x_axis, results['validation_0']['error'], label='Train')
ax.plot(x_axis, results['validation_1']['error'], label='Test')
ax.legend()
pyplot.ylabel('Classification Error')
pyplot.title('XGBoost Classification Error')
pyplot.show()
```

绘图结果如下：



### 2.7.5 测试

```
In [13]: #6.测试--模型训练好后，可以用训练好的模型对测试数据进行预测
         # make prediction
         preds = bst.predict(X_test)
         predictions = [round(value) for value in preds]

         test_accuracy = accuracy_score(y_test, predictions)
         print("Test Accuracy: %.2f%%" % (test_accuracy * 100.0))

         Test Accuracy: 99.81%

In [ ]:
```

## 2.8 Early-Stop

一种防止训练复杂模型过拟合的方法

-监控模型在校验集上的性能，如果在经过固定次数的迭代后，校验集上的性能不在提高的时候，结束训练过程。

-当在测试集上的训练下降而在训练集上的性能还提高的时候，发生了过拟合。

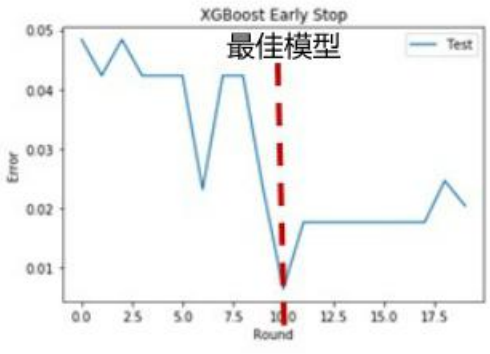
-防火防盗防过拟合

设置 early-stop 代码如下：

```
# 设置boosting迭代计算次数
num_round = 100
eval_set = [(X_validate, y_validate)]
bst.fit(X_train, y_train, early_stopping_rounds=10, eval_metric="error",
eval_set=eval_set, verbose=True)
```

通过下图看到第 10 次的错误率达到最低，是模型的最佳点。

```
[6]     validation_0-error:0.023256
[7]     validation_0-error:0.042326
[8]     validation_0-error:0.042326
[9]     validation_0-error:0.023256
[10]    validation_0-error:0.006512
[11]    validation_0-error:0.017674
[12]    validation_0-error:0.017674
[13]    validation_0-error:0.017674
[14]    validation_0-error:0.017674
[15]    validation_0-error:0.017674
[16]    validation_0-error:0.017674
[17]    validation_0-error:0.017674
[18]    validation_0-error:0.024651
[19]    validation_0-error:0.020465
[20]    validation_0-error:0.020465
Stopping. Best iteration:
[10]    validation_0-error:0.006512
```



## 2.8.1 导包和加载数据

In [1]:
```
#1. 导入xgboost包
import xgboost as xgb
from xgboost import XGBClassifier

# 加载LibSVM格式数据模块
from sklearn.datasets import load_svmlight_file

from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

from matplotlib import pyplot
```

```
D:\ProgramCJ\Anaconda\Anaconda3\lib\site-packages\sklearn\cross_va
 0.18 in favor of the model_selection module into which all the re
 the new CV iterators are different from that of this module. This
  "This module will be removed in 0.20.", DeprecationWarning)
```

In [2]:
```
#2.读取源数据
#my_workpath = './data/'
X_train,y_train = load_svmlight_file("D:\\BigData\\xgboost_Model\\
X_test,y_test = load_svmlight_file("D:\\BigData\\xgboost_Model\\Fi

X_train.shape
```

Out[2]: (6513, 126)

## 2.8.2 训练数据及给定模型参数

In [3]:
```
#3.训练集和测试集分离
# split data into train and test sets, 1/3的训练数据作为校验数据
seed = 7
test_size = 0.33
X_train_part, X_validate, y_train_part, y_validate= train_test_split(X_train,
    random_state=seed)

X_train_part.shape
X_validate.shape
```

Out[3]: (2150, 126)

In [4]:
```
#4, 训练参数设置
#max_depth: 树的最大深度。缺省值为6, 取值范围为: [1, ∞] eta: 为了防止过拟合,
#其他参数取默认值。
```

In [6]:
```
# specify parameters via map
param = {'max_depth':2, 'eta':1, 'silent':0, 'objective':'binary:logistic' }
print(param)
```

```
{'max_depth': 2, 'eta': 1, 'silent': 0, 'objective': 'binary:logistic'}
```

## 2.8.3 训练模型

In [7]:
```
#5.训练模型
```

In [8]:
```
# 设置boosting迭代计算次数
num_round = 100

#bst = XGBClassifier(param)
#bst = XGBClassifier()
bst =XGBClassifier(max_depth=2, learning_rate=0.1, n_estimators=num_round, silent=True, objective='binary:logistic')

#eval_set = [(X_train_part, y_train_part), (X_validation, y_validation)]
#bst.fit(X_train_part, y_train_part, eval_metric=["error", "logloss"], eval_set=eval_set, verbose=False)

eval_set =[(X_validate, y_validate)]
bst.fit(X_train_part, y_train_part, early_stopping_rounds=10, eval_metric="error",
    eval_set=eval_set, verbose=True)
```

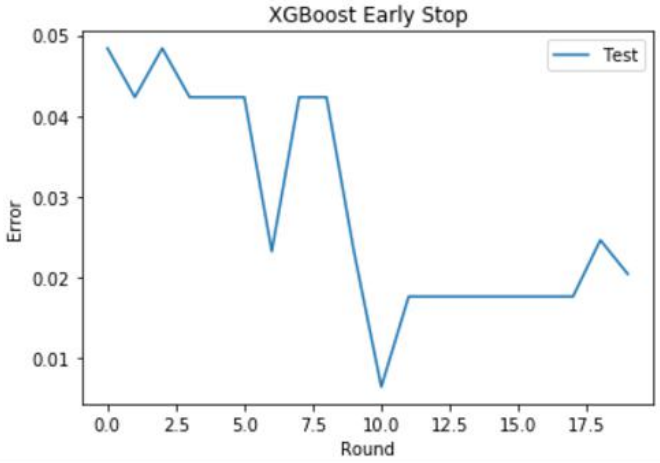
```
[0]     validation_0-error:0.048372
Will train until validation_0-error hasn't improved in 10 rounds.
[1]     validation_0-error:0.042326
[2]     validation_0-error:0.048372
[3]     validation_0-error:0.042326
[4]     validation_0-error:0.042326
[5]     validation_0-error:0.042326
[6]     validation_0-error:0.023256
[7]     validation_0-error:0.042326
[8]     validation_0-error:0.042326
[9]     validation_0-error:0.023256
[10]    validation_0-error:0.006512
[11]    validation_0-error:0.017674
[12]    validation_0-error:0.017674
[13]    validation_0-error:0.017674
[14]    validation_0-error:0.017674
[15]    validation_0-error:0.017674
[16]    validation_0-error:0.017674
[17]    validation_0-error:0.017674
[18]    validation_0-error:0.024651
[19]    validation_0-error:0.020465
[20]    validation_0-error:0.020465
Stopping. Best iteration:
[10]    validation_0-error:0.006512
```

## 2.8.4 根据学习曲线查看模型参数

#模型在每次校验集上的性能存在模型中，可用来进一步进行分析 model.evals result() 返回一个字典：评估数据集和分数。

```
In [ ]: #6.学习曲线
        #模型在每次校验集上的性能存在模型中，可用来进一步进行分析 model.eva
```

```
In [9]: # retrieve performance metrics
        results = bst.evals_result()
        #print(results)

        epochs = len(results['validation_0']['error'])
        x_axis = range(0, epochs)

        # plot log loss
        fig, ax = pyplot.subplots()
        ax.plot(x_axis, results['validation_0']['error'], label='Test')
        ax.legend()
        pyplot.ylabel('Error')
        pyplot.xlabel('Round')
        pyplot.title('XGBoost Early Stop')
        pyplot.show()
```




### 2.8.5 模型测试

```
In [10]: #7.测试
         # make prediction
         preds = bst.predict(X_test)
         predictions = [round(value) for value in preds]

         test_accuracy = accuracy_score(y_test, predictions)
         print("Test Accuracy: %.2f%%" % (test_accuracy * 100.0))
```

```
Test Accuracy: 97.27%
```

```
In [ ]:
```

# 2.9 交叉验证

一次的结果不具有代表性，高考中的一考定终生的说法其实就不有代表性。
我们在机器学习中更多的会使用交叉验证的方式去处理数据。
Train_test_split 将训练数据的一部分留出来做校验，不参与模型参数训练。
--优点：速度快

--缺点：训练数据少，一次校验集的划分会带来随机性

--适合训练数据规模较大的情况（如上百万条记录）

--适合较慢的机器学习模型

我们针对 train_test_split 的特点，使用交叉验证来减少这种随机性，但一般时间会比较久。

K 折交叉验证：将训练数据等分成 k 份（k 通常取值为 3，5 或 10）

--重复 k 次：每次留出一份做校验，其余 k-1 份做训练。

--K 次校验集上的平均性能视为模型在测试集上的性能评估。该估计比 train_test_split 得到的估计方差更小。

```
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score        #对给定参数的单个模型评估

kfold = KFold(n_splits=10, random_state=7)
results = cross_val_score(model, X, Y, cv=kfold)
print("Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

其中 KFold 函数中的 n_splits 时将数据集划分成几份，这里指定的是 10 份。

StratifiedKFold 采用分层抽样，对每一个类别的按相同比例来抽样。因此当每类样本不均衡或类别较多时，采用这种方法可以将数据集中每一类样本的数据等分。

## 2.9.1 导包和读入数据

In [8]:
```
# 1.运行 xgboost安装包中的示例程序
from xgboost import XGBClassifier

# 加载LibSVM格式数据模块
from sklearn.datasets import load_svmlight_file

#from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold
from sklearn.model_selection import cross_val_score

from sklearn.metrics import accuracy_score

from matplotlib import pyplot
```

In [2]:
```
#2.读取源数据
#my_workpath = './data/'
X_train,y_train = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_
X_test,y_test = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_La

X_train.shape
```

Out[2]: (6513, 126)

## 2.9.2 设置参数和训练模型

```
In [4]: #3.设置模型参数,按之前即可
        # specify parameters via map
        param = {'max_depth':2, 'eta':1, 'silent':0, 'objective':'binary:logistic' }
        print(param)
```

```
{'max_depth': 2, 'eta': 1, 'silent': 0, 'objective': 'binary:logistic'}
```

```
In [ ]: #4.构造模型
```

```
In [5]: # 设置boosting迭代计算次数
        num_round = 2
        #num_round = range(1, 101)
        #param_grid = dict(n_estimators=num_round)

        #bst = XGBClassifier(param)
        bst =XGBClassifier(max_depth=2, learning_rate=0.1,n_estimators=num_round,
                           silent=True, objective='binary:logistic')
```

## 2.9.3 交叉验证得到平均准确率

```
In [13]: #5.交叉验证，速度可能很慢
         # stratified k-fold cross validation evaluation of xgboost model
         #kfold = KFold(n_splits=10, random_state=7)
         kfold = StratifiedKFold(n_splits=10, random_state=7)
         #fit_params = {'eval_metric':"logloss"}
         #results = cross_val_score(bst, X_train, y_train, cv=kfold, fit_params)
         results = cross_val_score(bst, X_train, y_train, cv=kfold)
         print(results)
         print("CV Accuracy: %.2f%% (%.2f%%)" % (results.mean()*100, results.std()*100))
```

```
[ 0.69478528  0.85276074  0.95398773  0.97235023  0.96006144  0.98771121
  1.          1.          0.96927803  0.97695853]
CV Accuracy: 93.68% (9.00%)
```

0.18 之前的版本更改：

```
def test_kfold():
    from sklearn.cross_validation import StratifiedKFold
    from sklearn.cross_validation import cross_val_score
    kfold = StratifiedKFold(y=y_train,n_folds=10, random_state=7)
    result = cross_val_score(bst, X_train, y_train, cv=kfold)
    print (result)
    print ("CV Accuracy:%.2f%% (%.2f%%)" % (result.mean() * 100, result.std() * 100))

#4.交叉验证得到平均准确率
if __name__=='__main__':
    test_kfold()
    # [0.69478528  0.85276074  0.95398773  0.97235023  0.96006144  0.98771121
    #  1.          1.          0.96927803  0.97695853]
    # CV
    # Accuracy:93.68 % (9.00 %)
```

注意在0.18之前的版本分层抽样需要制定y的值，这里指定的是y_train

在第一则数据上的正确率为 0.69，第二则为 0.85 等等，将所有的正确率求和在求一个平均做为模型总体的正确率，即为 93.68%。

# 2.10GridSearchCV 网格搜索优化

给定一组参数的时候我们可以得到模型的评价，当参数多的时候我们希望能够自动得到模型的评价指标，如给出一个数据集的 10 个模型，最后能给出这 10 个模型那个模型是最好的。之前是用循环语句实现的，sklearn 帮我们实现了，我们只需要将 GridSearchCV 包 import 到当前空间即可使用。

一个参数可能组成空间中的一个点，两个参数组成空间中的一条线，N 个点会组成空间中的网结构。CV 的含义就是交叉验证的意思。

下面例子我们根据迭代次数(也就是合适的树的数目)来选择最佳的模型，设置参数评估的集合,n_estimator 迭代次数，范围为 1-50。GridSearchCV 函数中第一个参数是模型 bst（也就是模型），第二个参数是参数评估集合 param_grid,第三个参数是交叉验证的数目为 5（5 则交叉验证）。然后在传入 fit 进行训练即可，最后的输出结果放在 grid_score 中，最佳参模型和最佳模型得分放在 best_params 和 best_score 中。

该方法应用很广泛。

```
from sklearn.grid_search import GridSearchCV
# 设置boosting迭代计算次数搜索范围
param_test = { 'n_estimators': range(1, 51, 1)}
clf = GridSearchCV(estimator = bst, param_grid = param_test, cv=5)
clf.fit(X_train, y_train)
clf.grid_scores_, clf.best_params_, clf.best_score_
```

GridSearchCV 的参考代码如下：

- GridSearchCV(*estimator*, *param_grid*, *scoring=None*, *fit_params=None*, *n_jobs=1*, *iid=True*, *refit=True*, *cv=None*, *verbose=0*, *pre_dispatch='2*n_jobs'*, *error_score='raise'*, *return_train_score=True*)
  - Refit the best estimator with the entire dataset

Scoring 表示用什么准则来评价，比如二分类问题使用逻辑斯特损失函数来做，n_jobs 表示是否并行，Refit 是使用最佳参数在全部数据上训练的模型，，默认为 Ture。也就是说我们做 10 则交叉验证，实际上是做 11 次，最后一次是将得到的最佳参数应用于所有数据集上的情况。

## 2.10.1 导包和加载数据

```
In [15]: # 1.运行 xgboost安装包中的示例程序
from xgboost import XGBClassifier

# 加载LibSVM格式数据模块
from sklearn.datasets import load_svmlight_file

from sklearn.grid_search import GridSearchCV

from sklearn.metrics import accuracy_score

from matplotlib import pyplot
```

```
In [2]: #2.读取源数据
#my_workpath = './data/'
X_train,y_train = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_Lesson\\Le
X_test,y_test = load_svmlight_file("D:\\BigData\\xgboost_Model\\First_Lesson\\Lec1

X_train.shape
```

```
Out[2]: (6513, 126)
```

## 2.10.2 设置参数和训练模型

```
In [4]: #3.设置参数
# specify parameters via map
params = {'max_depth':2, 'eta':0.1, 'silent':0, 'objective':'binary:logistic' }
print(params)
```

```
{'max_depth': 2, 'eta': 0.1, 'silent': 0, 'objective': 'binary:logistic'}
```

```
In [7]: #4.训练模型
#bst = XGBClassifier(param)
bst =XGBClassifier(max_depth=2, learning_rate=0.1, silent=True, objective='binary:logistic')
```

## 2.10.3 网格交叉检验

在 Python3 中，range 函数已经没有返回列表功能，需要借助 list()来进行强制转换，代码如下：

```
In [60]: #5.网格交叉验证
# 设置boosting迭代计算次数
param_test = {
    'n_estimators':list(range(1, 51, 1)),
}
lista =[param_test]
```

```
In [61]: clf = GridSearchCV( estimator = bst, param_grid = param_test, scoring='accuracy', cv=5)
```

```
In [62]: clf.fit(X_train, y_train)
clf.grid_scores_, clf.best_params_, clf.best_score_
```

Python2 是以下写法：

```
In [7]: # 设置boosting迭代计算次数
param_test = {
 'n_estimators': range(1, 51, 1)
}
clf = GridSearchCV(estimator = bst, param_grid = param_test, scoring='accuracy', cv=5)
clf.fit(X_train, y_train)
clf.grid_scores_, clf.best_params_, clf.best_score_
```

隐藏空白

```
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 33},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 34},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 35},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 36},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 37},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 38},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 39},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 40},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 41},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 42},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 43},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 44},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 45},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 46},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 47},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 48},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 49},
 mean: 0.98419, std: 0.02040, params: {'n_estimators': 50}],
{'n_estimators': 30},
0.9841854752034392)
```

**注：全局函数 xrange()**

在 Python 2 里，有两种方法来获得一定范围内的数字：*range*()，它返回一个列表，还有 *range*()，它返回一个迭代器。在 Python 3 里，range()返回迭代器，xrange()不再存在了。

| Notes | Python 2 | Python 3 |
|---|---|---|
| ① | xrange(10) | range(10) |
| ② | a_list = range(10) | a_list = list(range(10)) |
| ③ | [i for iin xrange(10)] | [i for iin range(10)] |
| ④ | for i in range(10): | *no change* |
| ⑤ | sum(range(10)) | *no change* |

（1）在最简单的情况下，2to3 会简单地把 xrange()转换为 range()。

（2）如果你的 Python 2 代码使用 range()，2to3 不知道你是否需要一个列表，或者是否一个迭代器也行。出于谨慎，2to3 可能会报错，**然后使用 list()把 range()的返回值强制转换为列表类型。**

（3）如果在列表解析里有 xrange()函数，就没有必要将其返回值转换为一个列表，因为列表解析对迭代器同样有效。

（4）类似的，for 循环也能作用于迭代器，所以这里也没有改变任何东西。

（5）函数 sum()能作用于迭代器，所以 2to3 也没有在这里做出修改。就像返回值为视图(view)而不再是列表的字典类方法一样，这同样适用于 min()，max()，sum()，list()，tuple()，set()，sorted()，any()，all()。

### 2.10.4 模型测试

In [8]:
```
#make prediction
preds = clf.predict(X_test)
predictions = [round(value) for value in preds]

test_accuracy = accuracy_score(y_test, predictions)
print("Test Accuracy of gridsearchcv: %.2f%%" % (test_accuracy * 100.0))
```

```
Test Accuracy of gridsearchcv: 97.27%
```

## 2.11 模型评估总结

（1）通常 k-折交叉验证是评估机器学习模型的黄金准则（K=3,5,10）
（2）当类别较多时，或者每类样本不均衡时，采用 strtified 交叉验证。
（3）当训练数据集很大，train/test split 带来的模型性能估计偏差很小，或者模型训练很慢的时候，采用 train/test split。
（4）如果不清楚用那种模型评估方法，对回归问题，采用 10-fold cross-validation；对分类问题采用 stratified 10-fold cross-validation。

Stratified 用分层的思想，保证在每一个数据中是按比例分布的。