Libsvm和Liblinear都是国立台湾大学的Chih-Jen Lin博士开发的，Libsvm主要是用来进行非线性svm 分类器的生成，提出有一段时间了，而Liblinear则是去年才创建的，主要是应对large-scale的data classification，因为linear分类器的训练比非线性分类器的训练计算复杂度要低很多，时间也少很多，而且在large scale data上的性能和非线性的分类器性能相当，所以Liblinear是针对大数据而生的。

两者都是一个跨平台的通用工具库，支持windows/linux/mac os,代码本身是c++写的，同时也有matlab，python，java，c/c++扩展接口，方便不同语言环境使用，可以说是科研和企业人员的首选！像我这样在学校的一般用matlab/c++，而我同学在百度则主要用的是python/c++，所以只是各自侧重不一样，但所使用的核心还是其svm库。

[http://blog.sina.com.cn/s/blog\_5bd2cb260100ev25.html](http://blog.sina.com.cn/s/blog_5bd2cb260100ev25.html" \t "http://blog.csdn.net/sunlylorn/article/details/_blank) 这位网友对libsvm在matlab中的使用说明的很详细，可以参考下。

有关Liblinear和Libsvm各自的优势可以归纳如下：

1.libsvm用来就解决通用典型的分类问题

2.liblinear主要为大规模数据的线性模型设计

it can be able to handle large-scaled dataset 可以用来处理大规模的数据

it runs really faster than libsvm because it doesn't have to compute the kernel for any two points 由于采用线性核,所以不需要计算kernel value,速度更快

trust region method for optimization looks new for machine learning people

对于什么时候用线性模型的问题，我想上面的我举的例子用linear classifier就比较好，非线性分类不一定比线性分类器好，尤其是在样本及其有限，同时特征维度很高的情况下，因为样本有限的情况下，kernel map通常不准确，很有可能错误地划分类别空间，可能造成比线性模型更差的结果。

说到scale，我建议不要用libsvm里自带的scale，因为一旦使用这个工具，它就会把原来稀疏的数据，变成非稀疏的格式，这样不但会生成非常大的数据文件，而且liblinear对稀疏数据快速处理的优势就不能体现出来了。因此，要scale，就自己写一个，以保持原来稀疏的格式

liblinear的好处就是速度快，尤其是对稀疏的特征。缺点就是太吃内存了。10G的数据量需要接近50G的内存，数据量再大就没法做了 。

另外，对于多分类问题以及核函数的选取，以下经验规则可以借鉴：

如果如果特征数远远大于样本数的情况下,使用线性核就可以了.

如果特征数和样本数都很大,例如文档分类,一般使用线性核, LIBLINEAR比LIBSVM速度要快很多.

如果特征数远小于样本数,这种情况一般使用RBF.但是如果一定要用线性核,则选择LIBLINEAR较好,而且使用-s 2选项。

LibSVM使用说明

Libsvm是个简单、易用、并且高效的用于SVM分类和回归的软件。它能解决C-SVM分类、nu-SVM分类、单类SVM、epsilon-SVM回归以及nu-SVM回归问题。它还能提供一个自动模型选择工具用于C-SVM分类。

1. 安装以及数据格式

在unix系统，使用make来建立svm-train和svm-predict程序。不使用任何参数运行，可以显示它们的用法。对于其他的系统，可以使用Makefile来建立它们，或者使用预创建的库（windows在对应文件夹下）。

如果刚刚接触SVM并且数据不是很大，可以去‘tools’目录，在安装后使用easy.py，它能够自动完成所有事情，从数据scaling到参数选择。用法为：

Easy.py training\_file [testing\_file]

更多的说明可以见‘tools/README’

数据的格式如下：

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

.

.

.

每行都是一个样本，结尾为换行符’ \n ’。

对于分类，<label>是一个整数，代表类型标签（支持多分类）。

对于回归，<label>是一个目标值，可以是任意实数。

对于单类SVM，标签值可以不使用，所以可以是任何值。

<index1>:<value1>组给出特征值：<index>是一个整数，从1开始；<value>是一个实数。唯一的例外在于预计算核，<index>从0开始。

测试文件中的标签值只是用于计算准确度或误差，如果未知，可以填入任何值。

一个包含在库文件中的样例分类数据名为“heart\_scale”。如果要检查你的数据是否正确，可以使用tools/checkdata.py。

输入‘svm-train heart\_scale’，程序会读入训练数据，并输出模型文件‘heart\_scale.model’。如果你有一个测试集名为‘heart\_s.t’，则输入’svm-predict heart\_scale.t heart\_scale.model output’,这样就可以在output文件中包含预测得到的类别标签。

对于分类，如果训练数据中只有一个类，svm-train程序会报出警报，说明训练数据非常不均衡。在测试过程中，训练数据中的标签会直接返回。

有一些其他有用的程序：

Svm-scale：用于缩放输入数据的工具

Svm-toy：简单的图形化界面，用来展示SVM在平面中分割数据；还能在窗口中画数据点。Use "change" button to choose class 1, 2 or 3 (i.e., up to three classes are supported), "load" button to load data from a file, "save" button to save data to a file, "run" button to obtain an SVM model, and "clear" button to clear the window.

注意：load和save在分类和回归中都考虑密集的数据格式。对于分类，每个数据点有一个标签（颜色1 2 3）和[0 1]范围内的坐标值；对于回归问题，每个数据点有一个目标值y坐标值以及一个状态值x坐标。

在对应的文件夹内使用make建立该工具。

2. ‘svm-train’用法

Svm-train [option] training\_set\_file [model\_file]

Options:

-s svm\_type : set type of SVM (default 0)

0 -- C-SVC (multi-class classification)

1 -- nu-SVC (multi-class classification)

2 -- one-class SVM

3 -- epsilon-SVR (regression)

4 -- nu-SVR (regression)

-t kernel\_type : set type of kernel function (default 2)

0 -- linear: u'\*v

1 -- polynomial: (gamma\*u'\*v + coef0)^degree

2 -- radial basis function: exp(-gamma\*|u-v|^2)

3 -- sigmoid: tanh(gamma\*u'\*v + coef0)

4 -- precomputed kernel (kernel values in training\_set\_file)

-d degree : set degree in kernel function (default 3)

-g gamma : set gamma in kernel function (default 1/num\_features)

-r coef0 : set coef0 in kernel function (default 0)

-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)

-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)

-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)

-m cachesize : set cache memory size in MB (default 100)

-e epsilon : set tolerance of termination criterion (default 0.001)

-h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)

-b probability\_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)

-wi weight : set the parameter C of class i to weight\*C, for C-SVC (default 1)

-v n: n-fold cross validation mode

-q : quiet mode (no outputs)

The k in the -g option means the number of attributes in the input data.

option -v randomly splits the data into n parts and calculates cross

validation accuracy/mean squared error on them.

3. ‘Svm-predict’用法

svm-predict [options] test\_file model\_file output\_file

options:

-b probability\_estimates: whether to predict probability estimates, 0 or 1 (default 0); for one-class SVM only 0 is supported

model\_file is the model file generated by svm-train.

test\_file is the test data you want to predict.

svm-predict will produce output in the output\_file.

4. ‘svm-scale’用法

svm-scale [options] data\_filename

options:

-l lower : x scaling lower limit (default -1)

-u upper : x scaling upper limit (default +1)

-y y\_lower y\_upper : y scaling limits (default: no y scaling)

-s save\_filename : save scaling parameters to save\_filename

-r restore\_filename : restore scaling parameters from restore\_filename

5. 使用技巧

\* Scale your data. For example, scale each attribute to [0,1] or [-1,+1].

\* For C-SVC, consider using the model selection tool in the tools directory.

\* nu in nu-SVC/one-class-SVM/nu-SVR approximates the fraction of training

errors and support vectors.

\* If data for classification are unbalanced (e.g. many positive and

few negative), try different penalty parameters C by -wi (see

examples below).

\* Specify larger cache size (i.e., larger -m) for huge problems.

Examples

========

> svm-scale -l -1 -u 1 -s range train > train.scale

> svm-scale -r range test > test.scale

Scale each feature of the training data to be in [-1,1]. Scaling

factors are stored in the file range and then used for scaling the

test data.

> svm-train -s 0 -c 5 -t 2 -g 0.5 -e 0.1 data\_file

Train a classifier with RBF kernel exp(-0.5|u-v|^2), C=10, and

stopping tolerance 0.1.

> svm-train -s 3 -p 0.1 -t 0 data\_file

Solve SVM regression with linear kernel u'v and epsilon=0.1

in the loss function.

> svm-train -c 10 -w1 1 -w-2 5 -w4 2 data\_file

Train a classifier with penalty 10 = 1 \* 10 for class 1, penalty 50 =

5 \* 10 for class -2, and penalty 20 = 2 \* 10 for class 4.

> svm-train -s 0 -c 100 -g 0.1 -v 5 data\_file

Do five-fold cross validation for the classifier using

the parameters C = 100 and gamma = 0.1

> svm-train -s 0 -b 1 data\_file

> svm-predict -b 1 test\_file data\_file.model output\_file

Obtain a model with probability information and predict test data with

probability estimates

Liblinear使用说明

LIBLINEAR是一个解决大规模正则化线性分类和回归问题的库。目前它支持：

--L2正则化逻辑回归/L2损失支持向量分类/L1损失支持向量分类

--L1正则化L2损失支持向量分类/L1正则化逻辑回归

--L2正则化L2损失支持向量回归/L1损失支持向量回归

1. 什么时候使用 LIBLINEAR 而不是 LIBSVM？

对于一些有或没有非线性分布的大型数据，能给出相似的表现。不使用核，线性分类器或者回归模型能够有效的训练大规模数据集。这些数据通常有大规模特征，比如文本分类。

注意：虽然libnear通常都速度很快，但是它的默认求解器在某些情况下也会变慢（比如数据没有scaled或者参数C很大）。可以参考SVM指导手册的附录B来处理这些情况。

如果是新手并且你的数据集不是很大，你应该首先考虑LIBSVM。

2. 快速开始

阅读installation来安装LIBLINEAR。安装完成以后，有程序名为train和predict分别用于训练和测试。

关于数据格式，查看LIBSVM的README。注意特征序号从1开始。

为了得到跟好的结果，有时需要进行数据缩放。可以使用LIBSVM中的svm-scale程序。对于大规模以及系数的数据，使用‘-l 0’来保持稀疏性。

本软件使用一些level-1 BLAS次路径，需要的函数包含在此库文件中。如果机器上有有效的BLAS库，则可以修改makefile来使用它：

取消该行标记： #LIBS ?= -lblas

标记该行: #LIBS ?=blas/blas.a

3. `train' Usage

train [options] training\_set\_file [model\_file]

options:

-s type : set type of solver (default 1)

for multi-class classification

0 -- L2-regularized logistic regression (primal)

1 -- L2-regularized L2-loss support vector classification (dual)

2 -- L2-regularized L2-loss support vector classification (primal)

3 -- L2-regularized L1-loss support vector classification (dual)

4 -- support vector classification by Crammer and Singer

5 -- L1-regularized L2-loss support vector classification

6 -- L1-regularized logistic regression

7 -- L2-regularized logistic regression (dual)

for regression

11 -- L2-regularized L2-loss support vector regression (primal)

12 -- L2-regularized L2-loss support vector regression (dual)

13 -- L2-regularized L1-loss support vector regression (dual)

-c cost : set the parameter C (default 1)

-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)

-e epsilon : set tolerance of termination criterion

-s 0 and 2

|f'(w)|\_2 <= eps\*min(pos,neg)/l\*|f'(w0)|\_2,

where f is the primal function and pos/neg are # of

positive/negative data (default 0.01)

-s 11

|f'(w)|\_2 <= eps\*|f'(w0)|\_2 (default 0.001)

-s 1, 3, 4 and 7

Dual maximal violation <= eps; similar to libsvm (default 0.1)

-s 5 and 6

|f'(w)|\_1 <= eps\*min(pos,neg)/l\*|f'(w0)|\_1,

where f is the primal function (default 0.01)

-s 12 and 13\n"

|f'(alpha)|\_1 <= eps |f'(alpha0)|,

where f is the dual function (default 0.1)

-B bias : if bias >= 0, instance x becomes [x; bias]; if < 0, no bias term added (default -1)

-wi weight: weights adjust the parameter C of different classes (see README for details)

-v n: n-fold cross validation mode

-C : find parameter C (only for -s 0 and 2)

-q : quiet mode (no outputs)

Option -v randomly splits the data into n parts and calculates cross

validation accuracy on them.

Option -C conducts cross validation under different C values and finds

the best one. This options is supported only by -s 0 and -s 2. If

the solver is not specified, -s 2 is used.

Formulations:

For L2-regularized logistic regression (-s 0), we solve

min\_w w^Tw/2 + C \sum log(1 + exp(-y\_i w^Tx\_i))

For L2-regularized L2-loss SVC dual (-s 1), we solve

min\_alpha 0.5(alpha^T (Q + I/2/C) alpha) - e^T alpha

s.t. 0 <= alpha\_i,

For L2-regularized L2-loss SVC (-s 2), we solve

min\_w w^Tw/2 + C \sum max(0, 1- y\_i w^Tx\_i)^2

For L2-regularized L1-loss SVC dual (-s 3), we solve

min\_alpha 0.5(alpha^T Q alpha) - e^T alpha

s.t. 0 <= alpha\_i <= C,

For L1-regularized L2-loss SVC (-s 5), we solve

min\_w \sum |w\_j| + C \sum max(0, 1- y\_i w^Tx\_i)^2

For L1-regularized logistic regression (-s 6), we solve

min\_w \sum |w\_j| + C \sum log(1 + exp(-y\_i w^Tx\_i))

For L2-regularized logistic regression (-s 7), we solve

min\_alpha 0.5(alpha^T Q alpha) + \sum alpha\_i\*log(alpha\_i) + \sum (C-alpha\_i)\*log(C-alpha\_i) - a constant

s.t. 0 <= alpha\_i <= C,

where

Q is a matrix with Q\_ij = y\_i y\_j x\_i^T x\_j.

For L2-regularized L2-loss SVR (-s 11), we solve

min\_w w^Tw/2 + C \sum max(0, |y\_i-w^Tx\_i|-epsilon)^2

For L2-regularized L2-loss SVR dual (-s 12), we solve

min\_beta 0.5(beta^T (Q + lambda I/2/C) beta) - y^T beta + \sum |beta\_i|

For L2-regularized L1-loss SVR dual (-s 13), we solve

min\_beta 0.5(beta^T Q beta) - y^T beta + \sum |beta\_i|

s.t. -C <= beta\_i <= C,

where

Q is a matrix with Q\_ij = x\_i^T x\_j.

If bias >= 0, w becomes [w; w\_{n+1}] and x becomes [x; bias].

The primal-dual relationship implies that -s 1 and -s 2 give the same

model, -s 0 and -s 7 give the same, and -s 11 and -s 12 give the same.

We implement 1-vs-the rest multi-class strategy for classification.

In training i vs. non\_i, their C parameters are (weight from -wi)\*C

and C, respectively. If there are only two classes, we train only one

model. Thus weight1\*C vs. weight2\*C is used. See examples below.

We also implement multi-class SVM by Crammer and Singer (-s 4):

min\_{w\_m, \xi\_i} 0.5 \sum\_m ||w\_m||^2 + C \sum\_i \xi\_i

s.t. w^T\_{y\_i} x\_i - w^T\_m x\_i >= \e^m\_i - \xi\_i \forall m,i

where e^m\_i = 0 if y\_i = m,

e^m\_i = 1 if y\_i != m,

Here we solve the dual problem:

min\_{\alpha} 0.5 \sum\_m ||w\_m(\alpha)||^2 + \sum\_i \sum\_m e^m\_i alpha^m\_i

s.t. \alpha^m\_i <= C^m\_i \forall m,i , \sum\_m \alpha^m\_i=0 \forall i

where w\_m(\alpha) = \sum\_i \alpha^m\_i x\_i, and C^m\_i = C if m = y\_i, C^m\_i = 0 if m != y\_i.

4. `predict' Usage

predict [options] test\_file model\_file output\_file

options:

-b probability\_estimates: whether to output probability estimates, 0 or 1 (default 0); currently for logistic regression only

-q : quiet mode (no outputs)

Note that -b is only needed in the prediction phase. This is different from the setting of LIBSVM.

5. Examples

> train data\_file

Train linear SVM with L2-loss function.

> train -s 0 data\_file

Train a logistic regression model.

> train -v 5 -e 0.001 data\_file

Do five-fold cross-validation using L2-loss SVM.

Use a smaller stopping tolerance 0.001 than the default 0.1 if you want more accurate solutions.

> train -C data\_file

Conduct cross validation many times by L2-loss SVM and find the parameter C which achieves the best cross validation accuracy.

> train -C -s 0 -v 3 -c 0.5 -e 0.0001 data\_file

For parameter selection by -C, users can specify other solvers (currently -s 0 and -s 2 are supported) and different number of CV folds. Further, users can use the -c option to specify the smallest C value of the search range. This setting is useful when users want to rerun the parameter selection procedure from a specified C under a different setting, such as a stricter stopping tolerance -e 0.0001 in the above example.

> train -c 10 -w1 2 -w2 5 -w3 2 four\_class\_data\_file

Train four classifiers:

positive negative Cp Cn

class 1 class 2,3,4. 20 10

class 2 class 1,3,4. 50 10

class 3 class 1,2,4. 20 10

class 4 class 1,2,3. 10 10

> train -c 10 -w3 1 -w2 5 two\_class\_data\_file

If there are only two classes, we train ONE model. The C values for the two classes are 10 and 50.

> predict -b 1 test\_file data\_file.model output\_file

Output probability estimates (for logistic regression only).

6. 库使用

6.1 数据结构

struct feature\_node

{

int index; //特征编号，即特征向量中第几个特征 特征标号从1开始

double value; //特征值

};

struct problem //定义问题

{

int l, n; //l表示训练数据的数量，n表示特征的数量维数（如果bia>=0，则也包含bia这个特征）

double \*y; //目标值（或标签值）的向量

struct feature\_node \*\*x; //x是一个双重指针（指针向量），x[i][j]代表第i个特征向量的不为零的特征数据（特征编号，特征值），这样设计是为了适应特征向量是稀疏向量的情况，存储和计算上都有优势

//如果bias>=0,x最后要加一个元素，变为x；bias

double bias; /\* < 0 if no bias term \*/ //如果>=0，则假设每组特征数据的后面多一个特征

};

/\*

结构体 problem例子：

如果有以下训练数据：

LABEL ATTR1 ATTR2 ATTR3 ATTR4 ATTR5

----- ----- ----- ----- ----- -----

1 0 0.1 0.2 0 0

2 0 0.1 0.3 -1.2 0

1 0.4 0 0 0 0

2 0 0.1 0 1.4 0.5

3 -0.1 -0.2 0.1 1.1 0.1

and bias = 1, then the components of problem are:

l = 5

n = 6

y -> 1 2 1 2 3

x -> [ ] -> (2,0.1) (3,0.2) (6,1) (-1,?)

[ ] -> (2,0.1) (3,0.3) (4,-1.2) (6,1) (-1,?)

[ ] -> (1,0.4) (6,1) (-1,?)

[ ] -> (2,0.1) (4,1.4) (5,0.5) (6,1) (-1,?)

[ ] -> (1,-0.1) (2,-0.2) (3,0.1) (4,1.1) (5,0.1) (6,1) (-1,?)

\*/

enum { L2R\_LR, L2R\_L2LOSS\_SVC\_DUAL, L2R\_L2LOSS\_SVC, L2R\_L1LOSS\_SVC\_DUAL, MCSVM\_CS, L1R\_L2LOSS\_SVC, L1R\_LR, L2R\_LR\_DUAL, L2R\_L2LOSS\_SVR = 11, L2R\_L2LOSS\_SVR\_DUAL, L2R\_L1LOSS\_SVR\_DUAL }; /\* solver\_type \*/

struct parameter

{

int solver\_type; //求解器类型，所有的类型见上面的enum，默认为 L2R\_L2LOSS\_SVC\_DUAL

/\* these are for training only \*/

double eps; /\* stopping criteria \*/ //停止迭代条件

double C; //约束条件惩罚因子

int nr\_weight; //weight\_label和weight向量的元素个数（权重数组的大小），如果不想改变任何类的惩罚项，则设置为0

int \*weight\_label; //权重标签数组，每个weight[i]对应一个weight\_label[i]，意味着类weight\_label[i]的惩罚有weight[i]来调节

double\* weight; //权重数组

double p; //支持向量回归损失的灵敏度

double \*init\_sol;

}; //为了避免参数错误，应该在调用train()函数以后，调用check\_parameter来检查

/\*

solver\_type can be one of L2R\_LR, L2R\_L2LOSS\_SVC\_DUAL, L2R\_L2LOSS\_SVC, L2R\_L1LOSS\_SVC\_DUAL, MCSVM\_CS, L1R\_L2LOSS\_SVC, L1R\_LR, L2R\_LR\_DUAL, L2R\_L2LOSS\_SVR, L2R\_L2LOSS\_SVR\_DUAL, L2R\_L1LOSS\_SVR\_DUAL.

for classification

L2R\_LR L2-regularized logistic regression (primal)

L2R\_L2LOSS\_SVC\_DUAL L2-regularized L2-loss support vector classification (dual)

L2R\_L2LOSS\_SVC L2-regularized L2-loss support vector classification (primal)

L2R\_L1LOSS\_SVC\_DUAL L2-regularized L1-loss support vector classification (dual)

MCSVM\_CS support vector classification by Crammer and Singer

L1R\_L2LOSS\_SVC L1-regularized L2-loss support vector classification

L1R\_LR L1-regularized logistic regression

L2R\_LR\_DUAL L2-regularized logistic regression (dual)

for regression

L2R\_L2LOSS\_SVR L2-regularized L2-loss support vector regression (primal)

L2R\_L2LOSS\_SVR\_DUAL L2-regularized L2-loss support vector regression (dual)

L2R\_L1LOSS\_SVR\_DUAL L2-regularized L1-loss support vector regression (dual)

\*/

struct model

{

struct parameter param; //模型参数

int nr\_class; /\* number of classes \*///类别的数量，特别的而对于回归问题，该参数为2

int nr\_feature; //特征的数量

double \*w; //特征的权重，它的尺寸为 nr\_feature\*nr\_class，如果nr\_class=2，则为nr\_feature；如果bias>=0,则w是一个（nr\_feature+1）\*nr\_class的向量

int \*label; /\* label of each class \*/ //每类的标签值

double bias;

};

函数：

struct model\* train(const struct problem \*prob, const struct parameter \*param);

//根据给定的数据集和参数，构造并返回一个线性分类或回归模型

void cross\_validation(const struct problem \*prob, const struct parameter \*param, int nr\_fold, double \*target);

//该函数进行交叉验证，数据被分割成nr\_fold组。在给定参数下，每一组都使用 其他组数据训练的模型 进行验证。 验证过程中的预测标签存在target的向量中

void find\_parameter\_C(const struct problem \*prob, const struct parameter \*param, int nr\_fold, double start\_C, double max\_C, double \*best\_C, double \*best\_rate);

//此函数类似交叉验证函数cross\_validation，不过与交叉验证函数需要特定的C参数不同的是，此函数通过在C=start\_C，2\*start\_C，4\*start\_C...条件下进行多次交叉验证，找到最高交叉验证准确率的最优参数C

double predict\_values(const struct model \*model\_, const struct feature\_node \*x, double\* dec\_values);

//该函数在向量dec\_values中给定nr\_w决定值

//对于回归问题或者类别数为2时，nr\_w=1；一个例外：对于多分类SVM（-s 4），如果只有两类，nr\_w=2；

//对于其他情况，nr\_w为类别的数量

//对于多分类问题，采用的是一对多的分类策略（-s 0,1,2,3,5,6,7） 以及 多类对多类的策略(-s 4) 返回的是具有最高决定值的类

double predict(const struct model \*model\_, const struct feature\_node \*x);

//对于分类模型，该函数对某个样本x返回预测类型；对于回归模型，该函数计算法某个样本的回归值

double predict\_probability(const struct model \*model\_, const struct feature\_node \*x, double\* prob\_estimates);

//该函数给出了nr\_class（类别数量）概率估计值，在向量prob\_estimates中。nr\_class能够通过函数get\_nr\_class中获取，最高概率的类将会返回，目前只支持逻辑回归的概率输出

int save\_model(const char \*model\_file\_name, const struct model \*model\_);

//将模型信息存储到一个文件中，返回0代表成功存储

struct model \*load\_model(const char \*model\_file\_name);

//从文件中读入模型信息

int get\_nr\_feature(const struct model \*model\_);

//返回模型的样本维度的大小

int get\_nr\_class(const struct model \*model\_);

//返回模型的类别数，回归模型返回2

void get\_labels(const struct model \*model\_, int\* label);

//将模型中的标签输入到label中，对于回归模型标签是不变的

double get\_decfun\_coef(const struct model \*model\_, int feat\_idx, int label\_idx);

//对于index=feat\_idx的特征以及label index=lable\_idx的类，返回系数

double get\_decfun\_bias(const struct model \*model\_, int label\_idx);

//对于给定label\_idx的类返回对应的bias大小

void free\_model\_content(struct model \*model\_ptr);

//释放一个模型结果所占用的内存

void free\_and\_destroy\_model(struct model \*\*model\_ptr\_ptr);

//释放模型占用的内存并摧毁模型结果

void destroy\_param(struct parameter \*param);

//释放一个参数集占用的内存

const char \*check\_parameter(const struct problem \*prob, const struct parameter \*param);

//检查参数是否在解决问题的范围内，该函数应该在调用train以及cross\_validation之前调用，如果参数有效则返回NULL，否则会返回一个错误消息

int check\_probability\_model(const struct model \*model);

//如果被检查的模型支持概率输出，则返回1，否则返回0

int check\_regression\_model(const struct model \*model);

//如果被检查的模型是一个回归模型，则返回1，否则返回0

void set\_print\_string\_function(void (\*print\_func) (const char\*));

//用户可以自定义自己的输出格式通过print\_func，使用set\_print\_string\_function（NULL）为默认输出

6.2 类结构设计

6.2.1 function 类

类function是一个损失函数（目标函数）的基类，其功能是计算损失函数值、梯度值、以及Hessian矩阵等，这些值都会用于牛顿法、坐标下降法和其他算法的的求解过程。从function派生出了一些类，用于实现各种损失函数，派生关系如下图所示：

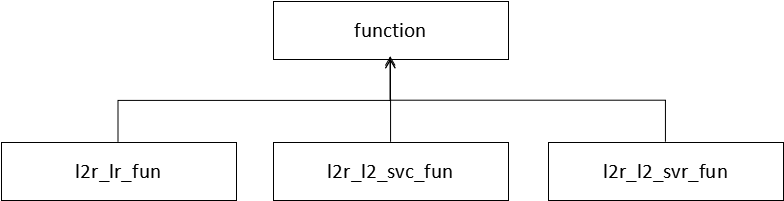


图12.5 function类的继承关系

下面先看function类的定义，这是一个虚基类，所有成员函数都留给派生类实现：

class function

{

public:

// 计算损失函数的值，w为权重数组，返回的是损失函数的值

virtual double fun(double \*w) = 0 ;

// 计算损失函数的梯度，w为权重数组，g为返回的梯度值

virtual void grad(double \*w, double \*g) = 0 ;

// 计算Hessian矩阵与向量的乘积，s为传入向量，Hs为返回的Hessian与向量s的乘积

virtual void Hv(double \*s, double \*Hs) = 0 ;

// 获取特征向量的维数，即变量数

virtual int get\_nr\_variable(void) = 0 ;

virtual ~function(void){}

};

类l2r\_lr\_fun继承自function类，实现了L2正则化logistic回归的损失函数，前面介绍过该函数的定义为：



类l2r\_l2\_svc\_fun是L2正则化L2损失函数支持向量机的损失函数，同样继承自function类，损失函数的计算公式为：



上面两个类都将用于TRON类的求解函数，为TRON提供计算函数值，梯度值和Hessian矩阵值的支持。

类l2r\_l2\_svr\_fun是L2正则化L2损失函数支持向量回归的目标函数，这个类只提供了计算目标函数、目标函数梯度的函数。

6.3 各种求解器的实现

6.3.1 TRON类

可信域牛顿法由类TRON实现，这个类为L2-正则化logistic回归和L2-正则化支持向量机的训练提供支持。

tron函数是可信域牛顿法求解的主体，w为返回的权重参数，直接按照之前的公式进行。

函数trcg实现了共轭梯度法搜索近似牛顿方向，具体参考前面的公式。g为梯度，s和r是公式中的相应向量。

6.3.2 Solver\_MCSVM\_CS类

Solver\_MCSVM\_CS实现了求解多类SVM分类对偶问题的坐标下降法。