Invariant Inference Framework

Generated by Doxygen 1.8.11

Contents

Chapter 1

Invariant Inference Framework:

This is the result of our implementation of the paper An Invariant Inference Framework by Active Learning and SVMs by Li Jiaying.

For you to run the experiments on your own machine, please follow the steps below to set up your experiment environment.

Work on Invariant Inference Framework

To build the framework currently is very easy, there is not much dependencies you need to satisfy before build the whole project.

Dependencies, for Windows/Linux/MacOSX Users:

- cmake version 2.8 or later.
- libsvm remember to put {libsvm}/bin folder into $PATH.
- z3 For Windows users, please put z3 to the folder

  ```
  C:/Program Files
  ```

- klee This is optional currently.
- [Build tools](), such as make, Visual Studio 2015, or Xcode.

###Build InvariantInferenceFramework

```
git clone git@github.com:lijiaying/InvariantInferenceFramework.git
cd InvariantInferenceFramework
cd test
mkdir build
cd build
cmake .. -G [your platform]  // just use cmake .. if you are not sure
make
```

Add your tests to this framework

As InvariantInferenceFramework is integrated with your examples, you need to do some modification on source code level before you can test your examples.

- READ carefully one example file in test folder before you write your own test.
- rewrite your loop code in a function with the name you like, my_loop_example for instance.

- modify function and function name as parameter for register_target which is called by main function.
- rename your test file with the number of parameters and a "_" as prefix.
- modify the second line in CMakeLists.txt in the project folder as the numbers of parameter you need in your program.
- After the above step, you can make your project and then run the executable file.

Experiments results:

- `simple2`
- `simple3`
- `ex1`
- `f1a`
- `f2`
- `substring1`

Chapter 2

Bug List

File color.h

unset_console_color is set the console back to black background, white forground, no strong comparision instead of the previous setting.

File config.h

No known bugs.

File equation.h

No known bugs.

File iif.h

No found bugs

File iif_assert.h

unset_console_color is set the console back to black background, white forground, no strong comparision instead of the previous setting.

File instrumentation.h

No known bugs

File ml_algo.h

Chapter 3

Hierarchical Index

3.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

Chapter 6

Data Structure Documentation

6.1 Cache Class Reference

Public Member Functions

- Cache (int l, long int size)
- ∼Cache ()
- int get_data (const int index, Qfloat ∗∗data, int len)
- void swap_index (int i, int j)

6.1.1 Constructor & Destructor Documentation

6.1.1.1 Cache::Cache (int *l*, long int *size*)

6.1.1.2 Cache::∼Cache ( )

6.1.2 Member Function Documentation

6.1.2.1 int Cache::get_data (const int *index*, Qfloat ∗∗ *data*, int *len*)

6.1.2.2 void Cache::swap_index (int *i*, int *j*)

The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.2 decision_function Struct Reference

Data Fields

- double ∗ alpha
- double rho

6.2.1 Field Documentation

6.2.1.1 double∗ decision_function::alpha

6.2.1.2 double decision_function::rho

The documentation for this struct was generated from the following file:

- src/svm_core.cpp

6.3 Equation Class Reference

This class defines an equation by storing all its coefficiencies. An equation is regarded a hyperplane in math.

```
#include <equation.h>
```

Public Member Functions

- Equation ()

  Default constructor. Set all its elements to value 0.
- Equation (double a0,...)

  Most useful constructor Set its elements to the given values, order keeps The first element is Theta0.
- Equation (const Equation &equ)

  Copy constructor.
- Equation & operator= (const Equation &rhs)

  Overwrite = operator.
- bool imply (const Equation &e2)

  This imply method checks whether this equation object can imply another one or not That is to say: ∗this ==> e2 ?? ∗this is default equation left side.
- int linear_solver (Solution &sol)

  A shell on linear_solver(equ, sol)
- int is_similar (const Equation &e, int precision=PRECISION)

  This method is used to check whether ∗this equation is similar to given equation e or not. ∗this ∼= e ???
- int roundoff (Equation &e)

  Do roundoff job for an equation.

Static Public Member Functions

- static int linear_solver (const Equation ∗equ, Solution &sol)

  The solver for an Equation.
- static double calc (Equation &equ, double ∗sol)

  This static method is used to get the position info for the given point against given equation.

Data Fields

- double theta0
- double theta [VARS]

Friends

- std::ostream & operator<< (std::ostream &out, const Equation &equ)

  Output the equation in a readable format.

6.3.1 Detailed Description

This class defines an equation by storing all its coefficiencies. An equation is regarded a hyperplane in math.

theta0 + theta[0] ∗ x_0 + theta[1] ∗ x_1 + ... + theta[VARS] ∗ x_{VARS} >= 0

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Equation::Equation ( )

Default constructor. Set all its elements to value 0.

6.3.2.2 Equation::Equation (double *a0,* *...*)

Most useful constructor Set its elements to the given values, order keeps The first element is Theta0.

6.3.2.3 Equation::Equation (const Equation & *equ*)

Copy constructor.

Parameters

equ	The equation to be copied.

6.3.3 Member Function Documentation

6.3.3.1 static double Equation::calc (Equation & *equ,* double ∗ *sol*) [inline], [static]

This static method is used to get the position info for the given point against given equation.

It just substitutes variants with the given point.

Parameters

equ	is the given equation, should not be NULL
sol	is the tested solution, should not be NULL

Returns

The distance/value of the solution to the given equation

6.3.3.2 bool Equation::imply (const Equation & *e2*)

This imply method checks whether this equation object can imply another one or not That is to say: ∗this ==> e2 ?? ∗this is default equation left side.

Currently, it is based on Z3 prover. And the default precision is set to E-8 (2.8f), which is changeable if need

Parameters

e2	is the equation right side

Returns

bool true if yes, false if no.

6.3.3.3 int Equation::is_similar (const Equation & *e*, int *precision* = PRECISION)

This method is used to check whether ∗this equation is similar to given equation e or not. ∗this ∼= e ???

Parameters

precision	defines how much variance we can bare. The default is 4, which means we can bare 0.0001 difference. In this case 1 ∼=1.00001, but 1!∼=1.000011

6.3.3.4 int Equation::linear_solver (Solution & *sol*) [inline]

A shell on linear_solver(equ, sol)

Parameters

sol	set by callee as a solution to given object

Returns

int 0 if no error.

6.3.3.5 static int Equation::linear_solver (const Equation ∗ *equ*, Solution & *sol*) [inline], [static]

The solver for an Equation.

This method calcuate the most informative points in space It return a points really on the margin or next to the margin

Parameters

sol	is set by callee as a solution to given object contains the solution, integer format

Returns

int 0 if no error.

equ == NULL means no equation is specified So we randomly generate points in given scope [minv, maxv]

< a flag to justify whether all the coefficients are zeros...

If all the coefficients are zeros.... We just randomly pickup solutions to return

< pick store the dimension that should not generate randomly The algo is we generate numbers randomly, unless the picked dimension The picked dimension should be calcuate based on equation and other dimensions

sometimes we can not get solution between given scope we try 10 times, if still no suitable solution, we pick the last one...

6.3.3.6 Equation & Equation::operator= (const Equation & *rhs*)

Overwrite = operator.

This is needed when we want to delete a equation in an equation list We copy the next equation to the current one, and repeat this process until tails

Parameters

rhs	The right-hand-side equation of assignment

6.3.3.7 int Equation::roundoff (Equation & *e*)

Do roundoff job for an equation.

Sometimes the equation has ugly coefficiencies we want to make it elegent, which is the purpose of involing this method Currently we have not done much work on this We have not even use gcd function to adjust the coefficients.

For example. 1.2345 x1 >= 2.4690 ==> x1 >= 2 2 x1 >= 5.000001 ==> x1 >= 2.5

Parameters

e	Contains the equation that has already rounded off

Returns

int 0 if no error.

6.3.4 Friends And Related Function Documentation

6.3.4.1 std::ostream& operator<< (std::ostream & *out,* const Equation & *equ*) [friend]

Output the equation in a readable format.

Example: 2{0} + 3{1} >= 5

Parameters

equ	the equation to be ouput

6.3.5 Field Documentation

6.3.5.1 double Equation::theta[VARS]

6.3.5.2 double Equation::theta0

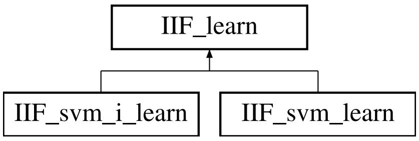
The documentation for this class was generated from the following files:

- include/equation.h
- src/equation.cpp

6.4 IIF_learn Class Reference

```
#include <iif_learn.h>
```

Inheritance diagram for IIF_learn:



Public Member Functions

- IIF_learn (States ∗gsets, int(∗func)(int ∗))
- IIF_learn ()
- void run_target (Solution &input)
- virtual int learn ()=0

Protected Member Functions

- void init_gsets ()

Protected Attributes

- States ∗ gsets
- int(∗ func)(int ∗)

6.4.1 Constructor & Destructor Documentation

6.4.1.1 IIF_learn::IIF_learn (**States** ∗ *gsets,* int(∗)(int ∗) *func*) [inline]

6.4.1.2 IIF_learn::IIF_learn ( ) [inline]

6.4.2 Member Function Documentation

6.4.2.1 void IIF_learn::init_gsets ( ) [inline], [protected]

6.4.2.2 virtual int IIF_learn::learn ( ) [pure virtual]

Implemented in IIF_svm_i_learn, and IIF_svm_learn.

6.4.2.3 void IIF_learn::run_target (**Solution** & *input*) [inline]

6.4.3 Field Documentation

6.4.3.1 int(∗ IIF_learn::func) (int ∗) [protected]

6.4.3.2 States∗ IIF_learn::gsets [protected]

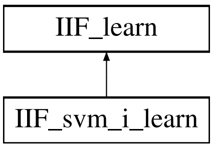
The documentation for this class was generated from the following file:

- include/iif_learn.h

6.5 IIF_svm_i_learn Class Reference

```
#include <iif_svm_i_learn.h>
```

Inheritance diagram for IIF_svm_i_learn:



Public Member Functions

- IIF_svm_i_learn (States ∗gsets, int(∗func)(int ∗), int max_iteration=max_iter)
- IIF_svm_i_learn ()
- virtual int learn ()

Protected Attributes

- SVM_I ∗ svm_i
- int max_iteration

Additional Inherited Members

6.5.1 Constructor & Destructor Documentation

6.5.1.1 IIF_svm_i_learn::IIF_svm_i_learn (States ∗ *gsets,* int(∗)(int ∗) *func,* int *max_iteration* = max_iter)

6.5.1.2 IIF_svm_i_learn::IIF_svm_i_learn ( )

6.5.2 Member Function Documentation

6.5.2.1 int IIF_svm_i_learn::learn ( ) [virtual]

Implements IIF_learn.

6.5.3 Field Documentation

6.5.3.1 int IIF_svm_i_learn::max_iteration [protected]

6.5.3.2 SVM_I∗ IIF_svm_i_learn::svm_i [protected]

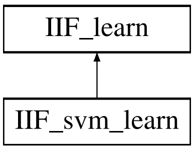
The documentation for this class was generated from the following files:

- include/iif_svm_i_learn.h
- src/iif_svm_i_learn.cpp

6.6 IIF_svm_learn Class Reference

```
#include <iif_svm_learn.h>
```

Inheritance diagram for IIF_svm_learn:



Public Member Functions

- IIF_svm_learn (States ∗gsets, int(∗func)(int ∗), int max_iteration=max_iter)
- IIF_svm_learn ()
- virtual int learn ()

Protected Attributes

- SVM ∗ svm
- int max_iteration

Additional Inherited Members

6.6.1 Constructor & Destructor Documentation

6.6.1.1 IIF_svm_learn::IIF_svm_learn (States ∗ *gsets,* int(∗)(int ∗) *func,* int *max_iteration* = max_iter)

6.6.1.2 IIF_svm_learn::IIF_svm_learn ( )

6.6.2 Member Function Documentation

6.6.2.1 int IIF_svm_learn::learn ( ) [virtual]

Implements IIF_learn.

6.6.3 Field Documentation

6.6.3.1 int IIF_svm_learn::max_iteration [protected]

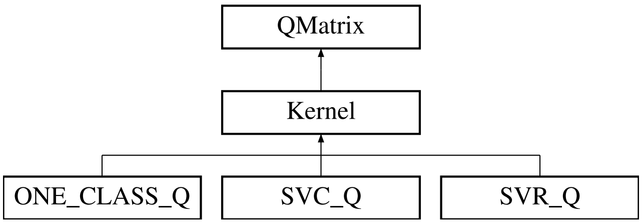
6.6.3.2 SVM∗ IIF_svm_learn::svm [protected]

The documentation for this class was generated from the following files:

- include/iif_svm_learn.h
- src/iif_svm_learn.cpp

6.7 Kernel Class Reference

Inheritance diagram for Kernel:



Public Member Functions

- Kernel (int l, svm_node ∗const ∗x, const svm_parameter &param)
- virtual ∼Kernel ()
- virtual Qfloat ∗ get_Q (int column, int len) const =0
- virtual double ∗ get_QD () const =0
- virtual void swap_index (int i, int j) const

Static Public Member Functions

- static double k_function (const svm_node ∗x, const svm_node ∗y, const svm_parameter &param)

Protected Attributes

- double(Kernel::∗ kernel_function)(int i, int j) const

6.7.1 Constructor & Destructor Documentation

6.7.1.1 Kernel::Kernel (int *l,* svm_node ∗const ∗ *x,* const svm_parameter & *param*)

6.7.1.2 Kernel::∼Kernel ( ) `[virtual]`

6.7.2 Member Function Documentation

6.7.2.1 virtual Qfloat∗ Kernel::get_Q (int *column,* int *len*) const `[pure virtual]`

Implements QMatrix.

Implemented in SVR_Q, ONE_CLASS_Q, and SVC_Q.

6.7.2.2 virtual double∗ Kernel::get_QD ( ) const `[pure virtual]`

Implements QMatrix.

Implemented in SVR_Q, ONE_CLASS_Q, and SVC_Q.

6.7.2.3 double Kernel::k_function (const svm_node ∗ *x,* const svm_node ∗ *y,* const svm_parameter & *param*) `[static]`

6.7.2.4 virtual void Kernel::swap_index (int *i,* int *j*) const `[inline]`, `[virtual]`

Implements QMatrix.

Reimplemented in SVR_Q, ONE_CLASS_Q, and SVC_Q.

6.7.3 Field Documentation

6.7.3.1 double(Kernel::∗ Kernel::kernel_function) (int i, int j) const [protected]

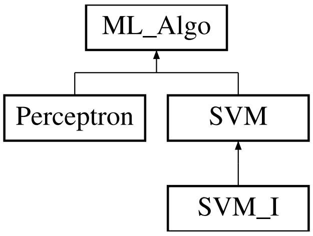
The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.8 ML_Algo Class Reference

```
#include <ml_algo.h>
```

Inheritance diagram for ML_Algo:



Public Member Functions

- ML_Algo ()
- virtual int prepare_training_data (States ∗gsets, int &pre_positive_size, int &pre_negative_size)=0

    init training data method. This should be called before any training happens.
- virtual int train ()=0

    The most important TRAIN method, which calls real training algorithm to do training.
- virtual double predict_on_training_set ()=0

    Calculate the predict precision of the training-model on the training set.
- virtual int check_question_set (States &qset)=0

    test on question state sets to see if there is an invalidation
- virtual int get_converged (Equation ∗previous_equations, int equation_num)=0

    check whether the training is converged or not
- virtual std::ostream & _print (std::ostream &out) const

    This is the function really called to output this object. We involve this as to support polymophism for operator <<.
- virtual int size ()=0

    This method returns the current problem size (the number of training states).
- virtual Equation ∗ roundoff (int &equation_num)=0

    Round off the whole training model.(equations)
- virtual int predict (double ∗x, int flag)=0

    Predict sample x against the whole training model.(equations)

Friends

- std::ostream & operator<< (std::ostream &out, const ML_Algo &mla)

    output the current trainig result of a ML_Algo

6.8.1 Constructor & Destructor Documentation

6.8.1.1 ML_Algo::ML_Algo ( ) [inline]

6.8.2 Member Function Documentation

6.8.2.1 virtual std::ostream& ML_Algo::_print (std::ostream & *out*) const [inline], [virtual]

This is the function really called to output this object. We involve this as to support polymophism for operator <<.

Reimplemented in SVM_I, and SVM.

6.8.2.2 virtual int ML_Algo::check_question_set (States & *qset*) [pure virtual]

test on question state sets to see if there is an invalidation

The method will output the inforamtion if a question trace invalidate the training model

Parameters

qset	is a reference type to question states.

Returns

int 0 if no error

Implemented in SVM_I, Perceptron, and SVM.

6.8.2.3 virtual int ML_Algo::get_converged (Equation ∗ *previous_equations,* int *equation_num*) [pure virtual]

check whether the training is converged or not

current_training_equations ∼= previous_trainig_equations ???

Parameters

previous_equations	contains all the equation we get from last trainig session.
equation_num	is the number of equations get from last training session

Returns

int 0 if converged

Implemented in SVM_I, and SVM.

6.8.2.4 virtual int ML_Algo::predict (double ∗ *x,* int *flag*) [pure virtual]

Predict sample x against the whole training model.(equations)

Parameters

x	contains the sample to be tested.
flag	leave this to be ZERO...

Returns

The label of prediction

Implemented in SVM_I, SVM, and Perceptron.

6.8.2.5 virtual double ML_Algo::predict_on_training_set ( ) [pure virtual]

Calculate the predict precision of the training-model on the training set.

Returns

double Return precision we can get. Should be a value between 0 and 1.

Implemented in SVM_I, Perceptron, and SVM.

6.8.2.6 virtual int ML_Algo::prepare_training_data (States ∗ *gsets,* int & *pre_positive_size,* int & *pre_negative_size*) [pure virtual]

init training data method. This should be called before any training happens.

Parameters

gsets	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
pre_positive_size	This records the last positive size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.
pre_negative_size	This records the last negative size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.

Returns

int 0 if no error

Implemented in SVM_I, Perceptron, and SVM.

6.8.2.7 virtual Equation∗ ML_Algo::roundoff (int & *equation_num*) [pure virtual]

Round off the whole training model.(equations)

Parameters

equation_num	set by callee to notify the number of equations we currently get

Returns

Eqation Point the rounded off equations. Remember to DELETE them after use by caller. Otherwise memory leak.

Implemented in SVM_I, SVM, and Perceptron.

6.8.2.8 virtual int ML_Algo::size ( ) [pure virtual]

This method returns the current problem size (the number of training states).

Returns

int the size of problem

Implemented in SVM_I, SVM, and Perceptron.

6.8.2.9 virtual int ML_Algo::train ( ) [pure virtual]

The most important TRAIN method, which calls real training algorithm to do training.

Returns

int 0 if no error.

Implemented in SVM_I, Perceptron, and SVM.

6.8.3 Friends And Related Function Documentation

6.8.3.1 std::ostream& operator<< (std::ostream & *out*, const ML_Algo & *mla*) [friend]

output the current trainig result of a ML_Algo

Parameters

mla	the ml_algo object to be output

Returns

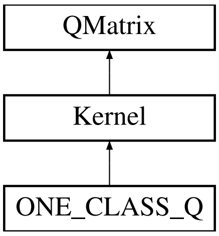
std::ostream

The documentation for this class was generated from the following file:

- include/ml_algo.h

6.9 ONE_CLASS_Q Class Reference

Inheritance diagram for ONE_CLASS_Q:



Public Member Functions

- ONE_CLASS_Q (const svm_problem &prob, const svm_parameter &param)

- Qfloat ∗ get_Q (int i, int len) const
- double ∗ get_QD () const
- void swap_index (int i, int j) const
- ∼ONE_CLASS_Q ()

Additional Inherited Members

6.9.1 Constructor & Destructor Documentation

6.9.1.1 ONE_CLASS_Q::ONE_CLASS_Q (const svm_problem & *prob,* const svm_parameter & *param*) [inline]

6.9.1.2 ONE_CLASS_Q::∼ONE_CLASS_Q ( ) [inline]

6.9.2 Member Function Documentation

6.9.2.1 Qfloat∗ ONE_CLASS_Q::get_Q (int *i,* int *len*) const [inline], [virtual]

Implements Kernel.

6.9.2.2 double∗ ONE_CLASS_Q::get_QD ( ) const [inline], [virtual]

Implements Kernel.

6.9.2.3 void ONE_CLASS_Q::swap_index (int *i,* int *j*) const [inline], [virtual]

Reimplemented from Kernel.

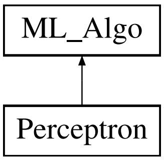
The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.10 Perceptron Class Reference

```
#include <perceptron.h>
```

Inheritance diagram for Perceptron:



Public Member Functions

- Perceptron (void(∗f)(const char ∗)=NULL, int max_size=10000)
- virtual ∼Perceptron ()
- virtual int prepare_training_data (States ∗gsets, int &pre_positive_size, int &pre_negative_size)

  init training data method. This should be called before any training happens.
- virtual int train ()

  The most important TRAIN method, which calls real training algorithm to do training.
- virtual double predict_on_training_set ()

Calculate the predict precision of the training-model on the training set.

- virtual int check_question_set (States &qset)

  test on question state sets to see if there is an invalidation
- virtual int size ()

  This method returns the current problem size (the number of training states).
- virtual Equation ∗ roundoff (int &num)

  Round off the whole training model.(equations)
- virtual int predict (double ∗v, int label=0)

  Predict sample x against the whole training model.(equations)

Data Fields

- Equation ∗ main_equation
- double training_label [max_items ∗2]
- double ∗ training_set [max_items ∗2]
- int length

Friends

- std::ostream & operator<< (std::ostream &out, const Perceptron &)

6.10.1 Constructor & Destructor Documentation

6.10.1.1 Perceptron::Perceptron (void(∗)(const char ∗) *f* = `NULL`, int *max_size* = `10000`)

6.10.1.2 Perceptron::∼Perceptron ( ) `[virtual]`

6.10.2 Member Function Documentation

6.10.2.1 int Perceptron::check_question_set (States & *qset*) `[virtual]`

test on question state sets to see if there is an invalidation

The method will output the inforamtion if a question trace invalidate the training model

Parameters

qset	is a reference type to question states.

Returns

int 0 if no error

Implements ML_Algo.

6.10.2.2 int Perceptron::predict (double ∗ *x*, int *flag* = `0`) `[virtual]`

Predict sample x against the whole training model.(equations)

Parameters

x	contains the sample to be tested.
flag	leave this to be ZERO...

Returns

The label of prediction

Implements ML_Algo.

6.10.2.3 double Perceptron::predict_on_training_set ( ) [virtual]

Calculate the predict precision of the training-model on the training set.

Returns

double Return precision we can get. Should be a value between 0 and 1.

Implements ML_Algo.

6.10.2.4 int Perceptron::prepare_training_data (States ∗ *gsets,* int & *pre_positive_size,* int & *pre_negative_size*) [virtual]

init training data method. This should be called before any training happens.

Parameters

gsets	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
pre_positive_size	This records the last positive size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.
pre_negative_size	This records the last negative size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.

Returns

int 0 if no error

Implements ML_Algo.

6.10.2.5 Equation ∗ Perceptron::roundoff (int & *equation_num*) [virtual]

Round off the whole training model.(equations)

Parameters

equation_num	set by callee to notify the number of equations we currently get

Returns

Eqation Point the rounded off equations. Remember to DELETE them after use by caller. Otherwise memory leak.

Implements ML_Algo.

6.10.2.6 int Perceptron::size ( ) [virtual]

This method returns the current problem size (the number of training states).

Returns

int the size of problem

Implements ML_Algo.

6.10.2.7 int Perceptron::train ( ) [virtual]

The most important TRAIN method, which calls real training algorithm to do training.

Returns

int 0 if no error.

Implements ML_Algo.

6.10.3 Friends And Related Function Documentation

6.10.3.1 std::ostream& operator<< (std::ostream & *out*, const Perceptron & *perceptron*) [friend]

6.10.4 Field Documentation

6.10.4.1 int Perceptron::length

6.10.4.2 Equation* Perceptron::main_equation

6.10.4.3 double Perceptron::training_label[max_items *2]

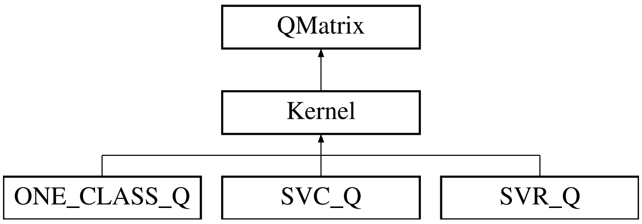
6.10.4.4 double* Perceptron::training_set[max_items *2]

The documentation for this class was generated from the following files:

- include/perceptron.h
- src/perceptron.cpp

6.11 QMatrix Class Reference

Inheritance diagram for QMatrix:



Public Member Functions

- virtual Qfloat ∗ get_Q (int column, int len) const =0
- virtual double ∗ get_QD () const =0
- virtual void swap_index (int i, int j) const =0
- virtual ∼QMatrix ()

6.11.1 Constructor & Destructor Documentation

6.11.1.1 virtual QMatrix::∼QMatrix ( ) `[inline]`, `[virtual]`

6.11.2 Member Function Documentation

6.11.2.1 virtual Qfloat∗ QMatrix::get_Q (int *column*, int *len*) const `[pure virtual]`

Implemented in SVR_Q, ONE_CLASS_Q, SVC_Q, and Kernel.

6.11.2.2 virtual double∗ QMatrix::get_QD ( ) const `[pure virtual]`

Implemented in SVR_Q, ONE_CLASS_Q, SVC_Q, and Kernel.

6.11.2.3 virtual void QMatrix::swap_index (int *i*, int *j*) const `[pure virtual]`

Implemented in SVR_Q, ONE_CLASS_Q, SVC_Q, and Kernel.

The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.12 Solution Class Reference

This class defines the format of a valid solution to an equation.

`#include <equation.h>`

Public Member Functions

- Solution ()

  Default constructor. Set all its elements to value 0.
- Solution (double a0,...)

  Most useful constructor Set its elements to the given values, order keeps.

Data Fields

- double x [VARS]

  The data field of Solution, stores all the values as a solution to an Equation.

Friends

- std::ostream & operator<< (std::ostream &out, const Solution &sol)

  support << operator simply output its elements as a tuple

6.12.1 Detailed Description

This class defines the format of a valid solution to an equation.

It contains values to each variants in an equation

6.12.2 Constructor & Destructor Documentation

6.12.2.1 Solution::Solution ( )

Default constructor. Set all its elements to value 0.

6.12.2.2 Solution::Solution (double *a0*, ...)

Most useful constructor Set its elements to the given values, order keeps.

Parameters

a0...	each element values for a solution

6.12.3 Friends And Related Function Documentation

6.12.3.1 std::ostream& operator<< (std::ostream & *out*, const Solution & *sol*) [friend]

support << operator simply output its elements as a tuple

Parameters

sol	The solution object to be output

6.12.4 Field Documentation

6.12.4.1 double Solution::x[VARS]

The data field of Solution, stores all the values as a solution to an Equation.

The documentation for this class was generated from the following files:

- include/equation.h
- src/equation.cpp

6.13 Solver::SolutionInfo Struct Reference

Data Fields

- double obj
- double rho
- double upper_bound_p
- double upper_bound_n
- double r

6.13.1 Field Documentation

6.13.1.1 double Solver::SolutionInfo::obj

6.13.1.2 double Solver::SolutionInfo::r

6.13.1.3 double Solver::SolutionInfo::rho

6.13.1.4 double Solver::SolutionInfo::upper_bound_n

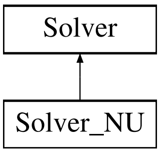
6.13.1.5 double Solver::SolutionInfo::upper_bound_p

The documentation for this struct was generated from the following file:

- src/svm_core.cpp

6.14 Solver Class Reference

Inheritance diagram for Solver:



Data Structures

- struct SolutionInfo

Public Member Functions

- Solver ()
- virtual ∼Solver ()
- void Solve (int l, const QMatrix &Q, const double ∗p_, const schar ∗y_, double ∗alpha_, double Cp, double Cn, double eps, SolutionInfo ∗si, int shrinking)

Protected Types

- enum { LOWER_BOUND, UPPER_BOUND, FREE }

Protected Member Functions

- double get_C (int i)
- void update_alpha_status (int i)
- bool is_upper_bound (int i)
- bool is_lower_bound (int i)
- bool is_free (int i)
- void swap_index (int i, int j)
- void reconstruct_gradient ()
- virtual int select_working_set (int &i, int &j)
- virtual double calculate_rho ()
- virtual void do_shrinking ()

Protected Attributes

- int active_size
- schar ∗ y
- double ∗ G
- char ∗ alpha_status
- double ∗ alpha
- const QMatrix ∗ Q
- const double ∗ QD
- double eps
- double Cp
- double Cn
- double ∗ p
- int ∗ active_set
- double ∗ G_bar
- int l
- bool unshrink

6.14.1 Member Enumeration Documentation

6.14.1.1 anonymous enum [protected]

Enumerator

LOWER_BOUND

UPPER_BOUND

FREE

6.14.2 Constructor & Destructor Documentation

6.14.2.1 Solver::Solver ( ) [inline]

6.14.2.2 virtual Solver::∼Solver ( ) [inline], [virtual]

6.14.3 Member Function Documentation

6.14.3.1 double Solver::calculate_rho ( ) [protected], [virtual]

6.14.3.2 void Solver::do_shrinking ( ) [protected], [virtual]

6.14.3.3 double Solver::get_C (int *i*) [inline], [protected]

6.14.3.4 bool Solver::is_free (int *i*) [inline], [protected]

6.14.3.5 bool Solver::is_lower_bound (int *i*) [inline], [protected]

6.14.3.6 bool Solver::is_upper_bound (int *i*) [inline], [protected]

6.14.3.7 void Solver::reconstruct_gradient ( ) [protected]

6.14.3.8 int Solver::select_working_set (int & *i*, int & *j*) [protected], [virtual]

6.14.3.9 void Solver::Solve (int *l*, const QMatrix & *Q*, const double ∗ *p_*, const schar ∗ *y_*, double ∗ *alpha_*, double *Cp*, double *Cn*, double *eps*, SolutionInfo ∗ *si*, int *shrinking*)

6.14.3.10 void Solver::swap_index (int *i*, int *j*) [protected]

6.14.3.11 void Solver::update_alpha_status (int *i*) [inline], [protected]

6.14.4 Field Documentation

6.14.4.1 int∗ Solver::active_set [protected]

6.14.4.2 int Solver::active_size [protected]

6.14.4.3 double∗ Solver::alpha [protected]

6.14.4.4 char∗ Solver::alpha_status [protected]

6.14.4.5 double Solver::Cn [protected]

6.14.4.6 double Solver::Cp [protected]

6.14.4.7 double Solver::eps [protected]

6.14.4.8 double∗ Solver::G [protected]

6.14.4.9 double∗ Solver::G_bar [protected]

6.14.4.10 int Solver::l [protected]

6.14.4.11 double∗ Solver::p [protected]

6.14.4.12 const QMatrix∗ Solver::Q [protected]

6.14.4.13 const double∗ Solver::QD [protected]

6.14.4.14 bool Solver::unshrink [protected]

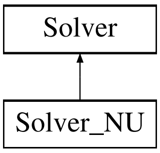
6.14.4.15 schar∗ Solver::y [protected]

The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.15 Solver_NU Class Reference

Inheritance diagram for Solver_NU:



Public Member Functions

- Solver_NU ()

- void Solve (int l, const QMatrix &Q, const double ∗p, const schar ∗y, double ∗alpha, double Cp, double Cn, double eps, SolutionInfo ∗si, int shrinking)

Additional Inherited Members

6.15.1 Constructor & Destructor Documentation

6.15.1.1 Solver_NU::Solver_NU ( ) [inline]

6.15.2 Member Function Documentation

6.15.2.1 void Solver_NU::Solve (int *l*, const QMatrix & *Q*, const double ∗ *p*, const schar ∗ *y*, double ∗ *alpha*, double *Cp*, double *Cn*, double *eps*, SolutionInfo ∗ *si*, int *shrinking*) [inline]

The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.16 States Class Reference

```
#include <states.h>
```

Public Member Functions

- States ()
- ∼States ()
- int add_states (double st[ ][VARS], int len)
- int traces_num ()
- int size ()
- void print_trace (int num)

Data Fields

- double(∗ values)[VARS]
- int ∗ index
- int p_index
- int label

Friends

- std::ostream & operator<< (std::ostream &out, const States &ss)

6.16.1 Constructor & Destructor Documentation

6.16.1.1 States::States ( )

6.16.1.2 States::∼States ( )

6.16.2 Member Function Documentation

6.16.2.1 int States::add_states (double *st[ ][VARS],* int *len*)

6.16.2.2 void States::print_trace (int *num*)

6.16.2.3 int States::size ( )

6.16.2.4 int States::traces_num ( )

6.16.3 Friends And Related Function Documentation

6.16.3.1 std::ostream& operator<< (std::ostream & *out,* const States & *ss*) [friend]

6.16.4 Field Documentation

6.16.4.1 int∗ States::index

6.16.4.2 int States::label

6.16.4.3 int States::p_index

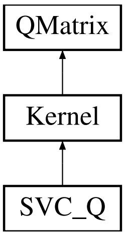
6.16.4.4 double(∗ States::values)[VARS]

The documentation for this class was generated from the following files:

- include/states.h
- src/states.cpp

6.17 SVC_Q Class Reference

Inheritance diagram for SVC_Q:



Public Member Functions

- SVC_Q (const svm_problem &prob, const svm_parameter &param, const schar ∗y_)
- Qfloat ∗ get_Q (int i, int len) const
- double ∗ get_QD () const
- void swap_index (int i, int j) const
- ∼SVC_Q ()

Additional Inherited Members

6.17.1 Constructor & Destructor Documentation

6.17.1.1 SVC_Q::SVC_Q (const svm_problem & *prob*, const svm_parameter & *param*, const schar ∗ *y_*) `[inline]`

6.17.1.2 SVC_Q::∼SVC_Q ( ) `[inline]`

6.17.2 Member Function Documentation

6.17.2.1 Qfloat∗ SVC_Q::get_Q (int *i*, int *len*) const `[inline]`, `[virtual]`

Implements Kernel.

6.17.2.2 double∗ SVC_Q::get_QD ( ) const `[inline]`, `[virtual]`

Implements Kernel.

6.17.2.3 void SVC_Q::swap_index (int *i*, int *j*) const `[inline]`, `[virtual]`

Reimplemented from Kernel.

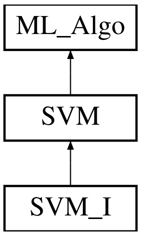
The documentation for this class was generated from the following file:

- src/svm_core.cpp

6.18 SVM Class Reference

`#include <svm.h>`

Inheritance diagram for SVM:



Public Member Functions

- SVM (void(∗f)(const char ∗)=NULL, int size=10000)
- virtual ∼SVM ()
- virtual int prepare_training_data (States ∗gsets, int &pre_positive_size, int &pre_negative_size)

  init training data method. This should be called before any training happens.
- virtual int train ()

  The most important TRAIN method, which calls real training algorithm to do training.
- virtual double predict_on_training_set ()

  Calculate the predict precision of the training-model on the training set.
- virtual int check_question_set (States &qset)

  test on question state sets to see if there is an invalidation
- virtual int get_converged (Equation ∗, int)

  check whether the training is converged or not
- virtual std::ostream & _print (std::ostream &out) const

This is the function really called to output this object. We involve this as to support polymophism for operator <<.

- virtual int size ()

  This method returns the current problem size (the number of training states).
- virtual Equation ∗ roundoff (int &num)

  Round off the whole training model.(equations)
- virtual int predict (double ∗v, int label=0)

  Predict sample x against the whole training model.(equations)

Data Fields

- svm_model ∗ model
- Equation ∗ main_equation
- svm_parameter param
- svm_problem problem
- double ∗ training_label
- double ∗∗ training_set

Protected Attributes

- int max_size

Friends

- std::ostream & operator<< (std::ostream &out, const SVM &svm)

6.18.1 Constructor & Destructor Documentation

6.18.1.1 SVM::SVM (void(∗)(const char ∗) *f* = NULL, int *size* = 10000)

6.18.1.2 SVM::∼SVM ( ) [virtual]

6.18.2 Member Function Documentation

6.18.2.1 std::ostream & SVM::_print (std::ostream & *out*) const [virtual]

This is the function really called to output this object. We involve this as to support polymophism for operator <<.

Reimplemented from ML_Algo.

Reimplemented in SVM_I.

6.18.2.2 int SVM::check_question_set (States & *qset*) [virtual]

test on question state sets to see if there is an invalidation

The method will output the inforamtion if a question trace invalidate the training model

Parameters

qset	is a reference type to question states.

Returns

int 0 if no error

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.3 int SVM::get_converged (Equation ∗ *previous_equations,* int *equation_num*) [virtual]

check whether the training is converged or not

current_training_equations ∼= previous_trainig_equations ???

Parameters

previous_equations	contains all the equation we get from last trainig session.
equation_num	is the number of equations get from last training session

Returns

int 0 if converged

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.4 int SVM::predict (double ∗ *x,* int *flag* = 0) [virtual]

Predict sample x against the whole training model.(equations)

Parameters

x	contains the sample to be tested.
flag	leave this to be ZERO...

Returns

The label of prediction

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.5 double SVM::predict_on_training_set ( ) [virtual]

Calculate the predict precision of the training-model on the training set.

Returns

double Return precision we can get. Should be a value between 0 and 1.

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.6 int SVM::prepare_training_data (States ∗ *gsets,* int & *pre_positive_size,* int & *pre_negative_size*) [virtual]

init training data method. This should be called before any training happens.

Parameters

gsets	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
pre_positive_size	This records the last positive size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.
pre_negative_size	This records the last negative size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.

Returns

int 0 if no error

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.7 Equation ∗ SVM::roundoff (int & *equation_num*) [virtual]

Round off the whole training model.(equations)

Parameters

equation_num	set by callee to notify the number of equations we currently get

Returns

Eqation Point the rounded off equations. Remember to DELETE them after use by caller. Otherwise memory leak.

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.8 int SVM::size ( ) [virtual]

This method returns the current problem size (the number of training states).

Returns

int the size of problem

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.2.9 int SVM::train ( ) [virtual]

The most important TRAIN method, which calls real training algorithm to do training.

Returns

int 0 if no error.

Implements ML_Algo.

Reimplemented in SVM_I.

6.18.3 Friends And Related Function Documentation

6.18.3.1 std::ostream& operator<< (std::ostream & *out*, const SVM & *svm*) [friend]

6.18.4 Field Documentation

6.18.4.1 Equation∗ SVM::main_equation

6.18.4.2 int SVM::max_size [protected]

6.18.4.3 svm_model∗ SVM::model

6.18.4.4 svm_parameter SVM::param

6.18.4.5 svm_problem SVM::problem

6.18.4.6 double∗ SVM::training_label

6.18.4.7 double∗∗ SVM::training_set

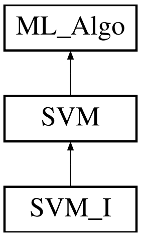
The documentation for this class was generated from the following files:

- include/svm.h
- src/svm.cpp

6.19 SVM_I Class Reference

```
#include <svm_i.h>
```

Inheritance diagram for SVM_I:



Public Member Functions

- SVM_I (void(∗f)(const char ∗)=NULL, int size=10000, int equ=16)
- ∼SVM_I ()
- virtual int prepare_training_data (States ∗gsets, int &pre_positive_size, int &pre_negative_size)

  init training data method. This should be called before any training happens.
- int train ()

The most important TRAIN method, which calls real training algorithm to do training.
- double predict_on_training_set ()

  Calculate the predict precision of the training-model on the training set.
- virtual int check_question_set (States &qset)

  test on question state sets to see if there is an invalidation
- virtual int get_converged (Equation ∗, int)

  check whether the training is converged or not
- virtual std::ostream & _print (std::ostream &out) const

  This is the function really called to output this object. We involve this as to support polymophism for operator <<.
- int size ()

  This method returns the current problem size (the number of training states).
- virtual Equation ∗ roundoff (int &num)

  Round off the whole training model.(equations)
- virtual int predict (double ∗v, int label=0)

  Predict sample x against the whole training model.(equations)

Data Fields

- svm_model ∗ model
- Equation ∗ equations
- int equ_num
- svm_parameter param
- States ∗ negatives

Protected Attributes

- int max_equ

Friends

- std::ostream & operator<< (std::ostream &out, const SVM_I &svm_i)

6.19.1 Constructor & Destructor Documentation

6.19.1.1 SVM_I::SVM_I (void(∗)(const char ∗) *f* = `NULL`, int *size* = `10000`, int *equ* = `16`)

6.19.1.2 SVM_I::∼SVM_I ( )

6.19.2 Member Function Documentation

6.19.2.1 std::ostream & SVM_I::_print (std::ostream & *out*) const [virtual]

This is the function really called to output this object. We involve this as to support polymophism for operator <<.

Reimplemented from SVM.

6.19.2.2 int SVM_I::check_question_set (States & *qset*) [virtual]

test on question state sets to see if there is an invalidation

The method will output the inforamtion if a question trace invalidate the training model

Parameters

qset	is a reference type to question states.

Returns

int 0 if no error

Reimplemented from SVM.

6.19.2.3 int SVM_I::get_converged (Equation ∗ *previous_equations,* int *equation_num*) [virtual]

check whether the training is converged or not

current_training_equations ∼= previous_trainig_equations ???

Parameters

previous_equations	contains all the equation we get from last trainig session.
equation_num	is the number of equations get from last training session

Returns

int 0 if converged

Reimplemented from SVM.

6.19.2.4 int SVM_I::predict (double ∗ *x,* int *flag* = 0) [virtual]

Predict sample x against the whole training model.(equations)

Parameters

x	contains the sample to be tested.
flag	leave this to be ZERO...

Returns

The label of prediction

Reimplemented from SVM.

6.19.2.5 double SVM_I::predict_on_training_set ( ) [virtual]

Calculate the predict precision of the training-model on the training set.

Returns

double Return precision we can get. Should be a value between 0 and 1.

Reimplemented from SVM.

6.19.2.6 int SVM_I::prepare_training_data (States * *gsets*, int & *pre_positive_size*, int & *pre_negative_size*) [virtual]

init training data method. This should be called before any training happens.

Parameters

gsets	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
pre_positive_size	This records the last positive size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.
pre_negative_size	This records the last negative size of states. And also set by callee to the new value Initially set to 0, as there is no elements in positive states. Calls afterwards should pass the value set by last call.

Returns

int 0 if no error

Reimplemented from SVM.

6.19.2.7 Equation * SVM_I::roundoff (int & *equation_num*) [virtual]

Round off the whole training model.(equations)

Parameters

equation_num	set by callee to notify the number of equations we currently get

Returns

Eqation Point the rounded off equations. Remember to DELETE them after use by caller. Otherwise memory leak.

Reimplemented from SVM.

6.19.2.8 int SVM_I::size ( ) [virtual]

This method returns the current problem size (the number of training states).

Returns

int the size of problem

Reimplemented from SVM.

6.19.2.9 int SVM_I::train ( ) [virtual]

The most important TRAIN method, which calls real training algorithm to do training.

Returns

int 0 if no error.

Reimplemented from SVM.

6.19.3 Friends And Related Function Documentation

6.19.3.1 std::ostream& operator<< (std::ostream & *out*, const SVM_I & *svm_i*) [friend]

6.19.4 Field Documentation

6.19.4.1 int SVM_I::equ_num

6.19.4.2 Equation* SVM_I::equations

6.19.4.3 int SVM_I::max_equ [protected]

6.19.4.4 svm_model* SVM_I::model

6.19.4.5 States* SVM_I::negatives

6.19.4.6 svm_parameter SVM_I::param

The documentation for this class was generated from the following files:

- include/svm_i.h
- src/svm_i.cpp

6.20 svm_model Struct Reference

```
#include <svm_core.h>
```

Data Fields

- struct svm_parameter param
- int nr_class
- int l
- struct svm_node ** SV
- double ** sv_coef
- double * rho
- double * probA
- double * probB
- int * sv_indices
- int * label
- int * nSV
- int free_sv

6.20.1 Field Documentation

6.20.1.1 int svm_model::free_sv

6.20.1.2 int svm_model::l

6.20.1.3 int* svm_model::label

6.20.1.4 int svm_model::nr_class

6.20.1.5 int∗ svm_model::nSV

6.20.1.6 struct svm_parameter svm_model::param

6.20.1.7 double∗ svm_model::probA

6.20.1.8 double∗ svm_model::probB

6.20.1.9 double∗ svm_model::rho

6.20.1.10 struct svm_node∗∗ svm_model::SV

6.20.1.11 double∗∗ svm_model::sv_coef

6.20.1.12 int∗ svm_model::sv_indices

The documentation for this struct was generated from the following file:

- include/svm_core.h

6.21 svm_node Struct Reference

```
#include <svm_core.h>
```

Data Fields

- double value

Friends

- std::ostream & operator<< (std::ostream &out, const svm_node &sn)

6.21.1 Friends And Related Function Documentation

6.21.1.1 std::ostream& operator<< (std::ostream & *out*, const svm_node & *sn*) [friend]

6.21.2 Field Documentation

6.21.2.1 double svm_node::value

The documentation for this struct was generated from the following file:

- include/svm_core.h

6.22 svm_parameter Struct Reference

```
#include <svm_core.h>
```

Data Fields

- int svm_type
- int kernel_type
- int degree
- double gamma
- double coef0
- double cache_size
- double eps
- double C
- int nr_weight
- int ∗ weight_label
- double ∗ weight
- double nu
- double p
- int shrinking
- int probability

6.22.1 Field Documentation

6.22.1.1 double svm_parameter::C

6.22.1.2 double svm_parameter::cache_size

6.22.1.3 double svm_parameter::coef0

6.22.1.4 int svm_parameter::degree

6.22.1.5 double svm_parameter::eps

6.22.1.6 double svm_parameter::gamma

6.22.1.7 int svm_parameter::kernel_type

6.22.1.8 int svm_parameter::nr_weight

6.22.1.9 double svm_parameter::nu

6.22.1.10 double svm_parameter::p

6.22.1.11 int svm_parameter::probability

6.22.1.12 int svm_parameter::shrinking

6.22.1.13 int svm_parameter::svm_type

6.22.1.14 double∗ svm_parameter::weight

6.22.1.15 int∗ svm_parameter::weight_label

The documentation for this struct was generated from the following file:

- include/svm_core.h

6.23 svm_problem Struct Reference

```
#include <svm_core.h>
```

Data Fields

- int l
- double ∗ y
- struct svm_node ∗∗ x

Friends

- std::ostream & operator<< (std::ostream &out, const svm_problem &sp)

6.23.1 Friends And Related Function Documentation

6.23.1.1 std::ostream& operator<< (std::ostream & *out*, const svm_problem & *sp*) [friend]

6.23.2 Field Documentation

6.23.2.1 int svm_problem::l

6.23.2.2 struct svm_node∗∗ svm_problem::x

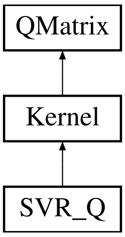
6.23.2.3 double∗ svm_problem::y

The documentation for this struct was generated from the following file:

- include/svm_core.h

6.24 SVR_Q Class Reference

Inheritance diagram for SVR_Q:



Public Member Functions

- SVR_Q (const svm_problem &prob, const svm_parameter &param)
- void swap_index (int i, int j) const
- Qfloat ∗ get_Q (int i, int len) const
- double ∗ get_QD () const
- ∼SVR_Q ()

Additional Inherited Members

6.24.1 Constructor & Destructor Documentation

6.24.1.1 SVR_Q::SVR_Q (const svm_problem & *prob*, const svm_parameter & *param*) [inline]

6.24.1.2 SVR_Q::∼SVR_Q ( ) [inline]

6.24.2 Member Function Documentation

6.24.2.1 Qfloat∗ SVR_Q::get_Q (int *i*, int *len*) const [inline], [virtual]

Implements Kernel.

6.24.2.2 double∗ SVR_Q::get_QD ( ) const [inline], [virtual]

Implements Kernel.

6.24.2.3 void SVR_Q::swap_index (int *i*, int *j*) const [inline], [virtual]

Reimplemented from Kernel.

The documentation for this class was generated from the following file:

- src/svm_core.cpp

Chapter 7

File Documentation

7.1 build/CMakeCache.txt File Reference

7.2 build/CMakeFiles/2.8.12.2/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define COMPILER_ID ""
- #define PLATFORM_ID ""
- #define ARCHITECTURE_ID ""
- #define DEC(n)
- #define HEX(n)

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"

7.2.1 Macro Definition Documentation

7.2.1.1 #define ARCHITECTURE_ID ""

7.2.1.2 #define COMPILER_ID ""

7.2.1.3 #define DEC(*n*)

Value:

```
(’0’ + (((n) / 10000000)%10)), \
  (’0’ + (((n) / 1000000)%10)),  \
  (’0’ + (((n) / 100000)%10)),   \
  (’0’ + (((n) / 10000)%10)),    \
  (’0’ + (((n) / 1000)%10)),     \
  (’0’ + (((n) / 100)%10)),      \
  (’0’ + (((n) / 10)%10)),       \
  (’0’ +  ((n) % 10))
```

7.2.1.4 #define HEX(*n*)

Value:

```
('0' + ((n)>>28 & 0xF)), \
  ('0' + ((n)>>24 & 0xF)), \
  ('0' + ((n)>>20 & 0xF)), \
  ('0' + ((n)>>16 & 0xF)), \
  ('0' + ((n)>>12 & 0xF)), \
  ('0' + ((n)>>8  & 0xF)), \
  ('0' + ((n)>>4  & 0xF)), \
  ('0' + ((n)     & 0xF))
```

7.2.1.5 #define PLATFORM_ID ""

7.2.2 Function Documentation

7.2.2.1 int main (int *argc,* char ∗ *argv[ ]*)

7.2.3 Variable Documentation

7.2.3.1 char const∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"

7.2.3.2 char const∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"

7.2.3.3 char const∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"

7.3 build/CMakeFiles/2.8.12.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define COMPILER_ID ""
- #define PLATFORM_ID ""
- #define ARCHITECTURE_ID ""
- #define DEC(n)
- #define HEX(n)

Functions

- int main (int argc, char ∗argv[ ])

Variables

- char const ∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
- char const ∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
- char const ∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"

7.3.1 Macro Definition Documentation

7.3.1.1 #define ARCHITECTURE_ID ""

7.3.1.2 #define COMPILER_ID ""

7.3.1.3 #define DEC(*n*)

Value:

```
(’0’ + (((n) / 10000000)%10)), \
  (’0’ + (((n) / 1000000)%10)),  \
  (’0’ + (((n) / 100000)%10)),   \
  (’0’ + (((n) / 10000)%10)),    \
  (’0’ + (((n) / 1000)%10)),     \
  (’0’ + (((n) / 100)%10)),      \
  (’0’ + (((n) / 10)%10)),       \
  (’0’ +  ((n) % 10))
```

7.3.1.4 #define HEX(*n*)

Value:

```
(’0’ + ((n)>>28 & 0xF)), \
  (’0’ + ((n)>>24 & 0xF)), \
  (’0’ + ((n)>>20 & 0xF)), \
  (’0’ + ((n)>>16 & 0xF)), \
  (’0’ + ((n)>>12 & 0xF)), \
  (’0’ + ((n)>>8  & 0xF)), \
  (’0’ + ((n)>>4  & 0xF)), \
  (’0’ + ((n)     & 0xF))
```

7.3.1.5 #define PLATFORM_ID ""

7.3.2 Function Documentation

7.3.2.1 int main (int *argc,* char ∗ *argv[ ]*)

7.3.3 Variable Documentation

7.3.3.1 char const∗ info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"

7.3.3.2 char const∗ info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"

7.3.3.3 char const∗ info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"

7.4 build/CMakeFiles/conj.dir/link.txt File Reference

7.5 build/CMakeFiles/ex1.dir/link.txt File Reference

7.6 build/CMakeFiles/f1.dir/link.txt File Reference

7.7 build/CMakeFiles/f2.dir/link.txt File Reference

7.8 build/CMakeFiles/f3.dir/link.txt File Reference

7.9 build/CMakeFiles/z3test.dir/link.txt File Reference

7.10 build/CMakeFiles/TargetDirectories.txt File Reference

7.11 CMakeLists.txt File Reference

7.12 include/color.h File Reference

Provide support for colorful console ouput.

```
#include <iostream>
```

Enumerations

- enum color {
  RED = 0, YELLOW, GREEN, BLUE,
  WHITE }

  This enumeration contains all the colors predifined in project. Here we only introduce RED, YELLOW, GREEN, BLUE, WHITE which is enough for our output. You can import more color if you want.

Functions

- void set_console_color (std::ostream &out, int color=YELLOW)

  This function sets the given stream to the given color, YELLOW is default.
- void unset_console_color (std::ostream &out)

  This function sets the console color back to origin setting, not the previous setting. By origin, we mean black background, white foreground, no strong comparision.

7.12.1 Detailed Description

Provide support for colorful console ouput.

This file contains the necessary function support for colorful console text output. The usage is also simple. Before you output something, call function set_console_color. And remember to call unset_console_color after your output.

Author

Li Jiaying

Bug unset_console_color is set the console back to black background, white forground, no strong comparision instead of the previous setting.

7.12.2 Enumeration Type Documentation

7.12.2.1 enum color

This enumeration contains all the colors predifined in project. Here we only introduce RED, YELLOW, GREEN, BLUE, WHITE which is enough for our output. You can import more color if you want.

Enumerator

RED

YELLOW

GREEN

BLUE

WHITE

7.12.3 Function Documentation

7.12.3.1 void set_console_color (std::ostream & *out,* int *color* = YELLOW)

This function sets the given stream to the given color, YELLOW is default.

Parameters

out	The ostream to be changed, defines which stream you want to set
color	The Color to set. YELLOW by default.

7.12.3.2 void unset_console_color (std::ostream & *out*)

This function sets the console color back to origin setting, not the previous setting. By origin, we mean black background, white foreground, no strong comparision.

7.13 include/config.h File Reference

Provide most configuration information for the whole project.

Macros

- #define VARS 2

  defines the number of paramenters by a given loop,\ This also is the number of parameters we need to record for processing. This should be set in /CMakeLists.txt file If it is not set correctly, you may come across a runtime error
- #define PRECISION 3

  is a integer, which defines precision as pow(10, -PRECISION) This should be set in /CMakeLists.txt file You'd better set this value in a scope [1, 12]

Functions

- bool register_program (int(∗func)(int ∗), const char ∗func_name=0)

  This function register the test program to the framework.

Variables

- int(∗ target_program)(int ∗)

  The pointer to test program, DO NOT assign directly Call register_program to set its value.
- const int max_items = 100000

  defines the initial max number items contains by states set. Better to be a number larger than 1000
- const int init_exes = 6 ∗ VARS

  defines the number of tests runs initially. Should be a positive integer.
- const int after_exes = 4 ∗ VARS

  defines the number of tests runs after the first time. Should be a positive integer.
- const int random_exes = 2

  defines the number of random tests runs each time, which is used to avoid bias caused by tests picking chioce. Should be a non-negative integer.
- const int max_iter = 32

  defines the max number of iterations tried by machine learning algorithm, Should be a positive integer. Usually set between 8-128

7.13.1 Detailed Description

Provide most configuration information for the whole project.

This file contains most of the setting which are used to customize the project.

Author

Li Jiaying

Bug No known bugs.

7.13.2 Macro Definition Documentation

7.13.2.1 #define PRECISION 3

is a integer, which defines precision as pow(10, -PRECISION) This should be set in /CMakeLists.txt file You'd better set this value in a scope [1, 12]

7.13.2.2 #define VARS 2

defines the number of paramenters by a given loop,\ This also is the number of parameters we need to record for processing. This should be set in /CMakeLists.txt file If it is not set correctly, you may come across a runtime error

7.13.3 Function Documentation

7.13.3.1 bool register_program (int(∗)(int ∗) *func,* const char ∗ *func_name* = 0)

This function register the test program to the framework.

Parameters

func	The function to be tested It involves a small validation test on the given function.
fun_name	defines the function name, can be ignored, or set to NULL

Returns

a boolean value false only when the given function is not valid.

7.13.4 Variable Documentation

7.13.4.1 const int after_exes = 4 ∗ VARS

defines the number of tests runs after the first time. Should be a positive integer.

7.13.4.2 const int init_exes = 6 ∗ VARS

defines the number of tests runs initially. Should be a positive integer.

7.13.4.3 const int max_items = 100000

defines the initial max number items contains by states set. Better to be a number larger than 1000

7.13.4.4 const int max_iter = 32

defines the max number of iterations tried by machine learning algorithm, Should be a positive integer. Usually set between 8-128

7.13.4.5 const int random_exes = 2

defines the number of random tests runs each time, which is used to avoid bias caused by tests picking chioce. Should be a non-negative integer.

7.13.4.6 int(∗ target_program) (int ∗)

The pointer to test program, DO NOT assign directly Call register_program to set its value.

7.14 include/equation.h File Reference

Defines the linear equation format and its solution format.

```
#include "config.h"
#include <cmath>
#include <cfloat>
#include <stdarg.h>
#include <cstdlib>
#include <iostream>
#include <iomanip>
```

Data Structures

- class Solution

  This class defines the format of a valid solution to an equation.

- class Equation

  This class defines an equation by storing all its coefficiencies. An equation is regarded a hyperplane in math.

Variables

- int maxv
- int minv

7.14.1 Detailed Description

Defines the linear equation format and its solution format.

Author

Li Jiaying

Bug No known bugs.

7.14.2 Variable Documentation

7.14.2.1 int maxv

7.14.2.2 int minv

7.15 include/iif.h File Reference

Contains all the files that needed to be included by a new test.

```
#include "config.h"
#include "instrumentation.h"
#include "ml_algo.h"
#include "svm.h"
#include "svm_i.h"
#include "color.h"
#include "equation.h"
#include "states.h"
#include "iif_learn.h"
#include "iif_svm_learn.h"
#include "iif_svm_i_learn.h"
#include "iif_assert.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
#include <cstdlib>
```

Variables

- int minv
- int maxv

7.15.1 Detailed Description

Contains all the files that needed to be included by a new test.

By include this file, it should resolve all the reference errors to the framework.

Author

Li Jiaying

Bug No found bugs

7.15.2 Variable Documentation

7.15.2.1 int maxv

7.15.2.2 int minv

7.16 include/iif_assert.h File Reference

Provide iif_assert and iif_assume support for system assume and assert macros.

Macros

- #define iif_assume(expr)

  Used to envelope loop precondition.
- #define iif_assert(expr)

  Used to envelope loop precondition.

Variables

- bool _passP

  a flag to justify whether the given input has pass loop precondition
- bool _passQ

  a flag to justify whether the given input has pass loop postcondition
- int assume_times

  integers values contain the call times to iif_assume and iif_assert, used to validate a given test
- int assert_times

7.16.1 Detailed Description

Provide iif_assert and iif_assume support for system assume and assert macros.

For each valid test, iif_assume and iif_assert should be called only once. Otherwise, the test is regarded as an invalid test.

Author

Li Jiaying

Bug unset_console_color is set the console back to black background, white forground, no strong comparision instead of the previous setting.

7.16.2 Macro Definition Documentation

7.16.2.1 #define iif_assert(*expr*)

Value:

```
do { \
    _passQ = (expr)? true : false;\
    assert_times++;\
} while(0)
```

Used to envelope loop precondition.

Do not support multiple calls

Parameters

expr	loop postcondition

7.16.2.2 #define iif_assume(*expr*)

Value:

```
do { \
    _passP = (expr)? true : false;\
    assume_times++;\
} while(0)
```

Used to envelope loop precondition.

Do not support multiple calls

Parameters

expr	loop precondition

7.16.3 Variable Documentation

7.16.3.1 bool _passP

a flag to justify whether the given input has pass loop precondition

7.16.3.2 bool _passQ

a flag to justify whether the given input has pass loop postcondition

7.16.3.3 int assert_times

7.16.3.4 int assume_times

integers values contain the call times to iif_assume and iif_assert, used to validate a given test

7.17 include/iif_learn.h File Reference

```
#include "config.h"
#include "states.h"
#include "equation.h"
#include "instrumentation.h"
#include "color.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
```

Data Structures

- class IIF_learn

7.18 include/iif_svm_i_learn.h File Reference

```
#include "config.h"
#include "iif_learn.h"
#include "ml_algo.h"
#include "svm_i.h"
#include "color.h"
#include "equation.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
```

Data Structures

- class IIF_svm_i_learn

7.19 include/iif_svm_learn.h File Reference

```
#include "config.h"
#include "iif_learn.h"
#include "ml_algo.h"
#include "svm.h"
#include "color.h"
#include "equation.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
```

Data Structures

- class IIF_svm_learn

7.20 include/instrumentation.h File Reference

Provide instrumentation function support for the framework.

```
#include "config.h"
#include "states.h"
#include <stdarg.h>
```

Enumerations

- enum trace_type { NEGATIVE = -1, QUESTION, POSITIVE, COUNTER_EXAMPLE }

  Contains all the FOUR trace typies here. Negative, Quesion, Positive and Counter_example.

Functions

- int add_state_int (int first,...)
- int add_state_double (double first,...)
- int m_int (int ∗)

  record furntions for each platform
- int m_double (double ∗)

  function jump list, DONOT use it unless you know what you are doing
- int before_loop ()

  This function should be called each time before executing loop.
- int after_loop (States ∗)

  This function should be called each time after executing loop.

7.20.1 Detailed Description

Provide instrumentation function support for the framework.

Author

Li Jiaying

Bug No known bugs

7.20.2 Enumeration Type Documentation

7.20.2.1 enum trace_type

Contains all the FOUR trace typies here. Negative, Quesion, Positive and Counter_example.

Negative = -1 because we want to compatible with natural meaning and svm labels. This also cause a problem to reassign states point in each test file which is ugly

Enumerator

NEGATIVE

QUESTION

POSITIVE

COUNTER_EXAMPLE

7.20.3 Function Documentation

7.20.3.1 int add_state_double (double *first,* ...)

7.20.3.2 int add_state_int (int *first,* ...)

7.20.3.3 int after_loop (States ∗)

This function should be called each time after executing loop.

Parameters

void	

Returns

void

7.20.3.4 int before_loop ( )

This function should be called each time before executing loop.

Parameters

void	

Returns

void

7.20.3.5 int m_double (double ∗)

function jump list, DONOT use it unless you know what you are doing

7.20.3.6 int m_int (int ∗)

record furntions for each platform

function jump list, DONOT use it unless you know what you are doing

7.21 include/ml_algo.h File Reference

Provide the base class for specific maching leanring algorithm.

```
#include <iostream>
#include "states.h"
#include "equation.h"
```

Data Structures

- class ML_Algo

7.21.1 Detailed Description

Provide the base class for specific maching leanring algorithm.

This file contains all the necessary function support for specific machine learning algorithm.

Author

Li Jiaying

Bug

7.22 include/perceptron.h File Reference

```
#include "config.h"
#include "instrumentation.h"
#include "color.h"
#include "ml_algo.h"
```

Data Structures

- class Perceptron

7.23 include/states.h File Reference

```
#include "config.h"
#include <iostream>
```

Data Structures

- class States

7.24 include/svm.h File Reference

```
#include "ml_algo.h"
#include "svm_core.h"
```

Data Structures

- class SVM

7.25 include/svm_core.h File Reference

```
#include "config.h"
#include "instrumentation.h"
#include "color.h"
#include <iostream>
```

Data Structures

- struct svm_node
- struct svm_problem
- struct svm_parameter
- struct svm_model

Macros

- #define LIBSVM_VERSION 320

Enumerations

- enum {
  C_SVC, NU_SVC, ONE_CLASS, EPSILON_SVR,
  NU_SVR }
- enum {
  LINEAR, POLY, RBF, SIGMOID,
  PRECOMPUTED }

Functions

- struct svm_model ∗ svm_train (const struct svm_problem ∗prob, const struct svm_parameter ∗param)
- void svm_cross_validation (const struct svm_problem ∗prob, const struct svm_parameter ∗param, int nr_fold, double ∗target)
- int svm_save_model (const char ∗model_file_name, const struct svm_model ∗model)
- struct svm_model ∗ svm_load_model (const char ∗model_file_name)
- int svm_get_svm_type (const struct svm_model ∗model)
- int svm_get_nr_class (const struct svm_model ∗model)
- void svm_get_labels (const struct svm_model ∗model, int ∗label)
- void svm_get_sv_indices (const struct svm_model ∗model, int ∗sv_indices)
- int svm_get_nr_sv (const struct svm_model ∗model)
- double svm_get_svr_probability (const struct svm_model ∗model)
- double svm_predict_values (const struct svm_model ∗model, const struct svm_node ∗x, double ∗dec_values)
- double svm_predict (const struct svm_model ∗model, const struct svm_node ∗x)
- double svm_predict_probability (const struct svm_model ∗model, const struct svm_node ∗x, double ∗prob↩_estimates)
- void svm_free_model_content (struct svm_model ∗model_ptr)
- void svm_free_and_destroy_model (struct svm_model ∗∗model_ptr_ptr)
- void svm_destroy_param (struct svm_parameter ∗param)
- const char ∗ svm_check_parameter (const struct svm_problem ∗prob, const struct svm_parameter ∗param)
- int svm_check_probability_model (const struct svm_model ∗model)
- void svm_set_print_string_function (void(∗print_func)(const char ∗))
- int svm_model_visualization (const svm_model ∗model, Equation ∗equ)
- void print_svm_samples (const svm_problem ∗sp)
- struct svm_model ∗ svm_l_train (const struct svm_problem ∗prob, const struct svm_parameter ∗param)

Variables

- int libsvm_version

7.25.1 Macro Definition Documentation

7.25.1.1 #define LIBSVM_VERSION 320

7.25.2 Enumeration Type Documentation

7.25.2.1 anonymous enum

Enumerator

C_SVC

- ***NU_SVC***
- ***ONE_CLASS***
- ***EPSILON_SVR***
- ***NU_SVR***

7.25.2.2 anonymous enum

Enumerator

- ***LINEAR***
- ***POLY***
- ***RBF***
- ***SIGMOID***
- ***PRECOMPUTED***

7.25.3 Function Documentation

7.25.3.1 void print_svm_samples (const svm_problem ∗ *sp*)

7.25.3.2 const char∗ svm_check_parameter (const struct svm_problem ∗ *prob,* const struct svm_parameter ∗ *param*)

7.25.3.3 int svm_check_probability_model (const struct svm_model ∗ *model*)

7.25.3.4 void svm_cross_validation (const struct svm_problem ∗ *prob,* const struct svm_parameter ∗ *param,* int *nr_fold,* double ∗ *target*)

7.25.3.5 void svm_destroy_param (struct svm_parameter ∗ *param*)

7.25.3.6 void svm_free_and_destroy_model (struct svm_model ∗∗ *model_ptr_ptr*)

7.25.3.7 void svm_free_model_content (struct svm_model ∗ *model_ptr*)

7.25.3.8 void svm_get_labels (const struct svm_model ∗ *model,* int ∗ *label*)

7.25.3.9 int svm_get_nr_class (const struct svm_model ∗ *model*)

7.25.3.10 int svm_get_nr_sv (const struct svm_model ∗ *model*)

7.25.3.11 void svm_get_sv_indices (const struct svm_model ∗ *model,* int ∗ *sv_indices*)

7.25.3.12 int svm_get_svm_type (const struct svm_model ∗ *model*)

7.25.3.13 double svm_get_svr_probability (const struct svm_model ∗ *model*)

7.25.3.14 struct svm_model∗ svm_l_train (const struct svm_problem ∗ *prob,* const struct svm_parameter ∗ *param*)

7.25.3.15 struct svm_model∗ svm_load_model (const char ∗ *model_file_name*)

7.25.3.16 int svm_model_visualization (const svm_model ∗ *model,* Equation ∗ *equ*)

7.25.3.17 double svm_predict (const struct svm_model ∗ *model,* const struct svm_node ∗ *x*)

7.25.3.18 double svm_predict_probability (const struct svm_model ∗ *model,* const struct svm_node ∗ *x,* double ∗ *prob_estimates*)

7.25.3.19 double svm_predict_values (const struct **svm_model** ∗ *model,* const struct **svm_node** ∗ *x,* double ∗ *dec_values*)

7.25.3.20 int svm_save_model (const char ∗ *model_file_name,* const struct **svm_model** ∗ *model*)

7.25.3.21 void svm_set_print_string_function (void(∗)(const char ∗) *print_func*)

7.25.3.22 struct **svm_model**∗ svm_train (const struct **svm_problem** ∗ *prob,* const struct **svm_parameter** ∗ *param*)

7.25.4 Variable Documentation

7.25.4.1 int libsvm_version

7.26 include/svm_i.h File Reference

```
#include "svm.h"
#include "color.h"
#include <iostream>
```

Data Structures

- class SVM_I

7.27 README.md File Reference

7.28 src/color.cpp File Reference

```
#include "color.h"
```

Functions

- void unset_console_color (std::ostream &out)

  This function sets the console color back to origin setting, not the previous setting. By origin, we mean black background, white foreground, no strong comparision.

7.28.1 Function Documentation

7.28.1.1 void unset_console_color (std::ostream & *out*)

This function sets the console color back to origin setting, not the previous setting. By origin, we mean black background, white foreground, no strong comparision.

7.29 src/config.cpp File Reference

```
#include "config.h"
#include "iif.h"
#include "instrumentation.h"
#include <iostream>
```

Functions

- bool check_target_program (int(∗func)(int ∗))
- bool register_program (int(∗func)(int ∗), const char ∗func_name)

    This function register the test program to the framework.

Variables

- int assume_times

    integers values contain the call times to iif_assume and iif_assert, used to validate a given test
- int assert_times
- int(∗ target_program)(int ∗) = NULL

    The pointer to test program, DO NOT assign directly Call register_program to set its value.
- int minv = -100
- int maxv = 100

7.29.1 Function Documentation

7.29.1.1 bool check_target_program (int(∗)(int ∗) *func*)

7.29.1.2 bool register_program (int(∗)(int ∗) *func,* const char ∗ *func_name =* 0)

This function register the test program to the framework.

Parameters

func	The function to be tested It involves a small validation test on the given function.
fun_name	defines the function name, can be ignored, or set to NULL

Returns

a boolean value false only when the given function is not valid.

7.29.2 Variable Documentation

7.29.2.1 int assert_times

7.29.2.2 int assume_times

integers values contain the call times to iif_assume and iif_assert, used to validate a given test

7.29.2.3 int maxv = 100

7.29.2.4 int minv = -100

7.29.2.5 int(∗ target_program) (int ∗) = NULL

The pointer to test program, DO NOT assign directly Call register_program to set its value.

7.30 src/equation.cpp File Reference

```
#include "equation.h"
#include <cstdlib>
#include <vector>
#include <iostream>
```

Functions

- double _roundoff (double x)
- std::ostream & operator<< (std::ostream &out, const Solution &sol)
- std::ostream & operator<< (std::ostream &out, const Equation &equ)

Variables

- const double UPBOUND = pow(0.1, PRECISION)

7.30.1 Function Documentation

7.30.1.1 double _roundoff (double *x*) [inline]

7.30.1.2 std::ostream& operator<< (std::ostream & *out*, const **Solution** & *sol*)

Parameters

sol	The solution object to be output

7.30.1.3 std::ostream& operator<< (std::ostream & *out*, const **Equation** & *equ*)

Example: 2{0} + 3{1} >= 5

Parameters

equ	the equation to be ouput

7.30.2 Variable Documentation

7.30.2.1 const double UPBOUND = pow(0.1, **PRECISION**)

7.31 src/iif_svm_i_learn.cpp File Reference

```
#include "config.h"
#include "ml_algo.h"
#include "svm.h"
#include "color.h"
#include "equation.h"
#include "iif_learn.h"
#include "iif_svm_i_learn.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
```

Functions

- static void print_null (const char ∗s)

7.31.1 Function Documentation

7.31.1.1 static void print_null (const char ∗ *s*) [static]

7.32 src/iif_svm_learn.cpp File Reference

```
#include "config.h"
#include "ml_algo.h"
#include "svm.h"
#include "color.h"
#include "equation.h"
#include "iif_svm_learn.h"
#include <iostream>
#include <float.h>
#include <string.h>
#include <assert.h>
```

Functions

- static void print_null (const char ∗s)

7.32.1 Function Documentation

7.32.1.1 static void print_null (const char ∗ *s*) [static]

7.33 src/instrumentation.cpp File Reference

```
#include <iostream>
#include <cstdio>
#include <cstdlib>
#include <time.h>
#include "instrumentation.h"
#include <assert.h>
```

Functions

- int add_state_int (int first...)
- int add_state_double (double first,...)
- int before_loop ()

  This function should be called each time before executing loop.
- int after_loop (States ∗gsets)

  This function should be called each time after executing loop.
- int m_double (double ∗p)

  function jump list, DONOT use it unless you know what you are doing
- int m_int (int ∗p)

  record furntions for each platform

Variables

- bool _passP = false

  a flag to justify whether the given input has pass loop precondition
- bool _passQ = false

  a flag to justify whether the given input has pass loop postcondition
- int assume_times = 0

  integers values contain the call times to iif_assume and iif_assert, used to validate a given test
- int assert_times = 0
- char lt [4][10] = { "Negative", "Question", "Positive", "Bugtrace"}
- char(∗ LabelTable)[10] = &lt[1]
- double temp_states [256][VARS]
- int temp_index

7.33.1 Function Documentation

7.33.1.1 int add_state_double (double *first,* ...)

7.33.1.2 int add_state_int (int *first...*)

7.33.1.3 int after_loop (States ∗)

This function should be called each time after executing loop.

Parameters

void	

Returns

void

7.33.1.4 int before_loop ( )

This function should be called each time before executing loop.

Parameters

void	

Returns

void

7.33.1.5 int m_double (double ∗ *p*)

function jump list, DONOT use it unless you know what you are doing

7.33.1.6 int m_int (int ∗)

record furntions for each platform

function jump list, DONOT use it unless you know what you are doing

7.33.2 Variable Documentation

7.33.2.1 bool _passP = false

a flag to justify whether the given input has pass loop precondition

7.33.2.2 bool _passQ = false

a flag to justify whether the given input has pass loop postcondition

7.33.2.3 int assert_times = 0

7.33.2.4 int assume_times = 0

integers values contain the call times to iif_assume and iif_assert, used to validate a given test

7.33.2.5 char(∗ LabelTable)[10] = &lt[1]

7.33.2.6 char lt[4][10] = { "Negative", "Question", "Positive", "Bugtrace"}

7.33.2.7 int temp_index

7.33.2.8 double temp_states[256][VARS]

7.34 src/perceptron.cpp File Reference

```
#include "perceptron.h"
#include "string.h"
```

Functions

- std::ostream & operator<< (std::ostream &out, const Perceptron &perceptron)

7.34.1 Function Documentation

7.34.1.1 std::ostream& operator<< (std::ostream & *out,* const Perceptron & *perceptron*)

7.35 src/states.cpp File Reference

```
#include "config.h"
#include "string.h"
#include "states.h"
#include <cstdlib>
#include <vector>
#include <iostream>
```

Functions

- std::ostream & operator<< (std::ostream &out, const States &ss)

7.35.1 Function Documentation

7.35.1.1 std::ostream& operator<< (std::ostream & *out,* const States & *ss*)

7.36 src/svm.cpp File Reference

```
#include "svm.h"
#include "svm_core.h"
#include "string.h"
```

Functions

- std::ostream & operator<< (std::ostream &out, const SVM &svm)

7.36.1 Function Documentation

7.36.1.1 std::ostream& operator<< (std::ostream & *out,* const SVM & *svm*)

7.37 src/svm_core.cpp File Reference

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <float.h>
#include <string.h>
#include <stdarg.h>
#include <limits.h>
#include <locale.h>
#include "svm.h"
```

Data Structures

- class Cache
- class QMatrix
- class Kernel
- class Solver
- struct Solver::SolutionInfo
- class Solver_NU
- class SVC_Q
- class ONE_CLASS_Q
- class SVR_Q
- struct decision_function

Macros

- #define INF HUGE_VAL
- #define TAU 1e-12
- #define Malloc(type, n) (type ∗)malloc((n)∗sizeof(type))
- #define FSCANF(_stream, _format, _var) do{ if (fscanf(_stream, _format, _var) != 1) return false; }while(0)

Typedefs

- typedef float Qfloat
- typedef signed char schar

Functions

- template<class T >
  static T min (T x, T y)
- template<class T >
  static T max (T x, T y)
- template<class T >
  static void swap (T &x, T &y)
- template<class S , class T >
  static void clone (T ∗&dst, S ∗src, int n)
- static double powi (double base, int times)
- static void print_string_stdout (const char ∗s)
- static void info (const char ∗fmt,...)
- static void solve_c_svc (const svm_problem ∗prob, const svm_parameter ∗param, double ∗alpha, Solver::↩ SolutionInfo ∗si, double Cp, double Cn)
- static void solve_nu_svc (const svm_problem ∗prob, const svm_parameter ∗param, double ∗alpha, Solver↩ ::SolutionInfo ∗si)
- static void solve_one_class (const svm_problem ∗prob, const svm_parameter ∗param, double ∗alpha, Solver::SolutionInfo ∗si)
- static void solve_epsilon_svr (const svm_problem ∗prob, const svm_parameter ∗param, double ∗alpha, Solver::SolutionInfo ∗si)
- static void solve_nu_svr (const svm_problem ∗prob, const svm_parameter ∗param, double ∗alpha, Solver↩ ::SolutionInfo ∗si)
- static decision_function svm_train_one (const svm_problem ∗prob, const svm_parameter ∗param, double Cp, double Cn)
- static void sigmoid_train (int l, const double ∗dec_values, const double ∗labels, double &A, double &B)
- static double sigmoid_predict (double decision_value, double A, double B)
- static void multiclass_probability (int k, double ∗∗r, double ∗p)

- static void svm_binary_svc_probability (const svm_problem ∗prob, const svm_parameter ∗param, double Cp, double Cn, double &probA, double &probB)
- static double svm_svr_probability (const svm_problem ∗prob, const svm_parameter ∗param)
- static void svm_group_classes (const svm_problem ∗prob, int ∗nr_class_ret, int ∗∗label_ret, int ∗∗start_ret, int ∗∗count_ret, int ∗perm)
- svm_model ∗ svm_train (const svm_problem ∗prob, const svm_parameter ∗param)
- void svm_cross_validation (const svm_problem ∗prob, const svm_parameter ∗param, int nr_fold, double ∗target)
- int svm_get_svm_type (const svm_model ∗model)
- int svm_get_nr_class (const svm_model ∗model)
- void svm_get_labels (const svm_model ∗model, int ∗label)
- void svm_get_sv_indices (const svm_model ∗model, int ∗indices)
- int svm_get_nr_sv (const svm_model ∗model)
- double svm_get_svr_probability (const svm_model ∗model)
- double svm_predict_values (const svm_model ∗model, const svm_node ∗x, double ∗dec_values)
- double svm_predict (const svm_model ∗model, const svm_node ∗x)
- double svm_predict_probability (const svm_model ∗model, const svm_node ∗x, double ∗prob_estimates)
- int svm_save_model (const char ∗model_file_name, const svm_model ∗model)
- static char ∗ readline (FILE ∗input)
- bool read_model_header (FILE ∗fp, svm_model ∗model)
- svm_model ∗ svm_load_model (const char ∗model_file_name)
- void svm_free_model_content (svm_model ∗model_ptr)
- void svm_free_and_destroy_model (svm_model ∗∗model_ptr_ptr)
- void svm_destroy_param (svm_parameter ∗param)
- const char ∗ svm_check_parameter (const svm_problem ∗prob, const svm_parameter ∗param)
- int svm_check_probability_model (const svm_model ∗model)
- void svm_set_print_string_function (void(∗print_func)(const char ∗))
- void print_svm_samples (const svm_problem ∗sp)
- int svm_model_visualization (const svm_model ∗model, Equation ∗equ)
- struct svm_model ∗ svm_l_train (const struct svm_problem ∗prob, const struct svm_parameter ∗param)

Variables

- int libsvm_version = LIBSVM_VERSION
- struct svm_node ∗ positive_nodes = NULL
- struct svm_node ∗ negative_nodes = NULL
- static void(∗ svm_print_string)(const char ∗) = &print_string_stdout
- static const char ∗ svm_type_table [ ]
- static const char ∗ kernel_type_table [ ]
- static char ∗ line = NULL
- static int max_line_len

7.37.1 Macro Definition Documentation

7.37.1.1 #define FSCANF(*_stream,* *_format,* *_var*) do{ if (fscanf(_stream, _format, _var) != 1) return false; }while(0)

7.37.1.2 #define INF HUGE_VAL

7.37.1.3 #define Malloc(*type,* *n*) (type ∗)malloc((n)∗sizeof(type))

7.37.1.4 #define TAU 1e-12

7.37.2 Typedef Documentation

7.37.2.1 typedef float Qfloat

7.37.2.2 typedef signed char schar

7.37.3 Function Documentation

7.37.3.1 template<class S , class T > static void clone (T ∗& *dst*, S ∗ *src*, int *n*) [inline], [static]

7.37.3.2 static void info (const char ∗ *fmt*, ...) [static]

7.37.3.3 template<class T > static T max (T *x*, T *y*) [inline], [static]

7.37.3.4 template<class T > static T min (T *x*, T *y*) [inline], [static]

7.37.3.5 static void multiclass_probability (int *k*, double ∗∗ *r*, double ∗ *p*) [static]

7.37.3.6 static double powi (double *base*, int *times*) [inline], [static]

7.37.3.7 static void print_string_stdout (const char ∗ *s*) [static]

7.37.3.8 void print_svm_samples (const svm_problem ∗ *sp*)

7.37.3.9 bool read_model_header (FILE ∗ *fp*, svm_model ∗ *model*)

7.37.3.10 static char∗ readline (FILE ∗ *input*) [static]

7.37.3.11 static double sigmoid_predict (double *decision_value*, double *A*, double *B*) [static]

7.37.3.12 static void sigmoid_train (int *l*, const double ∗ *dec_values*, const double ∗ *labels*, double & *A*, double & *B*) [static]

7.37.3.13 static void solve_c_svc (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double ∗ *alpha*, Solver::SolutionInfo ∗ *si*, double *Cp*, double *Cn*) [static]

7.37.3.14 static void solve_epsilon_svr (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double ∗ *alpha*, Solver::SolutionInfo ∗ *si*) [static]

7.37.3.15 static void solve_nu_svc (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double ∗ *alpha*, Solver::SolutionInfo ∗ *si*) [static]

7.37.3.16 static void solve_nu_svr (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double ∗ *alpha*, Solver::SolutionInfo ∗ *si*) [static]

7.37.3.17 static void solve_one_class (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double ∗ *alpha*, Solver::SolutionInfo ∗ *si*) [static]

7.37.3.18 static void svm_binary_svc_probability (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, double *Cp*, double *Cn*, double & *probA*, double & *probB*) [static]

7.37.3.19 const char∗ svm_check_parameter (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*)

7.37.3.20 int svm_check_probability_model (const svm_model ∗ *model*)

7.37.3.21 void svm_cross_validation (const svm_problem ∗ *prob*, const svm_parameter ∗ *param*, int *nr_fold*, double ∗ *target*)

7.37.3.22 void svm_destroy_param (svm_parameter ∗ *param*)

7.37.3.23 void svm_free_and_destroy_model (svm_model ∗∗ *model_ptr_ptr*)

7.37.3.24 void svm_free_model_content (svm_model ∗ *model_ptr*)

7.37.3.25 void svm_get_labels (const svm_model ∗ *model,* int ∗ *label*)

7.37.3.26 int svm_get_nr_class (const svm_model ∗ *model*)

7.37.3.27 int svm_get_nr_sv (const svm_model ∗ *model*)

7.37.3.28 void svm_get_sv_indices (const svm_model ∗ *model,* int ∗ *indices*)

7.37.3.29 int svm_get_svm_type (const svm_model ∗ *model*)

7.37.3.30 double svm_get_svr_probability (const svm_model ∗ *model*)

7.37.3.31 static void svm_group_classes (const svm_problem ∗ *prob,* int ∗ *nr_class_ret,* int ∗∗ *label_ret,* int ∗∗ *start_ret,* int ∗∗ *count_ret,* int ∗ *perm*) [static]

7.37.3.32 struct svm_model∗ svm_l_train (const struct svm_problem ∗ *prob,* const struct svm_parameter ∗ *param*)

7.37.3.33 svm_model∗ svm_load_model (const char ∗ *model_file_name*)

7.37.3.34 int svm_model_visualization (const svm_model ∗ *model,* Equation ∗ *equ*)

7.37.3.35 double svm_predict (const svm_model ∗ *model,* const svm_node ∗ *x*)

7.37.3.36 double svm_predict_probability (const svm_model ∗ *model,* const svm_node ∗ *x,* double ∗ *prob_estimates*)

7.37.3.37 double svm_predict_values (const svm_model ∗ *model,* const svm_node ∗ *x,* double ∗ *dec_values*)

7.37.3.38 int svm_save_model (const char ∗ *model_file_name,* const svm_model ∗ *model*)

7.37.3.39 void svm_set_print_string_function (void(∗)(const char ∗) *print_func*)

7.37.3.40 static double svm_svr_probability (const svm_problem ∗ *prob,* const svm_parameter ∗ *param*) [static]

7.37.3.41 svm_model∗ svm_train (const svm_problem ∗ *prob,* const svm_parameter ∗ *param*)

7.37.3.42 static decision_function svm_train_one (const svm_problem ∗ *prob,* const svm_parameter ∗ *param,* double *Cp,* double *Cn*) [static]

7.37.3.43 template<class T > static void swap (T & *x,* T & *y*) [inline], [static]

7.37.4 Variable Documentation

7.37.4.1 const char∗ kernel_type_table[ ] [static]

Initial value:

```
=
{
    "linear","polynomial","rbf","sigmoid","precomputed",NULL
}
```

7.37.4.2 int libsvm_version = LIBSVM_VERSION

7.37.4.3 char∗ line = NULL [static]

7.37.4.4 int max_line_len [static]

7.37.4.5 struct svm_node∗ negative_nodes = NULL

7.37.4.6 struct svm_node∗ positive_nodes = NULL

7.37.4.7 void(∗ svm_print_string) (const char ∗) = &print_string_stdout [static]

7.37.4.8 const char∗ svm_type_table[ ] [static]

Initial value:

```
=
{
    "c_svc","nu_svc","one_class","epsilon_svr","nu_svr",NULL
}
```

7.38 src/svm_i.cpp File Reference

```
#include "svm_i.h"
#include "string.h"
#include <vector>
```

Functions

- std::ostream & operator<< (std::ostream &out, const SVM_I &svm_i)

7.38.1 Function Documentation

7.38.1.1 std::ostream& operator<< (std::ostream & *out*, const SVM_I & *svm_i*)

7.39 test/1_conj.cpp File Reference

```
#include "iif.h"
#include <iostream>
```

Functions

- static int nondet ()
- int conj (int ∗a)
- int main (int argc, char ∗∗argv)

7.39.1 Function Documentation

7.39.1.1 int conj (int ∗ *a*)

7.39.1.2 int main (int *argc,* char ∗∗ *argv*)

7.39.1.3 static int nondet ( ) `[static]`

7.40 test/2_ex1.cpp File Reference

```
#include "iif.h"
```

Functions

- static int nondet ()
- int ex1 (int ∗a)
- int main (int argc, char ∗∗argv)

7.40.1 Function Documentation

7.40.1.1 int ex1 (int ∗ *a*)

7.40.1.2 int main (int *argc,* char ∗∗ *argv*)

7.40.1.3 static int nondet ( ) `[static]`

7.41 test/2_f1.cpp File Reference

```
#include "iif.h"
```

Functions

- int f1 (int ∗a)
- int main (int argc, char ∗∗argv)

7.41.1 Function Documentation

7.41.1.1 int f1 (int ∗ *a*)

7.41.1.2 int main (int *argc,* char ∗∗ *argv*)

7.42 test/2_f2.cpp File Reference

```
#include "iif.h"
#include <iostream>
```

Functions

- int f2 (int ∗a)
- int main (int argc, char ∗∗argv)

7.42.1 Function Documentation

7.42.1.1 int f2 (int ∗ *a*)

7.42.1.2 int main (int *argc,* char ∗∗ *argv*)

7.43 test/2_z3test.cpp File Reference

```
#include "iif.h"
```

Functions

- int main (int argc, char ∗∗argv)

7.43.1 Function Documentation

7.43.1.1 int main (int *argc,* char ∗∗ *argv*)

7.44 test/3_f3.cpp File Reference

```
#include "iif.h"
```

Functions

- int f3 (int ∗a)
- int main (int argc, char ∗∗argv)

7.44.1 Function Documentation

7.44.1.1 int f3 (int ∗ *a*)

7.44.1.2 int main (int *argc,* char ∗∗ *argv*)

Index