

Invariant Inference Framework

Generated by Doxygen 1.8.11

Contents

1	Invariant Inference Framework:	1
2	Bug List	3
3	Hierarchical Index	5
3.1	Class Hierarchy	5
4	Data Structure Index	7
4.1	Data Structures	7
5	File Index	9
5.1	File List	9
6	Data Structure Documentation	11
6.1	Cache Class Reference	11
6.1.1	Constructor & Destructor Documentation	11
6.1.1.1	Cache(int l, long int size)	11
6.1.1.2	~Cache()	11
6.1.2	Member Function Documentation	11
6.1.2.1	get_data(const int index, Qfloat **data, int len)	11
6.1.2.2	swap_index(int i, int j)	11
6.2	decision_function Struct Reference	11
6.2.1	Field Documentation	11
6.2.1.1	alpha	11
6.2.1.2	rho	12
6.3	Equation Class Reference	12
6.3.1	Detailed Description	13
6.3.2	Constructor & Destructor Documentation	13
6.3.2.1	Equation()	13
6.3.2.2	Equation(double a0,...)	13
6.3.2.3	Equation(const Equation &equ)	13
6.3.3	Member Function Documentation	13
6.3.3.1	calc(Equation &equ, double *sol)	13

6.3.3.2	<code>imply(const Equation &e2)</code>	13
6.3.3.3	<code>is_similar(const Equation &e, int precision=PRECISION)</code>	14
6.3.3.4	<code>linear_solver(Solution &sol)</code>	14
6.3.3.5	<code>linear_solver(const Equation *equ, Solution &sol)</code>	14
6.3.3.6	<code>operator=(const Equation &rhs)</code>	14
6.3.3.7	<code>roundoff(Equation &e)</code>	15
6.3.4	Friends And Related Function Documentation	15
6.3.4.1	<code>operator<<</code>	15
6.3.5	Field Documentation	15
6.3.5.1	<code>theta</code>	15
6.3.5.2	<code>theta0</code>	15
6.4	IIF_learn Class Reference	15
6.4.1	Constructor & Destructor Documentation	16
6.4.1.1	<code>IIF_learn(States *gsets, int(*func)(int *))</code>	16
6.4.1.2	<code>IIF_learn()</code>	16
6.4.2	Member Function Documentation	16
6.4.2.1	<code>init_gsets()</code>	16
6.4.2.2	<code>learn()=0</code>	16
6.4.2.3	<code>run_target(Solution &input)</code>	16
6.4.3	Field Documentation	16
6.4.3.1	<code>func</code>	16
6.4.3.2	<code>gsets</code>	16
6.5	IIF_svm_i_learn Class Reference	17
6.5.1	Constructor & Destructor Documentation	17
6.5.1.1	<code>IIF_svm_i_learn(States *gsets, int(*func)(int *), int max_iteration=max_iter)</code>	17
6.5.1.2	<code>IIF_svm_i_learn()</code>	17
6.5.2	Member Function Documentation	17
6.5.2.1	<code>learn()</code>	17
6.5.3	Field Documentation	17
6.5.3.1	<code>max_iteration</code>	17
6.5.3.2	<code>svm_i</code>	17
6.6	IIF_svm_learn Class Reference	17
6.6.1	Constructor & Destructor Documentation	18
6.6.1.1	<code>IIF_svm_learn(States *gsets, int(*func)(int *), int max_iteration=max_iter)</code>	18
6.6.1.2	<code>IIF_svm_learn()</code>	18
6.6.2	Member Function Documentation	18
6.6.2.1	<code>learn()</code>	18
6.6.3	Field Documentation	18
6.6.3.1	<code>max_iteration</code>	18
6.6.3.2	<code>svm</code>	18

6.7	Kernel Class Reference	18
6.7.1	Constructor & Destructor Documentation	19
6.7.1.1	Kernel(int l, svm_node *const *x, const svm_parameter ¶m)	19
6.7.1.2	~Kernel()	19
6.7.2	Member Function Documentation	19
6.7.2.1	get_Q(int column, int len) const =0	19
6.7.2.2	get_QD() const =0	19
6.7.2.3	k_function(const svm_node *x, const svm_node *y, const svm_parameter ¶m)	19
6.7.2.4	swap_index(int i, int j) const	19
6.7.3	Field Documentation	20
6.7.3.1	kernel_function	20
6.8	ML_Algo Class Reference	20
6.8.1	Constructor & Destructor Documentation	21
6.8.1.1	ML_Algo()	21
6.8.2	Member Function Documentation	21
6.8.2.1	_print(std::ostream &out) const	21
6.8.2.2	check_question_set(States &qset)=0	21
6.8.2.3	get_converged(Equation *previous_equations, int equation_num)=0	21
6.8.2.4	predict(double *x, int flag)=0	21
6.8.2.5	predict_on_training_set()=0	22
6.8.2.6	prepare_training_data(States *gsets, int &pre_positive_size, int &pre_negative_size)=0	22
6.8.2.7	roundoff(int &equation_num)=0	22
6.8.2.8	size()=0	23
6.8.2.9	train()=0	23
6.8.3	Friends And Related Function Documentation	23
6.8.3.1	operator<<	23
6.9	ONE_CLASS_Q Class Reference	23
6.9.1	Constructor & Destructor Documentation	24
6.9.1.1	ONE_CLASS_Q(const svm_problem &prob, const svm_parameter ¶m)	24
6.9.1.2	~ONE_CLASS_Q()	24
6.9.2	Member Function Documentation	24
6.9.2.1	get_Q(int i, int len) const	24
6.9.2.2	get_QD() const	24
6.9.2.3	swap_index(int i, int j) const	24
6.10	Perceptron Class Reference	24
6.10.1	Constructor & Destructor Documentation	25
6.10.1.1	Perceptron(void(*f)(const char *)=NULL, int max_size=10000)	25
6.10.1.2	~Perceptron()	25
6.10.2	Member Function Documentation	25

6.10.2.1	check_question_set(States &qset)	25
6.10.2.2	predict(double *v, int label=0)	25
6.10.2.3	predict_on_training_set()	26
6.10.2.4	prepare_training_data(States *gsets, int &pre_positive_size, int &pre_negative_size)	26
6.10.2.5	roundoff(int &num)	26
6.10.2.6	size()	27
6.10.2.7	train()	27
6.10.3	Friends And Related Function Documentation	27
6.10.3.1	operator<<	27
6.10.4	Field Documentation	27
6.10.4.1	length	27
6.10.4.2	main_equation	27
6.10.4.3	training_label	27
6.10.4.4	training_set	27
6.11	QMatrix Class Reference	27
6.11.1	Constructor & Destructor Documentation	28
6.11.1.1	~QMatrix()	28
6.11.2	Member Function Documentation	28
6.11.2.1	get_Q(int column, int len) const =0	28
6.11.2.2	get_QD() const =0	28
6.11.2.3	swap_index(int i, int j) const =0	28
6.12	Solution Class Reference	28
6.12.1	Detailed Description	29
6.12.2	Constructor & Destructor Documentation	29
6.12.2.1	Solution()	29
6.12.2.2	Solution(double a0,...)	29
6.12.3	Friends And Related Function Documentation	29
6.12.3.1	operator<<	29
6.12.4	Field Documentation	29
6.12.4.1	x	29
6.13	Solver::SolutionInfo Struct Reference	29
6.13.1	Field Documentation	30
6.13.1.1	obj	30
6.13.1.2	r	30
6.13.1.3	rho	30
6.13.1.4	upper_bound_n	30
6.13.1.5	upper_bound_p	30
6.14	Solver Class Reference	30
6.14.1	Member Enumeration Documentation	31

6.14.1.1	anonymous enum	31
6.14.2	Constructor & Destructor Documentation	31
6.14.2.1	Solver()	31
6.14.2.2	~Solver()	31
6.14.3	Member Function Documentation	31
6.14.3.1	calculate_rho()	31
6.14.3.2	do_shrinking()	31
6.14.3.3	get_C(int i)	31
6.14.3.4	is_free(int i)	31
6.14.3.5	is_lower_bound(int i)	31
6.14.3.6	is_upper_bound(int i)	31
6.14.3.7	reconstruct_gradient()	31
6.14.3.8	select_working_set(int &i, int &j)	31
6.14.3.9	Solve(int l, const QMatrix &Q, const double *p_, const schar *y_, double *alpha_↵ _, double Cp, double Cn, double eps, SolutionInfo *si, int shrinking)	31
6.14.3.10	swap_index(int i, int j)	32
6.14.3.11	update_alpha_status(int i)	32
6.14.4	Field Documentation	32
6.14.4.1	active_set	32
6.14.4.2	active_size	32
6.14.4.3	alpha	32
6.14.4.4	alpha_status	32
6.14.4.5	Cn	32
6.14.4.6	Cp	32
6.14.4.7	eps	32
6.14.4.8	G	32
6.14.4.9	G_bar	32
6.14.4.10	l	32
6.14.4.11	p	32
6.14.4.12	Q	32
6.14.4.13	QD	32
6.14.4.14	unshrink	32
6.14.4.15	y	32
6.15	Solver_NU Class Reference	32
6.15.1	Constructor & Destructor Documentation	33
6.15.1.1	Solver_NU()	33
6.15.2	Member Function Documentation	33
6.15.2.1	Solve(int l, const QMatrix &Q, const double *p, const schar *y, double *alpha, double Cp, double Cn, double eps, SolutionInfo *si, int shrinking)	33
6.16	States Class Reference	33

6.16.1	Constructor & Destructor Documentation	33
6.16.1.1	States()	33
6.16.1.2	~States()	33
6.16.2	Member Function Documentation	33
6.16.2.1	add_states(double st[][VARS], int len)	34
6.16.2.2	print_trace(int num)	34
6.16.2.3	size()	34
6.16.2.4	traces_num()	34
6.16.3	Friends And Related Function Documentation	34
6.16.3.1	operator<<	34
6.16.4	Field Documentation	34
6.16.4.1	index	34
6.16.4.2	label	34
6.16.4.3	p_index	34
6.16.4.4	values	34
6.17	SVC_Q Class Reference	34
6.17.1	Constructor & Destructor Documentation	34
6.17.1.1	SVC_Q(const svm_problem &prob, const svm_parameter ¶m, const schar *y_)	35
6.17.1.2	~SVC_Q()	35
6.17.2	Member Function Documentation	35
6.17.2.1	get_Q(int i, int len) const	35
6.17.2.2	get_QD() const	35
6.17.2.3	swap_index(int i, int j) const	35
6.18	SVM Class Reference	35
6.18.1	Constructor & Destructor Documentation	36
6.18.1.1	SVM(void(*f)(const char *)=NULL, int size=10000)	36
6.18.1.2	~SVM()	36
6.18.2	Member Function Documentation	36
6.18.2.1	_print(std::ostream &out) const	36
6.18.2.2	check_question_set(States &qset)	36
6.18.2.3	get_converged(Equation *, int)	37
6.18.2.4	predict(double *v, int label=0)	37
6.18.2.5	predict_on_training_set()	37
6.18.2.6	prepare_training_data(States *gsets, int &pre_positive_size, int &pre_negative_size)	38
6.18.2.7	roundoff(int &num)	38
6.18.2.8	size()	38
6.18.2.9	train()	38
6.18.3	Friends And Related Function Documentation	39

6.18.3.1	operator<<	39
6.18.4	Field Documentation	39
6.18.4.1	main_equation	39
6.18.4.2	max_size	39
6.18.4.3	model	39
6.18.4.4	param	39
6.18.4.5	problem	39
6.18.4.6	training_label	39
6.18.4.7	training_set	39
6.19	SVM_I Class Reference	39
6.19.1	Constructor & Destructor Documentation	40
6.19.1.1	SVM_I(void(*f)(const char *)=NULL, int size=10000, int equ=16)	40
6.19.1.2	~SVM_I()	40
6.19.2	Member Function Documentation	40
6.19.2.1	_print(std::ostream &out) const	40
6.19.2.2	check_question_set(States &qset)	40
6.19.2.3	get_converged(Equation *, int)	41
6.19.2.4	predict(double *v, int label=0)	41
6.19.2.5	predict_on_training_set()	41
6.19.2.6	prepare_training_data(States *gsets, int &pre_positive_size, int &pre_negative_size)	42
6.19.2.7	roundoff(int &num)	42
6.19.2.8	size()	42
6.19.2.9	train()	42
6.19.3	Friends And Related Function Documentation	43
6.19.3.1	operator<<	43
6.19.4	Field Documentation	43
6.19.4.1	equ_num	43
6.19.4.2	equations	43
6.19.4.3	max_equ	43
6.19.4.4	model	43
6.19.4.5	negatives	43
6.19.4.6	param	43
6.20	svm_model Struct Reference	43
6.20.1	Field Documentation	43
6.20.1.1	free_sv	43
6.20.1.2	l	43
6.20.1.3	label	43
6.20.1.4	nr_class	43
6.20.1.5	nSV	44

6.20.1.6	param	44
6.20.1.7	probA	44
6.20.1.8	probB	44
6.20.1.9	rho	44
6.20.1.10	SV	44
6.20.1.11	sv_coef	44
6.20.1.12	sv_indices	44
6.21	svm_node Struct Reference	44
6.21.1	Friends And Related Function Documentation	44
6.21.1.1	operator<<	44
6.21.2	Field Documentation	44
6.21.2.1	value	44
6.22	svm_parameter Struct Reference	44
6.22.1	Field Documentation	45
6.22.1.1	C	45
6.22.1.2	cache_size	45
6.22.1.3	coef0	45
6.22.1.4	degree	45
6.22.1.5	eps	45
6.22.1.6	gamma	45
6.22.1.7	kernel_type	45
6.22.1.8	nr_weight	45
6.22.1.9	nu	45
6.22.1.10	p	45
6.22.1.11	probability	45
6.22.1.12	shrinking	45
6.22.1.13	svm_type	45
6.22.1.14	weight	45
6.22.1.15	weight_label	45
6.23	svm_problem Struct Reference	46
6.23.1	Friends And Related Function Documentation	46
6.23.1.1	operator<<	46
6.23.2	Field Documentation	46
6.23.2.1	l	46
6.23.2.2	x	46
6.23.2.3	y	46
6.24	SVR_Q Class Reference	46
6.24.1	Constructor & Destructor Documentation	47
6.24.1.1	SVR_Q(const svm_problem &prob, const svm_parameter &param)	47
6.24.1.2	~SVR_Q()	47

6.24.2	Member Function Documentation	47
6.24.2.1	get_Q(int i, int len) const	47
6.24.2.2	get_QD() const	47
6.24.2.3	swap_index(int i, int j) const	47
7	File Documentation	49
7.1	build/CMakeCache.txt File Reference	49
7.2	build/CMakeFiles/2.8.12.2/CompilerIdC/CMakeCCompilerId.c File Reference	49
7.2.1	Macro Definition Documentation	49
7.2.1.1	ARCHITECTURE_ID	49
7.2.1.2	COMPILER_ID	49
7.2.1.3	DEC	49
7.2.1.4	HEX	50
7.2.1.5	PLATFORM_ID	50
7.2.2	Function Documentation	50
7.2.2.1	main(int argc, char *argv[])	50
7.2.3	Variable Documentation	50
7.2.3.1	info_arch	50
7.2.3.2	info_compiler	50
7.2.3.3	info_platform	50
7.3	build/CMakeFiles/2.8.12.2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference	50
7.3.1	Macro Definition Documentation	50
7.3.1.1	ARCHITECTURE_ID	50
7.3.1.2	COMPILER_ID	50
7.3.1.3	DEC	51
7.3.1.4	HEX	51
7.3.1.5	PLATFORM_ID	51
7.3.2	Function Documentation	51
7.3.2.1	main(int argc, char *argv[])	51
7.3.3	Variable Documentation	51
7.3.3.1	info_arch	51
7.3.3.2	info_compiler	51
7.3.3.3	info_platform	51
7.4	build/CMakeFiles/conj.dir/link.txt File Reference	51
7.5	build/CMakeFiles/ex1.dir/link.txt File Reference	51
7.6	build/CMakeFiles/f1.dir/link.txt File Reference	51
7.7	build/CMakeFiles/f2.dir/link.txt File Reference	51
7.8	build/CMakeFiles/f3.dir/link.txt File Reference	51
7.9	build/CMakeFiles/z3test.dir/link.txt File Reference	51
7.10	build/CMakeFiles/TargetDirectories.txt File Reference	51

7.11 CMakeLists.txt File Reference	52
7.12 include/color.h File Reference	52
7.12.1 Detailed Description	52
7.12.2 Enumeration Type Documentation	52
7.12.2.1 color	52
7.12.3 Function Documentation	53
7.12.3.1 set_console_color(std::ostream &out, int color=YELLOW)	53
7.12.3.2 unset_console_color(std::ostream &out)	53
7.13 include/config.h File Reference	53
7.13.1 Detailed Description	54
7.13.2 Macro Definition Documentation	54
7.13.2.1 PRECISION	54
7.13.2.2 VARS	54
7.13.3 Function Documentation	54
7.13.3.1 register_program(int(*func)(int *), const char *func_name=0)	54
7.13.4 Variable Documentation	54
7.13.4.1 after_exes	54
7.13.4.2 init_exes	54
7.13.4.3 max_items	54
7.13.4.4 max_iter	55
7.13.4.5 random_exes	55
7.13.4.6 target_program	55
7.14 include/equation.h File Reference	55
7.14.1 Detailed Description	55
7.14.2 Variable Documentation	56
7.14.2.1 maxv	56
7.14.2.2 minv	56
7.15 include/iif.h File Reference	56
7.15.1 Detailed Description	56
7.15.2 Variable Documentation	56
7.15.2.1 maxv	56
7.15.2.2 minv	56
7.16 include/iif_assert.h File Reference	56
7.16.1 Detailed Description	57
7.16.2 Macro Definition Documentation	57
7.16.2.1 iif_assert	57
7.16.2.2 iif_assume	57
7.16.3 Variable Documentation	58
7.16.3.1 _passP	58
7.16.3.2 _passQ	58

7.16.3.3	assert_times	58
7.16.3.4	assume_times	58
7.17	include/iif_learn.h File Reference	58
7.18	include/iif_svm_i_learn.h File Reference	59
7.19	include/iif_svm_learn.h File Reference	59
7.20	include/instrumentation.h File Reference	59
7.20.1	Detailed Description	60
7.20.2	Enumeration Type Documentation	60
7.20.2.1	trace_type	60
7.20.3	Function Documentation	60
7.20.3.1	add_state_double(double first,...)	60
7.20.3.2	add_state_int(int first,...)	60
7.20.3.3	after_loop(States *)	60
7.20.3.4	before_loop()	61
7.20.3.5	m_double(double *)	61
7.20.3.6	m_int(int *)	61
7.21	include/ml_algo.h File Reference	61
7.21.1	Detailed Description	61
7.22	include/perceptron.h File Reference	62
7.23	include/states.h File Reference	62
7.24	include/svm.h File Reference	62
7.25	include/svm_core.h File Reference	62
7.25.1	Macro Definition Documentation	63
7.25.1.1	LIBSVM_VERSION	63
7.25.2	Enumeration Type Documentation	63
7.25.2.1	anonymous enum	63
7.25.2.2	anonymous enum	64
7.25.3	Function Documentation	64
7.25.3.1	print_svm_samples(const svm_problem *sp)	64
7.25.3.2	svm_check_parameter(const struct svm_problem *prob, const struct svm_↵ parameter *param)	64
7.25.3.3	svm_check_probability_model(const struct svm_model *model)	64
7.25.3.4	svm_cross_validation(const struct svm_problem *prob, const struct svm_↵ parameter *param, int nr_fold, double *target)	64
7.25.3.5	svm_destroy_param(struct svm_parameter *param)	64
7.25.3.6	svm_free_and_destroy_model(struct svm_model **model_ptr_ptr)	64
7.25.3.7	svm_free_model_content(struct svm_model *model_ptr)	64
7.25.3.8	svm_get_labels(const struct svm_model *model, int *label)	64
7.25.3.9	svm_get_nr_class(const struct svm_model *model)	64
7.25.3.10	svm_get_nr_sv(const struct svm_model *model)	64

7.25.3.11	<code>svm_get_sv_indices(const struct svm_model *model, int *sv_indices)</code>	64
7.25.3.12	<code>svm_get_svm_type(const struct svm_model *model)</code>	64
7.25.3.13	<code>svm_get_svr_probability(const struct svm_model *model)</code>	64
7.25.3.14	<code>svm_l_train(const struct svm_problem *prob, const struct svm_parameter *param)</code>	64
7.25.3.15	<code>svm_load_model(const char *model_file_name)</code>	64
7.25.3.16	<code>svm_model_visualization(const struct svm_model *model, Equation *equ)</code>	64
7.25.3.17	<code>svm_predict(const struct svm_model *model, const struct svm_node *x)</code>	64
7.25.3.18	<code>svm_predict_probability(const struct svm_model *model, const struct svm_node *x, double *prob_estimates)</code>	64
7.25.3.19	<code>svm_predict_values(const struct svm_model *model, const struct svm_node *x, double *dec_values)</code>	65
7.25.3.20	<code>svm_save_model(const char *model_file_name, const struct svm_model *model)</code>	65
7.25.3.21	<code>svm_set_print_string_function(void(*print_func)(const char *))</code>	65
7.25.3.22	<code>svm_train(const struct svm_problem *prob, const struct svm_parameter *param)</code>	65
7.25.4	Variable Documentation	65
7.25.4.1	<code>libsvm_version</code>	65
7.26	<code>include/svm_i.h</code> File Reference	65
7.27	<code>README.md</code> File Reference	65
7.28	<code>src/color.cpp</code> File Reference	65
7.28.1	Function Documentation	65
7.28.1.1	<code>unset_console_color(std::ostream &out)</code>	65
7.29	<code>src/config.cpp</code> File Reference	65
7.29.1	Function Documentation	66
7.29.1.1	<code>check_target_program(int(*func)(int *))</code>	66
7.29.1.2	<code>register_program(int(*func)(int *), const char *func_name)</code>	66
7.29.2	Variable Documentation	66
7.29.2.1	<code>assert_times</code>	66
7.29.2.2	<code>assume_times</code>	66
7.29.2.3	<code>maxv</code>	66
7.29.2.4	<code>minv</code>	66
7.29.2.5	<code>target_program</code>	66
7.30	<code>src/equation.cpp</code> File Reference	67
7.30.1	Function Documentation	67
7.30.1.1	<code>_roundoff(double x)</code>	67
7.30.1.2	<code>operator<<(std::ostream &out, const Solution &sol)</code>	67
7.30.1.3	<code>operator<<(std::ostream &out, const Equation &equ)</code>	67
7.30.2	Variable Documentation	67
7.30.2.1	<code>UPBOUND</code>	67
7.31	<code>src/iif_svm_i_learn.cpp</code> File Reference	68
7.31.1	Function Documentation	68

7.31.1.1	<code>print_null(const char *s)</code>	68
7.32	<code>src/iif_svm_learn.cpp</code> File Reference	68
7.32.1	Function Documentation	68
7.32.1.1	<code>print_null(const char *s)</code>	68
7.33	<code>src/instrumentation.cpp</code> File Reference	68
7.33.1	Function Documentation	69
7.33.1.1	<code>add_state_double(double first,...)</code>	69
7.33.1.2	<code>add_state_int(int first...)</code>	69
7.33.1.3	<code>after_loop(States *gsets)</code>	69
7.33.1.4	<code>before_loop()</code>	69
7.33.1.5	<code>m_double(double *p)</code>	70
7.33.1.6	<code>m_int(int *p)</code>	70
7.33.2	Variable Documentation	70
7.33.2.1	<code>_passP</code>	70
7.33.2.2	<code>_passQ</code>	70
7.33.2.3	<code>assert_times</code>	70
7.33.2.4	<code>assume_times</code>	70
7.33.2.5	<code>LabelTable</code>	70
7.33.2.6	<code>lt</code>	70
7.33.2.7	<code>temp_index</code>	70
7.33.2.8	<code>temp_states</code>	70
7.34	<code>src/perceptron.cpp</code> File Reference	70
7.34.1	Function Documentation	71
7.34.1.1	<code>operator<<(std::ostream &out, const Perceptron &perceptron)</code>	71
7.35	<code>src/states.cpp</code> File Reference	71
7.35.1	Function Documentation	71
7.35.1.1	<code>operator<<(std::ostream &out, const States &ss)</code>	71
7.36	<code>src/svm.cpp</code> File Reference	71
7.36.1	Function Documentation	71
7.36.1.1	<code>operator<<(std::ostream &out, const SVM &svm)</code>	71
7.37	<code>src/svm_core.cpp</code> File Reference	71
7.37.1	Macro Definition Documentation	73
7.37.1.1	<code>FSCANF</code>	73
7.37.1.2	<code>INF</code>	73
7.37.1.3	<code>Malloc</code>	73
7.37.1.4	<code>TAU</code>	73
7.37.2	Typedef Documentation	73
7.37.2.1	<code>Qfloat</code>	74
7.37.2.2	<code>schar</code>	74
7.37.3	Function Documentation	74

7.37.3.1	<code>clone(T *&dst, S *src, int n)</code>	74
7.37.3.2	<code>info(const char *fmt,...)</code>	74
7.37.3.3	<code>max(T x, T y)</code>	74
7.37.3.4	<code>min(T x, T y)</code>	74
7.37.3.5	<code>multiclass_probability(int k, double **r, double *p)</code>	74
7.37.3.6	<code>powi(double base, int times)</code>	74
7.37.3.7	<code>print_string_stdout(const char *s)</code>	74
7.37.3.8	<code>print_svm_samples(const svm_problem *sp)</code>	74
7.37.3.9	<code>read_model_header(FILE *fp, svm_model *model)</code>	74
7.37.3.10	<code>readline(FILE *input)</code>	74
7.37.3.11	<code>sigmoid_predict(double decision_value, double A, double B)</code>	74
7.37.3.12	<code>sigmoid_train(int l, const double *dec_values, const double *labels, double &A, double &B)</code>	74
7.37.3.13	<code>solve_c_svc(const svm_problem *prob, const svm_parameter *param, double *alpha, Solver::SolutionInfo *si, double Cp, double Cn)</code>	74
7.37.3.14	<code>solve_epsilon_svr(const svm_problem *prob, const svm_parameter *param, double *alpha, Solver::SolutionInfo *si)</code>	74
7.37.3.15	<code>solve_nu_svc(const svm_problem *prob, const svm_parameter *param, double *alpha, Solver::SolutionInfo *si)</code>	74
7.37.3.16	<code>solve_nu_svr(const svm_problem *prob, const svm_parameter *param, double *alpha, Solver::SolutionInfo *si)</code>	74
7.37.3.17	<code>solve_one_class(const svm_problem *prob, const svm_parameter *param, double *alpha, Solver::SolutionInfo *si)</code>	74
7.37.3.18	<code>svm_binary_svc_probability(const svm_problem *prob, const svm_parameter *param, double Cp, double Cn, double &probA, double &probB)</code>	74
7.37.3.19	<code>svm_check_parameter(const svm_problem *prob, const svm_parameter *param)</code>	74
7.37.3.20	<code>svm_check_probability_model(const svm_model *model)</code>	74
7.37.3.21	<code>svm_cross_validation(const svm_problem *prob, const svm_parameter *param, int nr_fold, double *target)</code>	74
7.37.3.22	<code>svm_destroy_param(svm_parameter *param)</code>	75
7.37.3.23	<code>svm_free_and_destroy_model(svm_model **model_ptr_ptr)</code>	75
7.37.3.24	<code>svm_free_model_content(svm_model *model_ptr)</code>	75
7.37.3.25	<code>svm_get_labels(const svm_model *model, int *label)</code>	75
7.37.3.26	<code>svm_get_nr_class(const svm_model *model)</code>	75
7.37.3.27	<code>svm_get_nr_sv(const svm_model *model)</code>	75
7.37.3.28	<code>svm_get_sv_indices(const svm_model *model, int *indices)</code>	75
7.37.3.29	<code>svm_get_svm_type(const svm_model *model)</code>	75
7.37.3.30	<code>svm_get_svr_probability(const svm_model *model)</code>	75
7.37.3.31	<code>svm_group_classes(const svm_problem *prob, int *nr_class_ret, int **label_ret, int **start_ret, int **count_ret, int *perm)</code>	75
7.37.3.32	<code>svm_l_train(const struct svm_problem *prob, const struct svm_parameter *param)</code>	75
7.37.3.33	<code>svm_load_model(const char *model_file_name)</code>	75
7.37.3.34	<code>svm_model_visualization(const svm_model *model, Equation *equ)</code>	75

7.37.3.35	<code>svm_predict(const svm_model *model, const svm_node *x)</code>	75
7.37.3.36	<code>svm_predict_probability(const svm_model *model, const svm_node *x, double *prob_estimates)</code>	75
7.37.3.37	<code>svm_predict_values(const svm_model *model, const svm_node *x, double *dec_values)</code>	75
7.37.3.38	<code>svm_save_model(const char *model_file_name, const svm_model *model)</code>	75
7.37.3.39	<code>svm_set_print_string_function(void(*print_func)(const char *))</code>	75
7.37.3.40	<code>svm_svr_probability(const svm_problem *prob, const svm_parameter *param)</code>	75
7.37.3.41	<code>svm_train(const svm_problem *prob, const svm_parameter *param)</code>	75
7.37.3.42	<code>svm_train_one(const svm_problem *prob, const svm_parameter *param, double Cp, double Cn)</code>	75
7.37.3.43	<code>swap(T &x, T &y)</code>	75
7.37.4	Variable Documentation	75
7.37.4.1	<code>kernel_type_table</code>	75
7.37.4.2	<code>libsvm_version</code>	76
7.37.4.3	<code>line</code>	76
7.37.4.4	<code>max_line_len</code>	76
7.37.4.5	<code>negative_nodes</code>	76
7.37.4.6	<code>positive_nodes</code>	76
7.37.4.7	<code>svm_print_string</code>	76
7.37.4.8	<code>svm_type_table</code>	76
7.38	<code>src/svm_i.cpp</code> File Reference	76
7.38.1	Function Documentation	76
7.38.1.1	<code>operator<<(std::ostream &out, const SVM_I &svm_i)</code>	76
7.39	<code>test/1_conj.cpp</code> File Reference	76
7.39.1	Function Documentation	76
7.39.1.1	<code>conj(int *a)</code>	76
7.39.1.2	<code>main(int argc, char **argv)</code>	77
7.39.1.3	<code>nondet()</code>	77
7.40	<code>test/2_ex1.cpp</code> File Reference	77
7.40.1	Function Documentation	77
7.40.1.1	<code>ex1(int *a)</code>	77
7.40.1.2	<code>main(int argc, char **argv)</code>	77
7.40.1.3	<code>nondet()</code>	77
7.41	<code>test/2_f1.cpp</code> File Reference	77
7.41.1	Function Documentation	77
7.41.1.1	<code>f1(int *a)</code>	77
7.41.1.2	<code>main(int argc, char **argv)</code>	77
7.42	<code>test/2_f2.cpp</code> File Reference	77
7.42.1	Function Documentation	78
7.42.1.1	<code>f2(int *a)</code>	78

7.42.1.2	main(int argc, char **argv)	78
7.43	test/2_z3test.cpp File Reference	78
7.43.1	Function Documentation	78
7.43.1.1	main(int argc, char **argv)	78
7.44	test/3_f3.cpp File Reference	78
7.44.1	Function Documentation	78
7.44.1.1	f3(int *a)	78
7.44.1.2	main(int argc, char **argv)	78
Index		79

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

```
1 C:/Program Files
```
- **klee** This is optional currently.
- [Build tools](), such as make, Visual Studio 2015, or Xcode.

###Build InvariantInferenceFramework

```
1 git clone git@github.com:lijiaying/InvariantInferenceFramework.git
2 cd InvariantInferenceFramework
3 cd test
4 mkdir build
5 cd build
6 cmake .. -G [your platform] // just use cmake .. if you are not sure
7 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 foreground, 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 foreground, 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:

Cache	11
decision_function	11
Equation	12
IIF_learn	15
IIF_svm_i_learn	17
IIF_svm_learn	17
ML_Algo	20
Perceptron	24
SVM	35
SVM_I	39
QMatrix	27
Kernel	18
ONE_CLASS_Q	23
SVC_Q	34
SVR_Q	46
Solution	28
Solver::SolutionInfo	29
Solver	30
Solver_NU	32
States	33
svm_model	43
svm_node	44
svm_parameter	44
svm_problem	46

Chapter 4

Data Structure Index

4.1 Data Structures

Here are the data structures with brief descriptions:

Cache	11
decision_function	11
Equation	
This class defines an equation by storing all its coefficients. An equation is regarded a hyper-plane in math	12
IIF_learn	15
IIF_svm_i_learn	17
IIF_svm_learn	17
Kernel	18
ML_Algo	20
ONE_CLASS_Q	23
Perceptron	24
QMatrix	27
Solution	
This class defines the format of a valid solution to an equation	28
Solver::SolutionInfo	29
Solver	30
Solver_NU	32
States	33
SVC_Q	34
SVM	35
SVM_I	39
svm_model	43
svm_node	44
svm_parameter	44
svm_problem	46
SVR_Q	46

Chapter 5

File Index

5.1 File List

Here is a list of all files with brief descriptions:

build/CMakeFiles/2.8.12.2/CompilerIdC/CMakeCCompilerId.c	49
build/CMakeFiles/2.8.12.2/CompilerIdCXX/CMakeCXXCompilerId.cpp	50
include/color.h	
Provide support for colorful console output	52
include/config.h	
Provide most configuration information for the whole project	53
include/equation.h	
Defines the linear equation format and its solution format	55
include/iif.h	
Contains all the files that needed to be included by a new test	56
include/iif_assert.h	
Provide iif_assert and iif_assume support for system assume and assert macros	56
include/iif_learn.h	58
include/iif_svm_i_learn.h	59
include/iif_svm_learn.h	59
include/instrumentation.h	
Provide instrumentation function support for the framework	59
include/ml_algo.h	
Provide the base class for specific machine learning algorithm	61
include/perceptron.h	62
include/states.h	62
include/svm.h	62
include/svm_core.h	62
include/svm_i.h	65
src/color.cpp	65
src/config.cpp	65
src/equation.cpp	67
src/iif_svm_i_learn.cpp	68
src/iif_svm_learn.cpp	68
src/instrumentation.cpp	68
src/perceptron.cpp	70
src/states.cpp	71
src/svm.cpp	71
src/svm_core.cpp	71
src/svm_i.cpp	76
test/1_conj.cpp	76
test/2_ex1.cpp	77
test/2_f1.cpp	77

test/2_f2.cpp	77
test/2_z3test.cpp	78
test/3_f3.cpp	78

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 coefficients. 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 coefficients. An equation is regarded a hyperplane in math.

$$\text{theta0} + \text{theta}[0] * x_0 + \text{theta}[1] * x_1 + \dots + \text{theta}[\text{VARS}] * x_{\{\text{VARS}\}} \geq 0$$

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

<i>equ</i>	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

<i>equ</i>	is the given equation, should not be NULL
<i>sol</i>	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

<i>e2</i>	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

<i>precision</i>	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

<i>sol</i>	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 calculate the most informative points in space It return a points really on the margin or next to the margin

Parameters

<i>sol</i>	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 calculate 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

<i>rhs</i>	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 coefficients we want to make it elegant, which is the purpose of involving this method Currently we have not done much work on this We have not even use gcd function to adjust the coefficients.

For example. $1.2345 x_1 \geq 2.4690 \implies x_1 \geq 2$ $2 x_1 \geq 5.000001 \implies x_1 \geq 2.5$

Parameters

<i>e</i>	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: $2\{0\} + 3\{1\} \geq 5$

Parameters

<i>equ</i>	the equation to be output
------------	---------------------------

6.3.5 Field Documentation

6.3.5.1 double Equation::theta[VARs]

6.3.5.2 double Equation::theta0

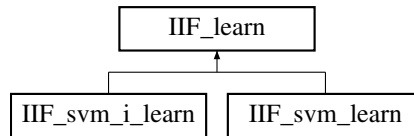
The documentation for this class was generated from the following files:

- [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]

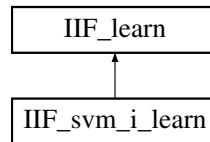
The documentation for this class was generated from the following file:

- [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]

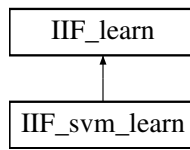
The documentation for this class was generated from the following files:

- [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]`

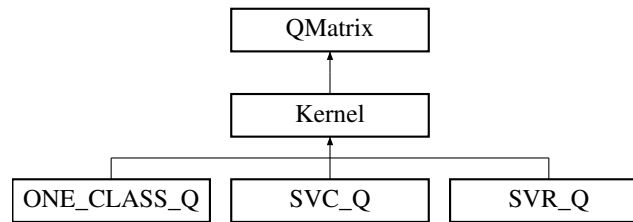
6.6.3.2 `SVM* IIF_svm_learn::svm` `[protected]`

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::k_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]`

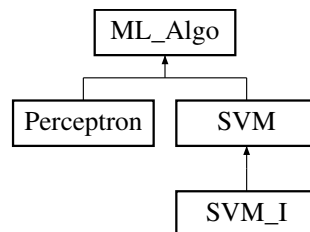
The documentation for this class was generated from the following file:

- [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 polymorphism 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](#) &m1a)
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 polymorphism 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 information if a question trace invalidate the training model

Parameters

<i>qset</i>	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

<i>previous_equations</i>	contains all the equation we get from last training session.
<i>equation_num</i>	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

<i>x</i>	contains the sample to be tested.
<i>flag</i>	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

<i>gsets</i>	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
<i>pre_positive_size</i>	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.
<i>pre_negative_size</i>	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

<i>equation_num</i>	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

<i>mla</i>	the ml_algo object to be output
------------	---------------------------------

Returns

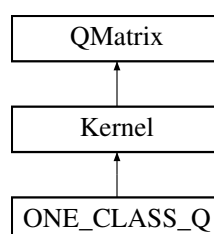
std::ostream

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](#) ¶m)

- [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](#).

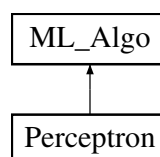
The documentation for this class was generated from the following file:

- [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

<i>qset</i>	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

<i>x</i>	contains the sample to be tested.
<i>flag</i>	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

<i>gsets</i>	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
<i>pre_positive_size</i>	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.
<i>pre_negative_size</i>	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

<i>equation_num</i>	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

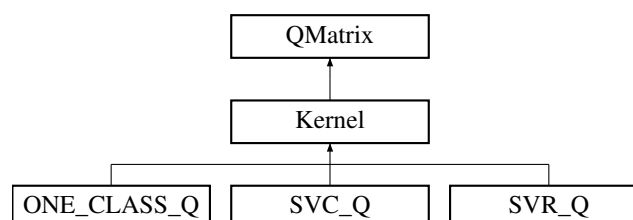
6.10.4.1 `int Perceptron::length`6.10.4.2 `Equation* Perceptron::main_equation`6.10.4.3 `double Perceptron::training_label[max_items *2]`6.10.4.4 `double* Perceptron::training_set[max_items *2]`

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

<i>a0...</i>	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

<i>sol</i>	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`

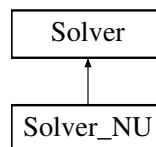
6.13.1.5 `double Solver::SolutionInfo::upper_bound_p`

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]

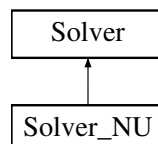
6.14.4.15 `schar* Solver::y` [protected]

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 [I](#), 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 [I](#), 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 stf][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`

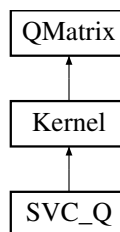
6.16.4.4 `double(* States::values)[VARs]`

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](#) ¶m, 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](#).

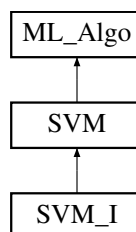
The documentation for this class was generated from the following file:

- [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 polymorphism 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 polymorphism 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 information 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

<i>previous_equations</i>	contains all the equation we get from last trainig session.
<i>equation_num</i>	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

<i>x</i>	contains the sample to be tested.
<i>flag</i>	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

<i>gsets</i>	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
<i>pre_positive_size</i>	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.
<i>pre_negative_size</i>	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

<i>equation_num</i>	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`

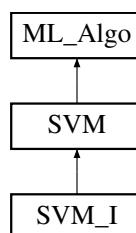
The documentation for this class was generated from the following files:

- [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 polymorphism 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 polymorphism 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 information if a question trace invalidate the training model

Parameters

<i>qset</i>	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

<i>previous_equations</i>	contains all the equation we get from last trainig session.
<i>equation_num</i>	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

<i>x</i>	contains the sample to be tested.
<i>flag</i>	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

<i>gsets</i>	The states array to store all the generated states information. The size must be 4, and index -1 should be accessible
<i>pre_positive_size</i>	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.
<i>pre_negative_size</i>	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

<i>equation_num</i>	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`

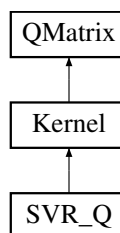
6.23.2.3 `double* svm_problem::y`

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` ¶m)
- 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 output.

```
#include <iostream>
```

Enumerations

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

This enumeration contains all the colors predefined 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 comparison.

7.12.1 Detailed Description

Provide support for colorful console output.

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 foreground, no strong comparison instead of the previous setting.

7.12.2 Enumeration Type Documentation

7.12.2.1 enum `color`

This enumeration contains all the colors predefined 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

<i>out</i>	The ostream to be changed, defines which stream you want to set
<i>color</i>	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 comparison.

7.13 include/config.h File Reference

Provide most configuration information for the whole project.

Macros

- #define **VARs** 2
defines the number of parameters 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 $\text{pow}(10, -\text{PRECISION})$ 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

<i>func</i>	The function to be tested It involves a small validation test on the given function.
<i>func_name</i>	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 coefficients. 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 foreground, no strong comparison 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

<code>expr</code>	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

<i>expr</i>	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

<i>void</i>	
-------------	--

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 struct 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 comparison.

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 comparison.

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

<i>func</i>	The function to be tested It involves a small validation test on the given function.
<i>fun_name</i>	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

<i>sol</i>	The solution object to be output
------------	----------------------------------

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

Example: $2\{0\} + 3\{1\} \geq 5$

Parameters

<i>equ</i>	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

<i>void</i>	
-------------	--

Returns

void

7.33.1.4 int [before_loop](#) ()

This function should be called each time before executing loop.

Parameters

<code>void</code>	
-------------------	--

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] = &It[1]**7.33.2.6 char It[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

- _passP
 - iif_assert.h, 58
 - instrumentation.cpp, 70
 - _passQ
 - iif_assert.h, 58
 - instrumentation.cpp, 70
 - _print
 - ML_Algo, 21
 - SVM_I, 40
 - SVM, 36
 - _roundoff
 - equation.cpp, 67
 - ~Cache
 - Cache, 11
 - ~Kernel
 - Kernel, 19
 - ~ONE_CLASS_Q
 - ONE_CLASS_Q, 24
 - ~Perceptron
 - Perceptron, 25
 - ~QMatrix
 - QMatrix, 28
 - ~SVM
 - SVM, 36
 - ~SVC_Q
 - SVC_Q, 35
 - ~SVM_I
 - SVM_I, 40
 - ~SVR_Q
 - SVR_Q, 47
 - ~Solver
 - Solver, 31
 - ~States
 - States, 33
 - 1_conj.cpp
 - conj, 76
 - main, 76
 - nondet, 77
 - 2_ex1.cpp
 - ex1, 77
 - main, 77
 - nondet, 77
 - 2_f1.cpp
 - f1, 77
 - main, 77
 - 2_f2.cpp
 - f2, 78
 - main, 78
 - 2_z3test.cpp

- main, 78
- 3_f3.cpp
 - f3, 78
 - main, 78
- ARCHITECTURE_ID
 - CMakeCCompilerId.c, 49
 - CMakeCXXCompilerId.cpp, 50
- active_set
 - Solver, 32
- active_size
 - Solver, 32
- add_state_double
 - instrumentation.cpp, 69
 - instrumentation.h, 60
- add_state_int
 - instrumentation.cpp, 69
 - instrumentation.h, 60
- add_states
 - States, 33
- after_exes
 - config.h, 54
- after_loop
 - instrumentation.cpp, 69
 - instrumentation.h, 60
- alpha
 - decision_function, 11
 - Solver, 32
- alpha_status
 - Solver, 32
- assert_times
 - config.cpp, 66
 - iif_assert.h, 58
 - instrumentation.cpp, 70
- assume_times
 - config.cpp, 66
 - iif_assert.h, 58
 - instrumentation.cpp, 70
- BLUE
 - color.h, 52
- before_loop
 - instrumentation.cpp, 69
 - instrumentation.h, 61
- build/CMakeCache.txt, 49
- build/CMakeFiles/2.8.12.2/CompilerIdC/CMakeC↵
CompilerId.c, 49
- build/CMakeFiles/2.8.12.2/CompilerIdCXX/CMakeCX↵
XCompilerId.cpp, 50
- build/CMakeFiles/TargetDirectories.txt, 51

- build/CMakeFiles/conj.dir/link.txt, 51
- build/CMakeFiles/ex1.dir/link.txt, 51
- build/CMakeFiles/f1.dir/link.txt, 51
- build/CMakeFiles/f2.dir/link.txt, 51
- build/CMakeFiles/f3.dir/link.txt, 51
- build/CMakeFiles/z3test.dir/link.txt, 51
- C
 - svm_parameter, 45
- C_SVC
 - svm_core.h, 63
- CMakeCCompilerId.c
 - ARCHITECTURE_ID, 49
 - COMPILER_ID, 49
 - DEC, 49
 - HEX, 49
 - info_arch, 50
 - info_compiler, 50
 - info_platform, 50
 - main, 50
 - PLATFORM_ID, 50
- CMakeCXXCompilerId.cpp
 - ARCHITECTURE_ID, 50
 - COMPILER_ID, 50
 - DEC, 50
 - HEX, 51
 - info_arch, 51
 - info_compiler, 51
 - info_platform, 51
 - main, 51
 - PLATFORM_ID, 51
- CMakeLists.txt, 52
- COMPILER_ID
 - CMakeCCompilerId.c, 49
 - CMakeCXXCompilerId.cpp, 50
- COUNTER_EXAMPLE
 - instrumentation.h, 60
- Cache, 11
 - ~Cache, 11
 - Cache, 11
 - get_data, 11
 - swap_index, 11
- cache_size
 - svm_parameter, 45
- calc
 - Equation, 13
- calculate_rho
 - Solver, 31
- check_question_set
 - ML_Algo, 21
 - Perceptron, 25
 - SVM_I, 40
 - SVM, 36
- check_target_program
 - config.cpp, 66
- clone
 - svm_core.cpp, 74
- Cn
 - Solver, 32
- coef0
 - svm_parameter, 45
- color
 - color.h, 52
- color.cpp
 - unset_console_color, 65
- color.h
 - BLUE, 52
 - color, 52
 - GREEN, 52
 - RED, 52
 - set_console_color, 53
 - unset_console_color, 53
 - WHITE, 52
 - YELLOW, 52
- config.cpp
 - assert_times, 66
 - assume_times, 66
 - check_target_program, 66
 - maxv, 66
 - minv, 66
 - register_program, 66
 - target_program, 66
- config.h
 - after_exes, 54
 - init_exes, 54
 - max_items, 54
 - max_iter, 54
 - PRECISION, 54
 - random_exes, 55
 - register_program, 54
 - target_program, 55
 - VARS, 54
- conj
 - 1_conj.cpp, 76
- Cp
 - Solver, 32
- DEC
 - CMakeCCompilerId.c, 49
 - CMakeCXXCompilerId.cpp, 50
- decision_function, 11
 - alpha, 11
 - rho, 11
- degree
 - svm_parameter, 45
- do_shrinking
 - Solver, 31
- EPSILON_SVR
 - svm_core.h, 64
- eps
 - Solver, 32
 - svm_parameter, 45
- equ_num
 - SVM_I, 43
- Equation, 12
 - calc, 13
 - Equation, 13

- imply, [13](#)
 - is_similar, [14](#)
 - linear_solver, [14](#)
 - operator<<, [15](#)
 - operator=, [14](#)
 - roundoff, [15](#)
 - theta, [15](#)
 - theta0, [15](#)
- equation.cpp
 - _roundoff, [67](#)
 - operator<<, [67](#)
 - UPBOUND, [67](#)
- equation.h
 - maxv, [56](#)
 - minv, [56](#)
- equations
 - SVM_I, [43](#)
- ex1
 - 2_ex1.cpp, [77](#)
- f1
 - 2_f1.cpp, [77](#)
- f2
 - 2_f2.cpp, [78](#)
- f3
 - 3_f3.cpp, [78](#)
- FREE
 - Solver, [31](#)
- FSCANF
 - svm_core.cpp, [73](#)
- free_sv
 - svm_model, [43](#)
- func
 - IIF_learn, [16](#)
- G
 - Solver, [32](#)
- G_bar
 - Solver, [32](#)
- GREEN
 - color.h, [52](#)
- gamma
 - svm_parameter, [45](#)
- get_QD
 - Kernel, [19](#)
 - ONE_CLASS_Q, [24](#)
 - QMatrix, [28](#)
 - SVC_Q, [35](#)
 - SVR_Q, [47](#)
- get_C
 - Solver, [31](#)
- get_converged
 - ML_Algo, [21](#)
 - SVM_I, [41](#)
 - SVM, [37](#)
- get_data
 - Cache, [11](#)
- get_Q
 - Kernel, [19](#)
 - ONE_CLASS_Q, [24](#)
 - QMatrix, [28](#)
 - SVC_Q, [35](#)
 - SVR_Q, [47](#)
- gsets
 - IIF_learn, [16](#)
- HEX
 - CMakeCCompilerId.c, [49](#)
 - CMakeCXXCompilerId.cpp, [51](#)
- IIF_learn, [15](#)
 - func, [16](#)
 - gsets, [16](#)
 - IIF_learn, [16](#)
 - init_gsets, [16](#)
 - learn, [16](#)
 - run_target, [16](#)
- IIF_svm_i_learn, [17](#)
 - IIF_svm_i_learn, [17](#)
 - learn, [17](#)
 - max_iteration, [17](#)
 - svm_i, [17](#)
- IIF_svm_learn, [17](#)
 - IIF_svm_learn, [18](#)
 - learn, [18](#)
 - max_iteration, [18](#)
 - svm, [18](#)
- INF
 - svm_core.cpp, [73](#)
- iif.h
 - maxv, [56](#)
 - minv, [56](#)
- iif_assert
 - iif_assert.h, [57](#)
- iif_assert.h
 - _passP, [58](#)
 - _passQ, [58](#)
 - assert_times, [58](#)
 - assume_times, [58](#)
 - iif_assert, [57](#)
 - iif_assume, [57](#)
- iif_assume
 - iif_assert.h, [57](#)
- iif_svm_i_learn.cpp
 - print_null, [68](#)
- iif_svm_learn.cpp
 - print_null, [68](#)
- imply
 - Equation, [13](#)
- include/color.h, [52](#)
- include/config.h, [53](#)
- include/equation.h, [55](#)
- include/iif.h, [56](#)
- include/iif_assert.h, [56](#)
- include/iif_learn.h, [58](#)
- include/iif_svm_i_learn.h, [59](#)
- include/iif_svm_learn.h, [59](#)
- include/instrumentation.h, [59](#)

- include/ml_algo.h, 61
- include/perceptron.h, 62
- include/states.h, 62
- include/svm.h, 62
- include/svm_core.h, 62
- include/svm_i.h, 65
- index
 - States, 34
- info
 - svm_core.cpp, 74
- info_arch
 - CMakeCCompilerId.c, 50
 - CMakeCXXCompilerId.cpp, 51
- info_compiler
 - CMakeCCompilerId.c, 50
 - CMakeCXXCompilerId.cpp, 51
- info_platform
 - CMakeCCompilerId.c, 50
 - CMakeCXXCompilerId.cpp, 51
- init_exes
 - config.h, 54
- init_gsets
 - IIF_learn, 16
- instrumentation.cpp
 - _passP, 70
 - _passQ, 70
 - add_state_double, 69
 - add_state_int, 69
 - after_loop, 69
 - assert_times, 70
 - assume_times, 70
 - before_loop, 69
 - LabelTable, 70
 - It, 70
 - m_double, 70
 - m_int, 70
 - temp_index, 70
 - temp_states, 70
- instrumentation.h
 - add_state_double, 60
 - add_state_int, 60
 - after_loop, 60
 - before_loop, 61
 - COUNTER_EXAMPLE, 60
 - m_double, 61
 - m_int, 61
 - NEGATIVE, 60
 - POSITIVE, 60
 - QUESTION, 60
 - trace_type, 60
- is_free
 - Solver, 31
- is_lower_bound
 - Solver, 31
- is_similar
 - Equation, 14
- is_upper_bound
 - Solver, 31
- k_function
 - Kernel, 19
- Kernel, 18
 - ~Kernel, 19
 - get_QD, 19
 - get_Q, 19
 - k_function, 19
 - Kernel, 19
 - kernel_function, 20
 - swap_index, 19
- kernel_function
 - Kernel, 20
- kernel_type
 - svm_parameter, 45
- kernel_type_table
 - svm_core.cpp, 75
- I
 - Solver, 32
 - svm_model, 43
 - svm_problem, 46
- LIBSVM_VERSION
 - svm_core.h, 63
- LINEAR
 - svm_core.h, 64
- LOWER_BOUND
 - Solver, 31
- label
 - States, 34
 - svm_model, 43
- LabelTable
 - instrumentation.cpp, 70
- learn
 - IIF_learn, 16
 - IIF_svm_i_learn, 17
 - IIF_svm_learn, 18
- length
 - Perceptron, 27
- libsvm_version
 - svm_core.cpp, 75
 - svm_core.h, 65
- line
 - svm_core.cpp, 76
- linear_solver
 - Equation, 14
- It
 - instrumentation.cpp, 70
- m_double
 - instrumentation.cpp, 70
 - instrumentation.h, 61
- m_int
 - instrumentation.cpp, 70
 - instrumentation.h, 61
- ML_Algo, 20
 - _print, 21
 - check_question_set, 21
 - get_converged, 21
 - ML_Algo, 21

- operator<<, 23
- predict, 21
- predict_on_training_set, 22
- prepare_training_data, 22
- roundoff, 22
- size, 22
- train, 23
- main
 - 1_conj.cpp, 76
 - 2_ex1.cpp, 77
 - 2_f1.cpp, 77
 - 2_f2.cpp, 78
 - 2_z3test.cpp, 78
 - 3_f3.cpp, 78
 - CMakeCCompilerId.c, 50
 - CMakeCXXCompilerId.cpp, 51
- main_equation
 - Perceptron, 27
 - SVM, 39
- Malloc
 - svm_core.cpp, 73
- max
 - svm_core.cpp, 74
- max_equ
 - SVM_I, 43
- max_items
 - config.h, 54
- max_iter
 - config.h, 54
- max_iteration
 - IIF_svm_i_learn, 17
 - IIF_svm_learn, 18
- max_line_len
 - svm_core.cpp, 76
- max_size
 - SVM, 39
- maxv
 - config.cpp, 66
 - equation.h, 56
 - iif.h, 56
- min
 - svm_core.cpp, 74
- minv
 - config.cpp, 66
 - equation.h, 56
 - iif.h, 56
- model
 - SVM_I, 43
 - SVM, 39
- multiclass_probability
 - svm_core.cpp, 74
- NEGATIVE
 - instrumentation.h, 60
- nSV
 - svm_model, 43
- NU_SVC
 - svm_core.h, 63
- NU_SVR
 - svm_core.h, 64
- negative_nodes
 - svm_core.cpp, 76
- negatives
 - SVM_I, 43
- nondet
 - 1_conj.cpp, 77
 - 2_ex1.cpp, 77
- nr_class
 - svm_model, 43
- nr_weight
 - svm_parameter, 45
- nu
 - svm_parameter, 45
- ONE_CLASS_Q, 23
 - ~ONE_CLASS_Q, 24
 - get_QD, 24
 - get_Q, 24
 - ONE_CLASS_Q, 24
 - swap_index, 24
- ONE_CLASS
 - svm_core.h, 64
- obj
 - Solver::SolutionInfo, 30
- operator<<
 - Equation, 15
 - equation.cpp, 67
 - ML_Algo, 23
 - Perceptron, 27
 - perceptron.cpp, 71
 - SVM_I, 43
 - SVM, 39
 - Solution, 29
 - States, 34
 - states.cpp, 71
 - svm.cpp, 71
 - svm_i.cpp, 76
 - svm_node, 44
 - svm_problem, 46
- operator=
 - Equation, 14
- p
 - Solver, 32
 - svm_parameter, 45
- p_index
 - States, 34
- PLATFORM_ID
 - CMakeCCompilerId.c, 50
 - CMakeCXXCompilerId.cpp, 51
- POLY
 - svm_core.h, 64
- POSITIVE
 - instrumentation.h, 60
- PRECISION
 - config.h, 54
- PRECOMPUTED
 - svm_core.h, 64

- param
 - SVM_I, 43
 - SVM, 39
 - svm_model, 44
- Perceptron, 24
 - ~Perceptron, 25
 - check_question_set, 25
 - length, 27
 - main_equation, 27
 - operator<<, 27
 - Perceptron, 25
 - predict, 25
 - predict_on_training_set, 26
 - prepare_training_data, 26
 - roundoff, 26
 - size, 26
 - train, 27
 - training_label, 27
 - training_set, 27
- perceptron.cpp
 - operator<<, 71
- positive_nodes
 - svm_core.cpp, 76
- powi
 - svm_core.cpp, 74
- predict
 - ML_Algo, 21
 - Perceptron, 25
 - SVM_I, 41
 - SVM, 37
- predict_on_training_set
 - ML_Algo, 22
 - Perceptron, 26
 - SVM_I, 41
 - SVM, 37
- prepare_training_data
 - ML_Algo, 22
 - Perceptron, 26
 - SVM_I, 41
 - SVM, 37
- print_null
 - iif_svm_i_learn.cpp, 68
 - iif_svm_learn.cpp, 68
- print_string_stdout
 - svm_core.cpp, 74
- print_svm_samples
 - svm_core.cpp, 74
 - svm_core.h, 64
- print_trace
 - States, 34
- probA
 - svm_model, 44
- probability
 - svm_parameter, 45
- probB
 - svm_model, 44
- problem
 - SVM, 39
- Q
 - Solver, 32
- QMatrix, 27
 - ~QMatrix, 28
 - get_QD, 28
 - get_Q, 28
 - swap_index, 28
- QUESTION
 - instrumentation.h, 60
- QD
 - Solver, 32
- Qfloat
 - svm_core.cpp, 73
- r
 - Solver::SolutionInfo, 30
- RBF
 - svm_core.h, 64
- README.md, 65
- RED
 - color.h, 52
- random_exes
 - config.h, 55
- read_model_header
 - svm_core.cpp, 74
- readline
 - svm_core.cpp, 74
- reconstruct_gradient
 - Solver, 31
- register_program
 - config.cpp, 66
 - config.h, 54
- rho
 - decision_function, 11
 - Solver::SolutionInfo, 30
 - svm_model, 44
- roundoff
 - Equation, 15
 - ML_Algo, 22
 - Perceptron, 26
 - SVM_I, 42
 - SVM, 38
- run_target
 - IIF_learn, 16
- SIGMOID
 - svm_core.h, 64
- SVC_Q, 34
 - ~SVC_Q, 35
 - get_QD, 35
 - get_Q, 35
 - SVC_Q, 34
 - swap_index, 35
- SVM_I, 39
 - _print, 40
 - ~SVM_I, 40
 - check_question_set, 40
 - equ_num, 43
 - equations, 43

- get_converged, 41
- max_equ, 43
- model, 43
- negatives, 43
- operator<<, 43
- param, 43
- predict, 41
- predict_on_training_set, 41
- prepare_training_data, 41
- roundoff, 42
- SVM_I, 40
- size, 42
- train, 42
- SVR_Q, 46
 - ~SVR_Q, 47
 - get_QD, 47
 - get_Q, 47
 - SVR_Q, 47
 - swap_index, 47
- SVM, 35
 - _print, 36
 - ~SVM, 36
 - check_question_set, 36
 - get_converged, 37
 - main_equation, 39
 - max_size, 39
 - model, 39
 - operator<<, 39
 - param, 39
 - predict, 37
 - predict_on_training_set, 37
 - prepare_training_data, 37
 - problem, 39
 - roundoff, 38
 - SVM, 36
 - size, 38
 - train, 38
 - training_label, 39
 - training_set, 39
- schar
 - svm_core.cpp, 74
- select_working_set
 - Solver, 31
- set_console_color
 - color.h, 53
- shrinking
 - svm_parameter, 45
- sigmoid_predict
 - svm_core.cpp, 74
- sigmoid_train
 - svm_core.cpp, 74
- size
 - ML_Algo, 22
 - Perceptron, 26
 - SVM_I, 42
 - SVM, 38
 - States, 34
- Solution, 28
 - operator<<, 29
 - Solution, 29
 - x, 29
- Solve
 - Solver, 31
 - Solver_NU, 33
- solve_c_svc
 - svm_core.cpp, 74
- solve_epsilon_svr
 - svm_core.cpp, 74
- solve_nu_svc
 - svm_core.cpp, 74
- solve_nu_svr
 - svm_core.cpp, 74
- solve_one_class
 - svm_core.cpp, 74
- Solver, 30
 - ~Solver, 31
 - active_set, 32
 - active_size, 32
 - alpha, 32
 - alpha_status, 32
 - calculate_rho, 31
 - Cn, 32
 - Cp, 32
 - do_shrinking, 31
 - eps, 32
 - FREE, 31
 - G, 32
 - G_bar, 32
 - get_C, 31
 - is_free, 31
 - is_lower_bound, 31
 - is_upper_bound, 31
 - I, 32
 - LOWER_BOUND, 31
 - p, 32
 - Q, 32
 - QD, 32
 - reconstruct_gradient, 31
 - select_working_set, 31
 - Solve, 31
 - Solver, 31
 - swap_index, 31
 - UPPER_BOUND, 31
 - unshrink, 32
 - update_alpha_status, 32
 - y, 32
- Solver::SolutionInfo, 29
 - obj, 30
 - r, 30
 - rho, 30
 - upper_bound_n, 30
 - upper_bound_p, 30
- Solver_NU, 32
 - Solve, 33
 - Solver_NU, 33
- src/color.cpp, 65

- src/config.cpp, 65
- src/equation.cpp, 67
- src/iif_svm_i_learn.cpp, 68
- src/iif_svm_learn.cpp, 68
- src/instrumentation.cpp, 68
- src/perceptron.cpp, 70
- src/states.cpp, 71
- src/svm.cpp, 71
- src/svm_core.cpp, 71
- src/svm_i.cpp, 76
- States, 33
 - ~States, 33
 - add_states, 33
 - index, 34
 - label, 34
 - operator<<, 34
 - p_index, 34
 - print_trace, 34
 - size, 34
 - States, 33
 - traces_num, 34
 - values, 34
- states.cpp
 - operator<<, 71
- SV
 - svm_model, 44
- sv_coef
 - svm_model, 44
- sv_indices
 - svm_model, 44
- svm
 - IIF_svm_learn, 18
- svm.cpp
 - operator<<, 71
- svm_l_train
 - svm_core.cpp, 75
 - svm_core.h, 64
- svm_binary_svc_probability
 - svm_core.cpp, 74
- svm_check_parameter
 - svm_core.cpp, 74
 - svm_core.h, 64
- svm_check_probability_model
 - svm_core.cpp, 74
 - svm_core.h, 64
- svm_core.cpp
 - clone, 74
 - FSCANF, 73
 - INF, 73
 - info, 74
 - kernel_type_table, 75
 - libsvm_version, 75
 - line, 76
 - Malloc, 73
 - max, 74
 - max_line_len, 76
 - min, 74
 - multiclass_probability, 74
 - negative_nodes, 76
 - positive_nodes, 76
 - powi, 74
 - print_string_stdout, 74
 - print_svm_samples, 74
 - Qfloat, 73
 - read_model_header, 74
 - readline, 74
 - schar, 74
 - sigmoid_predict, 74
 - sigmoid_train, 74
 - solve_c_svc, 74
 - solve_epsilon_svr, 74
 - solve_nu_svc, 74
 - solve_nu_svr, 74
 - solve_one_class, 74
 - svm_l_train, 75
 - svm_binary_svc_probability, 74
 - svm_check_parameter, 74
 - svm_check_probability_model, 74
 - svm_cross_validation, 74
 - svm_destroy_param, 74
 - svm_free_and_destroy_model, 75
 - svm_free_model_content, 75
 - svm_get_labels, 75
 - svm_get_nr_class, 75
 - svm_get_nr_sv, 75
 - svm_get_sv_indices, 75
 - svm_get_svm_type, 75
 - svm_get_svr_probability, 75
 - svm_group_classes, 75
 - svm_load_model, 75
 - svm_model_visualization, 75
 - svm_predict, 75
 - svm_predict_probability, 75
 - svm_predict_values, 75
 - svm_print_string, 76
 - svm_save_model, 75
 - svm_set_print_string_function, 75
 - svm_svr_probability, 75
 - svm_train, 75
 - svm_train_one, 75
 - svm_type_table, 76
 - swap, 75
 - TAU, 73
- svm_core.h
 - C_SVC, 63
 - EPSILON_SVR, 64
 - LIBSVM_VERSION, 63
 - LINEAR, 64
 - libsvm_version, 65
 - NU_SVC, 63
 - NU_SVR, 64
 - ONE_CLASS, 64
 - POLY, 64
 - PRECOMPUTED, 64
 - print_svm_samples, 64
 - RBF, 64

- SIGMOID, [64](#)
- svm_l_train, [64](#)
- svm_check_parameter, [64](#)
- svm_check_probability_model, [64](#)
- svm_cross_validation, [64](#)
- svm_destroy_param, [64](#)
- svm_free_and_destroy_model, [64](#)
- svm_free_model_content, [64](#)
- svm_get_labels, [64](#)
- svm_get_nr_class, [64](#)
- svm_get_nr_sv, [64](#)
- svm_get_sv_indices, [64](#)
- svm_get_svm_type, [64](#)
- svm_get_svr_probability, [64](#)
- svm_load_model, [64](#)
- svm_model_visualization, [64](#)
- svm_predict, [64](#)
- svm_predict_probability, [64](#)
- svm_predict_values, [64](#)
- svm_save_model, [65](#)
- svm_set_print_string_function, [65](#)
- svm_train, [65](#)
- svm_cross_validation
 - svm_core.cpp, [74](#)
 - svm_core.h, [64](#)
- svm_destroy_param
 - svm_core.cpp, [74](#)
 - svm_core.h, [64](#)
- svm_free_and_destroy_model
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_free_model_content
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_labels
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_nr_class
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_nr_sv
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_sv_indices
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_svm_type
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_get_svr_probability
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_group_classes
 - svm_core.cpp, [75](#)
- svm_i
 - libsvm_i_learn, [17](#)
- svm_i.cpp
 - operator<<, [76](#)
- svm_load_model
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_model, [43](#)
 - free_sv, [43](#)
 - l, [43](#)
 - label, [43](#)
 - nSV, [43](#)
 - nr_class, [43](#)
 - param, [44](#)
 - probA, [44](#)
 - probB, [44](#)
 - rho, [44](#)
 - SV, [44](#)
 - sv_coef, [44](#)
 - sv_indices, [44](#)
- svm_model_visualization
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_node, [44](#)
 - operator<<, [44](#)
 - value, [44](#)
- svm_parameter, [44](#)
 - C, [45](#)
 - cache_size, [45](#)
 - coef0, [45](#)
 - degree, [45](#)
 - eps, [45](#)
 - gamma, [45](#)
 - kernel_type, [45](#)
 - nr_weight, [45](#)
 - nu, [45](#)
 - p, [45](#)
 - probability, [45](#)
 - shrinking, [45](#)
 - svm_type, [45](#)
 - weight, [45](#)
 - weight_label, [45](#)
- svm_predict
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_predict_probability
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_predict_values
 - svm_core.cpp, [75](#)
 - svm_core.h, [64](#)
- svm_print_string
 - svm_core.cpp, [76](#)
- svm_problem, [46](#)
 - l, [46](#)
 - operator<<, [46](#)
 - x, [46](#)
 - y, [46](#)
- svm_save_model
 - svm_core.cpp, [75](#)
 - svm_core.h, [65](#)
- svm_set_print_string_function

- svm_core.cpp, 75
 - svm_core.h, 65
- svm_svr_probability
 - svm_core.cpp, 75
- svm_train
 - svm_core.cpp, 75
 - svm_core.h, 65
- svm_train_one
 - svm_core.cpp, 75
- svm_type
 - svm_parameter, 45
- svm_type_table
 - svm_core.cpp, 76
- swap
 - svm_core.cpp, 75
- swap_index
 - Cache, 11
 - Kernel, 19
 - ONE_CLASS_Q, 24
 - QMatrix, 28
 - SVC_Q, 35
 - SVR_Q, 47
 - Solver, 31
- TAU
 - svm_core.cpp, 73
- target_program
 - config.cpp, 66
 - config.h, 55
- temp_index
 - instrumentation.cpp, 70
- temp_states
 - instrumentation.cpp, 70
- test/1_conj.cpp, 76
- test/2_ex1.cpp, 77
- test/2_f1.cpp, 77
- test/2_f2.cpp, 77
- test/2_z3test.cpp, 78
- test/3_f3.cpp, 78
- theta
 - Equation, 15
- theta0
 - Equation, 15
- trace_type
 - instrumentation.h, 60
- traces_num
 - States, 34
- train
 - ML_Algo, 23
 - Perceptron, 27
 - SVM_I, 42
 - SVM, 38
- training_label
 - Perceptron, 27
 - SVM, 39
- training_set
 - Perceptron, 27
 - SVM, 39
- UPBOUND
 - equation.cpp, 67
- UPPER_BOUND
 - Solver, 31
- unset_console_color
 - color.cpp, 65
 - color.h, 53
- unshrink
 - Solver, 32
- update_alpha_status
 - Solver, 32
- upper_bound_n
 - Solver::SolutionInfo, 30
- upper_bound_p
 - Solver::SolutionInfo, 30
- VARS
 - config.h, 54
- value
 - svm_node, 44
- values
 - States, 34
- WHITE
 - color.h, 52
- weight
 - svm_parameter, 45
- weight_label
 - svm_parameter, 45
- x
 - Solution, 29
 - svm_problem, 46
- y
 - Solver, 32
 - svm_problem, 46
- YELLOW
 - color.h, 52