```c
#ifndef _LIBSVM_H
#define _LIBSVM_H

#define LIBSVM_VERSION 317

#ifdef __cplusplus
extern "C" {
#endif

extern int libsvm_version;

struct svm_node
{
        int index;
        double value;
};

struct svm_problem
{
        int l;
        double *y;
        struct svm_node **x;
};

enum { C_SVC, NU_SVC, ONE_CLASS, EPSILON_SVR, NU_SVR }; /* svm_type */
enum { LINEAR, POLY, RBF, SIGMOID, PRECOMPUTED }; /* kernel_type */

struct svm_parameter
{
        int svm_type;
        int kernel_type;
        int degree;     /* for poly */
        double gamma;   /* for poly/rbf/sigmoid */
        double coef0;   /* for poly/sigmoid */

        /* these are for training only */
        double cache_size; /* in MB */
        double eps;     /* stopping criteria */
        double C;       /* for C_SVC, EPSILON_SVR and NU_SVR */
        int nr_weight;          /* for C_SVC */
        int *weight_label;      /* for C_SVC */
        double* weight;         /* for C_SVC */
        double nu;      /* for NU_SVC, ONE_CLASS, and NU_SVR */
        double p;       /* for EPSILON_SVR */
        int shrinking;  /* use the shrinking heuristics */
        int probability; /* do probability estimates */
};

//
// svm_model
//
struct svm_model
{
        struct svm_parameter param;     /* parameter */
        int nr_class;           /* number of classes, = 2 in regression/one class svm */
        int l;                  /* total #SV */
        struct svm_node **SV;           /* SVs (SV[l]) */
        double **sv_coef;       /* coefficients for SVs in decision functions (sv_coef[k-1]
[l]) */
        double *rho;            /* constants in decision functions (rho[k*(k-1)/2]) */
        double *probA;          /* pariwise probability information */
        double *probB;
        int *sv_indices;        /* sv_indices[0,...,nSV-1] are values in [1,...,num_traning
_data] to indicate SVs in the training set */

        /* for classification only */

        int *label;             /* label of each class (label[k]) */
        int *nSV;               /* number of SVs for each class (nSV[k]) */
                                /* nSV[0] + nSV[1] + ... + nSV[k-1] = l */
        /* XXX */
        int free_sv;            /* 1 if svm_model is created by svm_load_model*/
                                /* 0 if svm_model is created by svm_train */
};

struct svm_model *svm_train(const struct svm_problem *prob, const struct svm_parameter *par
am);
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, dou
ble* 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 *));

#ifdef __cplusplus
}
#endif

#endif /* _LIBSVM_H */
```