REFERENCES


[29] LibSVM “README” tutorial

[30] Hsu., Chang, C-W., Lin, C. C., A practical Guide to Support Vector Classification, Department of Computer Science, National Taiwan University.

[31] Qiong Li., Applying support vector machine to predict hourly cooling load in the building, 2009, 86, 2249-2256.


[34] Hour, Z., Lian, Z., An Application of support vector machines in cooling load prediction, School of Mechanical and Electrical Engineering, Shenzhen Polytechnic and Institute of Refrigeration and Cryogenics Shanhai Jiao Tong University, 2009.


[44] MATLAB 6.5 user’s manual. The Math Works, Inc


APPENDIX A: AN OVERVIEW OF STATISTICAL LEARNING THEORY

University of Moratuwa, Sri Lanka. Electronic Theses & Dissertations www.lib.mrt.ac.lk
APPENDIX A: AN OVERVIEW OF STATISTICAL LEARNING THEORY

Vapnik (1995) presents all the machine learning problems as the following:
Given a set of data points \(((x_1,y_1),(x_2,y_2),\ldots,(x_l,y_l))\) \((x_i\in \mathbb{X} \subset \mathcal{R}^2, y_i \in \mathbb{Y} \subset \mathcal{R}, l\) is the number of data points, for regression estimation and density estimation and \(y_i \in \mathbb{Y} \subset \mathcal{N}\) for pattern recognition) randomly and independently generated from an unknown probability distribution \(p(x,y)\), find a function \(f(x,a)\) that has the minimal risk function (A.1).

\[
R(f) = \int_{x,y} L(y, f(x,a))p(x,y)dx\,dy
\]  
(A.1)

Where \(a\) is the parameter of \(f(x,a)\). \(R(f)\) is called the generalization error or the expected test error. It is a measure of the generalization performance of \(f(x,a)\). \(L(y, f(x,a))\) is called the loss function. It is a measure of the deviations between the actual values and the estimated values on the data points generated from \(p(x,y)\).

As \(p(x,y)\) is unknown, traditional methods attempt to estimate \(f(x,a)\) by minimizing the empirical risk function:

\[
R_{emp}(f) = \frac{1}{l} \sum_{i=1}^{l} L(x,a))
\]  
(A.2)

\(R_{emp}(f)\) is called the empirical error. That is, is estimated by training samples. Empirical Risk Minimization (ERM) principle is to minimize the generalization error by minimizing the empirical error \(R_{emp}(f)\). Traditional neural networks utilize this principle.

However, because of the limited number of \(l\), sometimes \(R_{emp}(f)\) cannot estimate \(R(f)\) well. As described in the statistical learning theory, and have the following relationship:

\[
R(f) \leq R_{emp}(f) + \Omega(\frac{1}{l})
\]  
(A.3)
Where \( h \) is called the Vapnik-Chervonenkis (VC) dimension. It is a measure of the capacity of \( f(x,a) \), which means that the ability of \( f(x,a) \) to learn any training data point without error. \( \Omega \left( \frac{1}{h} \right) \) is called the confidence interval, a decreasing function of \( \frac{1}{h} \), which is the ration of the number of training samples into the VC dimension of the estimator. Equation (A.3) shows that the value of \( R(f) \) depends both on \( R_{emp}(f) \) and \( \Omega \left( \frac{1}{h} \right) \). Hence, \( R_{emp}(f) \) can accurately estimate \( R(f) \) only when \( \Omega \left( \frac{1}{h} \right) \) is small enough. Because of this, Vapnik developed the Structural Risk Minimization (SRM) principle. The SRM principle is: one defines a nested structure, \( S_1 \subset S_2 \subset \ldots \subset S_m \ldots \), as shown in Appendix B.1, on the set of functions \( S = \{ f(\cdot, a), a \in \Lambda \} \) with their VC-dimensions satisfying \( h_1 \subset h_2 \subset \ldots \subset h_m \ldots \), and then chooses the structure element \( S_k \) with the minimal upper bound of the generalization error \( R(f) \).

Appendix A.1: A structure on the set of functions is determined by the nested subsets of functions.

The objective of SRM principle is to estimate \( f(x,a) \) by minimizing both the empirical error \( R_{emp}(f) \) and the confidence interval \( \Omega \left( \frac{1}{h} \right) \), as shown in Appendix A.2. It defines a trade-off between the quality of the approximation of the given data and the complexity of the approximating function.
Appendix A.2: The bound on the risk is the sum of the empirical risk and of the confidence interval. The smallest bound of the risk is achieved on some appropriate element of the structure (Source: Vapnik, 1995)
APPENDIX B: AN INTRODUCTION TO LIBSVM 2.6 PROGRAM
APPENDIX B: AN INTRODUCTION TO LIBSVM 2.6 PROGRAM

Libsvm is a simple, easy-to-use, and efficient software for SVM classification and regression. Libsvm 2.6 was developed by Chih-Chung Chang and Chih-Jen Lin in 2001. It can solve C-SVM classification, nu-SVM classification, one-class-SVM, epsilon-SVM regression, and nu-SVM regression. It also provides an automatic model selection tool for C-SVM classification.

Libsvm 2.6 is available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

The format of training and testing data file is:
<label><index1>:<value1><index2>:<value2> ...</label>

<label> is the target value of the training data. For classification, it should be an integer which identifies a class (multi-class classification is supported). For regression, it's any real number. For one-class SVM, it's not used so can be any number. <index> is an integer starting from 1, <value> is a real number. The labels in the testing data file are only used to calculate accuracy or error. If they are unknown, just fill this column with a number.

‘svm-train’ Usage

options:
-s svm_type : set type of SVM (default 0)
0 -- C-SVC
1 -- nu-SVC
2 -- one-class SVM
3 -- epsilon-SVR
4 -- nu-SVR
-t kernel_type : set type of kernel function (default 2)
0 -- linear: u'*v
1 -- polynomial: (gamma*u'*v + coef0)^degree
2 -- radial basis function: exp(-gamma*|u-v|^2)
3 -- sigmoid: tanh(gamma*u'*v + coef0)

-d degree : set degree in kernel function (default 3)
-g gamma : set gamma in kernel function (default 1/k)
-r coef0 : set coef0 in kernel function (default 0)
-c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
-n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
-p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
-m cachesize : set cache memory size in MB (default 40)
-e epsilon : set tolerance of termination criterion (default 0.001)
-h shrinking: whether to use the shrinking heuristics, 0 or 1 (default 1)
-b probability_estimates: whether to train an SVC or SVR model for probability
estimates, 0 or 1 (default 0)
-wi weight : set the parameter C of class i to weight*C in C-SVC (default 1)
-v n: n-fold cross validation mode

The k in the -g option means the number of attributes in the input data.
option -v randomly splits the data into n parts and calculates cross validation
accuracy/mean squared error on them.

‘sparse-predict’ Usage

Usage: svm-predict [options] test_filemodel_fileoutput_file
-b probability estimates: whether to predict probability estimates, 0 or 1 (default 0); oneclass SVM not supported yet
model_file is the model file generated by svm-train.
test_file is the test data you want to predict.
svm-predict will produce output in the output_file.
(Source: Chang and Lin, 2001)
APPENDIX C: MAT LAB CODES FOR PROGRAM
APPENDIX C: MAT LAB CODES FOR PROGRAM

devidePoint=600;
shift=10;
% output data vection creation for svmtrain
% normalize the output
output = (ElectricalConsumption(:,3))
min(ElectricalConsumption(:,3)))/(max(ElectricalConsumption(:,3))
min(ElectricalConsumption(:,3)));

TrainOutput = output(shift+1:devidePoint+1,:); TestOutput =
output(devidePoint+2:end,:);

% this section belongs to input data matrix creation
*normalize inputs
Humidity_norm = (Humidity(:,4)-min(Humidity(:,4)))/(max(Humidity(:,4)-
min(Humidity(:,4))));
MaxTemperature_norm = (MaxTemperature(:,4)-
min(MaxTemperature(:,4)))/(max(MaxTemperature(:,4))-min(MaxTemperature(:,4)));
SolarRadiation_norm = (SolarRadiation(:,4)-
min(SolarRadiation(:,4)))/(max(SolarRadiation(:,4))-min(SolarRadiation(:,4)));
%moving average input & Outputs
%outputMa1 = transpose(tsmovavg(transpose(output), 'e', shift));
%outputMa2 = transpose(tsmovavg(transpose(MaxTemperature_norm ), 'e',
shift));
%outputMa3 = transpose(tsmovavg(transpose( Humidity_norm), 'e', shift));
%outputMa4 = transpose(tsmovavg(transpose(SolarRadiation_norm ), 'e', shift));
%create time delay input

timeDelayInput1   = [output(1:end)];
timeDelayInput11 = [output(1);output(1:end-1)];

%timeDelayInput2 = [output(1:end)];
%timeDelayInput22 = [output(1);output(1:end-1)];
%timeDelayInput222 = [output(1:2);output(1:end-2)];

%timeDelayInput3 = [outputMa3(1:end)];
%timeDelayInput33 = [outputMa3(1);outputMa3(1:end-1)];
%timeDelayInput333 = [outputMa3(1:2);outputMa3(1:end-2)];

%timeDelayInput4 = [outputMa4(1:end)];
%timeDelayInput44 = [outputMa4(1);outputMa4(1:end-1)];
%timeDelayInput444 = [outputMa4(1:2);outputMa4(1:end-2)];

input =
[timeDelayInput1,timeDelayInput11,timeDelayInput111,MaxTemperature_norm,Hybridity_norm,SolarRadiation_norm,\%outputMa4];

TrainInput = input(shift:devidePoint,:);
TestInput = input(devidePoint+1:end-1,:);

%% train svm model and set parameters

% Usage: model = svmtrain(training_label_vector, training_instance_matrix, 'libsvm_options');

% svm_options:
% -s svm_type : set type of SVM (default 0)
% 0 -- C-SVC
% 1 -- nu-SVC
% 2 -- one-class SVM
% 3 -- epsilon-SVR
% 4 -- nu-SVR
% -t kernel_type : set type of kernel function (default 2)
%  0 -- linear: u'*v
%  1 -- polynomial: (gamma*u'*v + coef0)^degree
%  2 -- radial basis function: exp(-gamma*|u-v|^2)
%  3 -- sigmoid: tanh(gamma*u'*v + coef0)
%  4 -- precomputed kernel (kernel values in training_instance_matrix)

% -d degree : set degree in kernel function (default 3)
% -g gamma : set gamma in kernel function (default 1/num_features)
% -r coef0 : set coef0 in kernel function (default 0)
% -c cost : set the parameter C of C-SVC, epsilon-SVR, and nu-SVR (default 1)
% -n nu : set the parameter nu of nu-SVC, one-class SVM, and nu-SVR (default 0.5)
% -p epsilon : set the epsilon in loss function of epsilon-SVR (default 0.1)
% -m cachesize : set cache memory size in MB (default 100)
% -e epsilon : set tolerance of termination criterion (default 0.001)
% -h shrinking : whether to use the shrinking heuristics, 0 or 1 (default 1)
% -b probability_estimates : whether to train a SVC or SVR model for probability estimates, 0 or 1 (default 0)
% -wi weight : set the parameter C of class i to weight*C, for C-SVC (default 1)
% -v n : n-fold cross validation mode
% -q : quiet mode (no outputs)

crossvalidation = svmtrain(TrainOutput,TrainInput,'-m 800 -v 5')
[K1 = [(1:630), 'K']; % include sample serial number as first column
%model = svmtrain(label_vector, K1,'-t 4');
%[predict_label, accuracy, dec_values] = svmpredict(label_vector, K1, model); % test the training data
ForecastingModel = svmtrain(TrainOutput,TrainInput,'-s 3 -t 2 -g 0.25 -c 1 -p 0.125');

%svm-scale -l -1 -u 1 -s range train > train.scale
%svm-scale -r range test > test.scale
%Scale each feature of the training data to be in [-1,1]. Scaling factors are stored in the file range and then used for scaling the %test data.

%s svm-train -s 0 -c 5 -t 2 -g 0.5 -e 0.1 data_file
%Train a classifier with RBF kernel \( \exp(-0.5|u-v|^2) \), \( C=10 \), and %stopping tolerance 0.1.
%svm-train -s 3 -p 0.1 -t 0 data_file
%Solve SVM regression with linear kernel \( u'v \) and epsilon=0.1 in the loss function.
%svm-train -c 10 -w1 1 -w2 5 -w4 2 data_file
%Train a classifier with penalty 10 = 1 * 10 for class 1, penalty 50 = 5 * 10 for class 2, and penalty 20 = 2 * 10 for class 4.
%svm-train -s 0 -c 100 -g 0.1 -v 5 data_file
%Do five-fold cross validation for the classifier using the parameters \( C = 100 \) and \( \gamma = 0.1 \)
%svm-train -s 0 -b 1 data_file
%svm-predict -b 1 test_file data_file.model output_file
%Obtain a model with probability information and predict test data with probability estimates

%svm_parameter(svm_type=0,kernel_type=2,gamma=1,cache_size=40,eps=0.001,C =1,nr_weight=0,shrinking=1)
%crossvalidation = svmtrain(TrainOutput,TrainInput,'-m 800 -v 5')
%s/vm-train -c 2 -g 2 svmguide1.scale
%. /svm-predict svmguide1.t.scale svmguide1.scale.model svmguide1.t.predict
%. /svm-train -s 4 -t 2 -g .1 -c 120 TrainFile.txt ModelFile.txt
%. /svm-predict TestFile.txt ModelFile.txtOutputFile.txt.
%./svm-train svmguide1
%./svm-predict svmguide1.t svmguide1.model svmguide1.t.predict
% test svm model

[predictionP, accuracyP, decvalueP] = svmpredict(TestOutput, TestInput, ForecastingModel, '-b 0');

predictionP;
accuracyP;
decvalueP;
TestOutput;
p1 = plot(predictionP);
set(p1,'Color','red')
hold on
p2 = plot(TestOutput);
set(p2,'Color','black')
hold on
p3 = plot(Humidity_norm)
set(p3,'Color','red')
hold on

p4 = plot(MaxTemperature_norm)
set(p4,'Color','black')
hold on

p5 = plot(SolarRadiation_norm)
set(p5,'Color','blue')
hold on

MSE & R^2, Nsv data

For constant gamma = 0.1 and varying C values
p2 = plot(X, Y)
%set(p2,'Color','red')
%p3=plot(X,Z)
%set(p3,'Color','black')

%p5=plot(X,xxa)
%set(p5,'Color','blue')
%hold on