**LAB 4: Support Vector Machine/Regression with R, Matlab and Python**

**Objectives:**

**Materials**

|  |  |
| --- | --- |
| **File Name** | **Description** |
| *Climate\_zone\_cali.csv* | Calibration data set for classifying climate zone in Republic of Korea |
| *Climate\_zone\_vali.csv* | Validation data set for classifying climate zone in Republic of Korea |
| *PM10\_cal\_AIRS.csv* | Calibration data set for predicting PM 10 in South Korea |
| *PM10\_val\_AIRS.csv* | Validation data set for predicting PM 10 in South Korea |
| *svm.m* | Matlab code of SVM |
| *svm.r* | R code of SVM |
| *Python\_SVM .py* | Python code of SVM |

**PART I: SVM in Matlab**

**Task 1. For classification**

1. Setup environment: Specify directory and open data

* First, change the directory
* Open calibration and validation file of climate zone using ‘readtable’ function
* Predictor data to which the SVM classifier is trained, specified as a matrix of numeric values.
* Class labels to which the SVM model is trained, specified as a categorical or character array, logical or numeric vector, or cell array of strings.

path = 'G:\class\_SVM\matlab\';

% open calibration file as table

cali = readtable([path 'climate\_zone\_cali.csv']);

cali\_input = table2array(cali(:,1:size(cali,2)-1)); % calibration input

cali\_target = table2array(cali(:,12)); %calibration target

% open validation file as table

vali = readtable([path 'climate\_zone\_vali.csv']);

vali\_input = table2array(vali(:,1:size(cali,2)-1)); % validation input

vali\_target = table2array(vali(:,12)); %validation target

1. Make SVM model

* Find the type of class for iteration and make empty cell array for save SVM model

classes = unique(cali\_target); % class type

SVMModels = cell(size(classes,1),1);

* For each class:
  1. Create a logical vector (indx) indicating whether an observation is a member of the class.
  2. Train an SVM classifier using the predictor data and indx.
  3. Store the classifier in a cell of a cell array.
* You can set-up parameters such as kernel function, standardization…….
* If you using optimization function…

for j = 1:numel(classes);

% Create binary classes for each classifier

indx = strcmp(cali\_target,classes(j));

%optimization

SVMModels{j} = fitcsvm(X,Y,'OptimizeHyperparameters','all',...

'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName', 'expected-improvement-plus'))

end

for j = 1:numel(classes);

% Create binary classes for each classifier

indx = strcmp(cali\_target,classes(j));

% Train an SVM classifier using cali\_input data and indx and Store the classifier in a cell of a cell array.

SVMModels{j} = fitcsvm(cali\_input,indx,'ClassNames',[false true],'Standardize',true,...

'KernelFunction','rbf','BoxConstraint',1);

end

1. Predict using validation data

* Each row of Scores contains three scores. The index of the element with the largest score is the index of the class to which the new class observation most likely belongs.
* Associate each new observation with the classifier that gives it the maximum score.

for j = 1:numel(classes);

[~,score] = predict(SVMModels{j},vali\_input);

% Second column contains positive-class scores

Scores(:,j) = score(:,2);

end

[~,maxScore] = max(Scores,[],2);

**Task 2. For regression**

1. Setup environment : Specify directory and open data

path = 'G:\class\_SVM\matlab\';

cali = readtable([path 'PM10\_cal\_AIRS.csv']); % open calibration file

vali = readtable([path 'PM10\_val\_AIRS.csv']); % open validation file

1. Make SVR model and predicting

SVRmodel = fitrsvm(cali,'PM10','KernelFunction','gaussian','KernelScale','auto','Standardize',true);

% optimization

%SVRmodel = fitrsvm(X,Y,'OptimizeHyperparameters','all', 'HyperparameterOptimizationOptions',struct('AcquisitionFunctionName',...

% 'expected-improvement-plus'))

test = predict(SVRmodel,vali);

**PART II: SVM in R**

**Task 1. For classification**

1. Setup environment: Specify directory and open data

* First, install the “e1071” packages and open library
* Change the directory using ‘setwd’
* Open calibration and validation file of climate zone using ‘read.csv’
* Set the input variables and target variable for calibration and validation file, respectively

install.packages("e1071")

library(e1071)

#set the directory

setwd("C:/Users/Seohui Park/Desktop/SVM\_AIRS\_lab/")

#read data

cal\_class = read.csv(file="calimate\_zone\_cali.csv")

val\_class = read.csv(file=" calimate\_zone\_vali.csv")

head(cal\_class,5) # show header

attach(cal\_class) # 데이터를 R 검색 경로에 추가하여 변수명으로 접근가능하게 함. 해제는 detach

x <- subset(cal\_class, select = -target) # calibration input, all variables except row of target

y <- target # calibration target

x\_val <- val\_class[,-11] # validation input, all variables except row of 11(target)

y\_val<- val\_class[,11] # validation target

1. Make SVM model

* create SVM model with default kernel parameter

## classification mode

# default with factor response:

model <- svm(target ~ ., data = cal\_class)

summary(model)

# alternatively the traditional interface:

model\_2 <- svm(x, y)

summary(model\_2)

* model and model\_2 show same SVM model summary.
* If you successfully run the above code, you can see this tables

# test with train data with default SVM model

pred <- fitted(model)

system.time(pred <- fitted(model)) # check running time

# Check accuracy:

y <- cal\_class$target

table(pred, y)

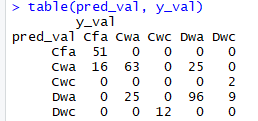
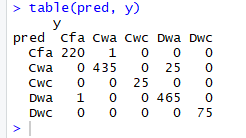
attach(val\_class) # 데이터를 R 검색 경로에 추가하여 변수명으로 접근가능하게 함. 해제는 detach

x\_val <- subset(val\_class, select = -target)

y\_val <- target

pred\_val <- predict(model, x\_val) # test with test data with default SVM model

table(pred\_val, y\_val) # Check accuracy:



1. Optimize kernel parameter

* To find the best kernel parameters for SVM model, tune function is used with parameter range you set.
* The tuning parameter is conducted by using grid search method.
* The kernel function should be changed depending on the data

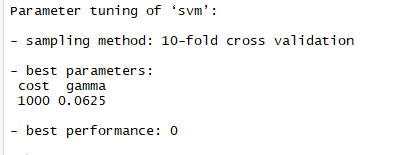
# Tuning SVM to find best cost and gamma

svm\_tune <- tune(svm, train.x=x, train.y=y, kernel="radial", ranges=list(cost=10^(-1:3), gamma=2^(-5:1)))

#gamma=c(.1,.2,.3,.4,.5,.6,.7,.8,.9,1,1.5,2, you can also set the parameter range with this form

print(svm\_tune)

* Kernel: ‘linear’, ‘radial’, ‘polynomial’, ‘sigmoid’
* Also, you should use the kernel parameter depending on the kernel
* You can see the best parameters by tuning the SVM



1. Create new SVM model

* After you find the best cost and gamma, you can create new svm model again and try to run again
* If you do NOT want to standardize the variables, add the ‘scale = FALSE’ as option

#After finding the best cost and gamma

svm\_model\_after\_tune <- svm(target ~ ., data=cal\_class, kernel="radial", cost=1000, gamma=0.025)

#scale=FALSE: NOT standardization

summary(svm\_model\_after\_tune)

# test with test data with SVM model after tuning

pred\_val\_after\_tune <- predict(svm\_model\_after\_tune, x\_val)

table(pred\_val\_after\_tune, y\_val) # Check accuracy

* Compare the confusion matrix result of prediction using default SVM model and ne SVM model after tuning.

**Task 2. For regression**

1. Setup environment: Specify directory and open data

* First, install the “e1071” packages and open library
* Change the directory using ‘setwd’
* Open calibration and validation file of climate zone using ‘read.csv’
* Set the input variables and target variable for calibration and validation file, respectively

## try regression mode

install.packages("e1071")

library(e1071)

#set the directory

setwd("C:/Users/Seohui Park/Desktop/SVM\_AIRS\_lab/")

#read data

cal\_reg = read.csv(file="PM10\_cal\_AIRS.csv")

val\_reg = read.csv(file="PM10\_val\_AIRS.csv")

head(cal\_class,5) # show header

attach(cal\_class) # 데이터를 R 검색 경로에 추가하여 변수명으로 접근가능하게 함. 해제는 detach

x <- subset(cal\_reg, select = -PM10) # calibration input, all variables except row of target

y <- PM10 # calibration target

x\_val <- val\_reg[,-11] # validation input, all variables except row of 11(target)

y\_val<- val\_reg[,11] # validation target

1. Create SVM regression(SVR) model

* Create SVR model with default parameter
* Estimate train data and test data using default SVR model, respectively

# estimate model and predict input values for train data

model <- svm(x, y)

predicted\_Y <- predict(model, x)

# estimate model and predict input values for test data

newdata <- data.frame(y\_val)

predicted\_Y\_val = predict(model, newdata)

1. Tuning SVR model

* To find the best parameters for SVR model, tune function is used with parameter range you set.
* The tuning parameter is conducted by using grid search method.
* Create new SVR model using the best optimized model by tuning
* Estimate train and test dataset using new SVR model
* Also, you can compare the results from default SVR model and new SVR model by plotting them.

## Tuning SVR model by varying values of maximum allowable error and cost parameter

#Tune the SVM model

OptModel=tune(svm, PM10~., data=cal\_reg,ranges=list(elsilon=seq(0,1), cost=10^(-1:3)))

model\_tune<-OptModel$best.model

predicted\_Y\_tune <- predict(model\_tune, x)

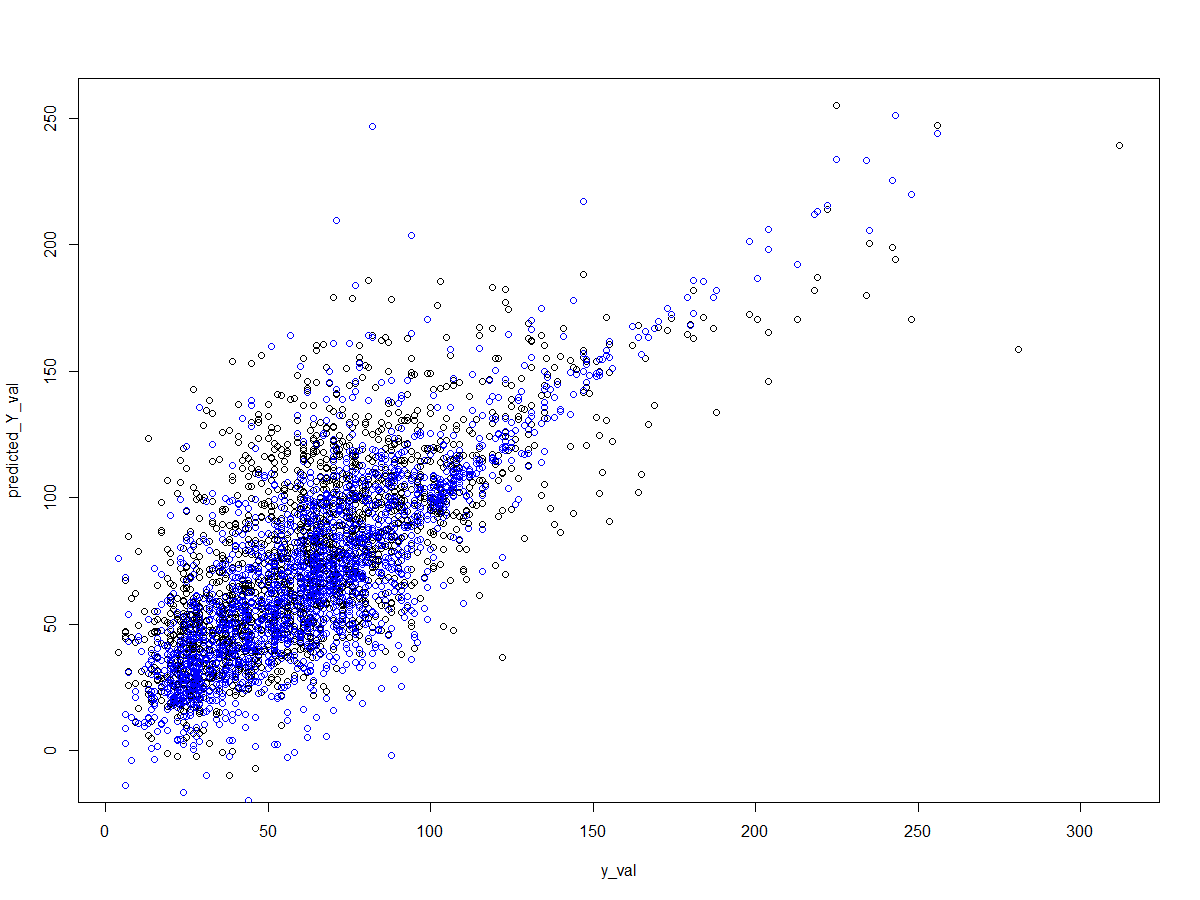
predicted\_Y\_val\_tune <- predict(model\_tune, x\_val)

# visualize

plot(y\_val,predicted\_Y\_val)

points(y\_val, predicted\_Y\_val\_tune, col = 4) # adding points in the previous plot

* You can get the below graph by plotting the results



**PART III: SVM in Python**

**Task 1. For classification**

1. Setup environment: Specify directory and open data

* First, install libraries and import utilities described below



* Set your workspace and import dataset
* In this process, you can standardize dataset of each variable
  + Preprocessing.scale() : standardization

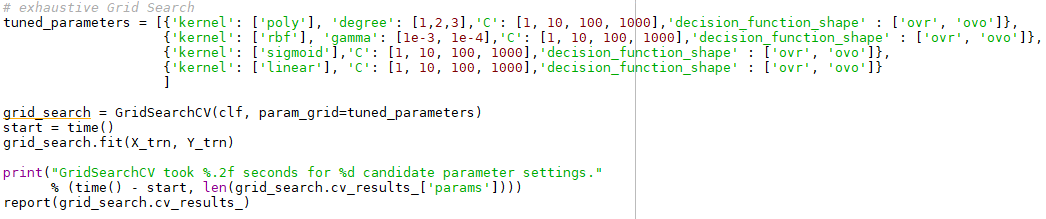


1. Build a SVM model and optimize parameters

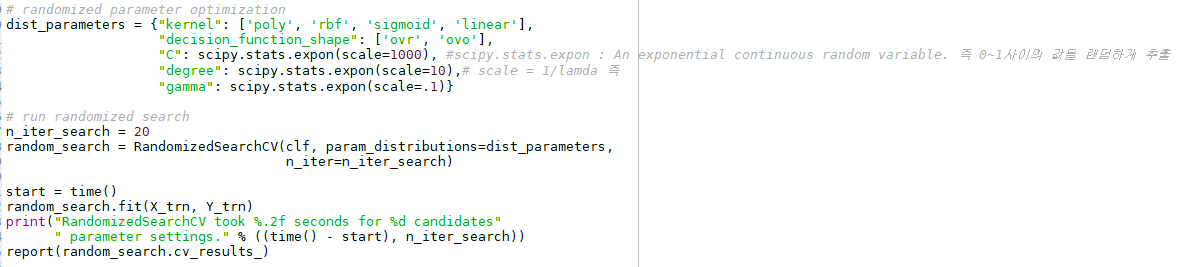
* Define a SVM function as “clf”



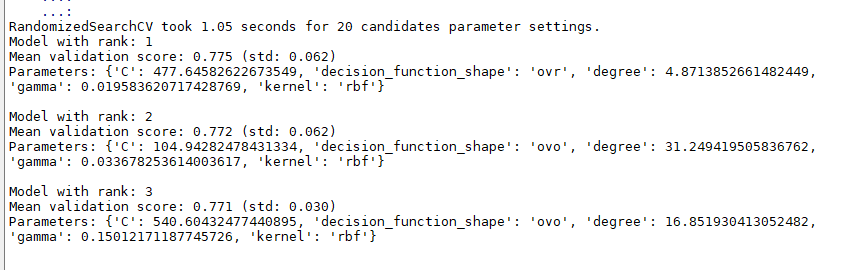
* SVM model parameters
  + kernel = ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’
  + C = float, optional (default = 1.0)
  + Degree = int, optional (default=3)
  + Gamma = float, optional (default=’auto’)
* Two parameter optimization method
  + Exhaustive grid search



* + Randomized parameter optimization
    - You can set number of model with randomized parameters (n\_iter\_search)

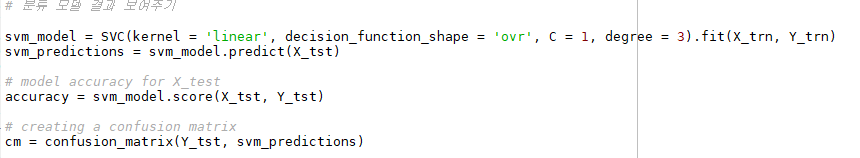


* Able to find out best parameters

****

1. Set and run the SVM model using optimized parameters

* Build a SVM model and optimize parameters
* Adjust optimized parameters to SVM model and conduct classification



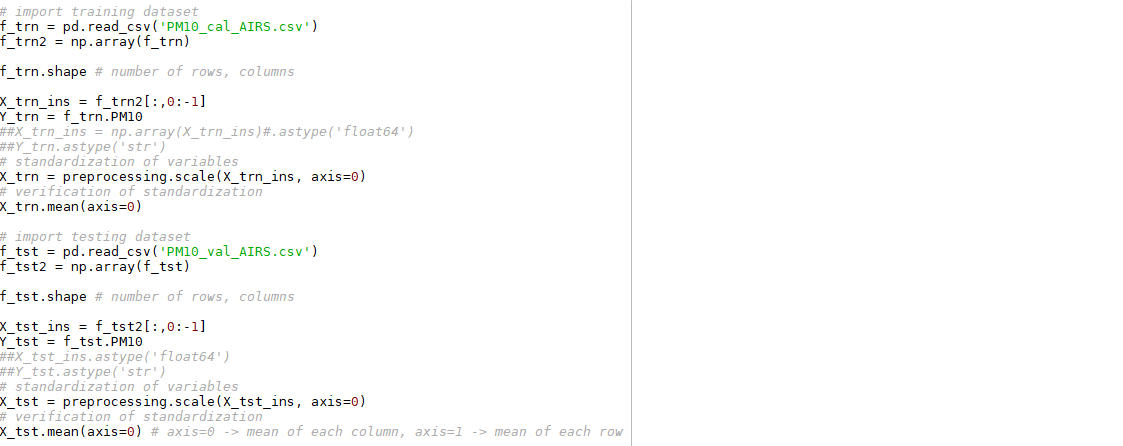
* Able to find out best parameters

**Task 2. For regression**

1. Setup environment: Specify directory and open data
   * First, install libraries and import utilities described below



* Set your workspace and import dataset
* In this process, you can standardize dataset of each variable
  + Preprocessing.scale() : standardization

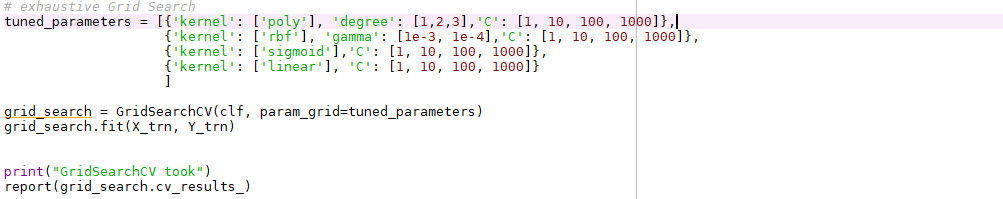


1. Build a SVM model and optimize parameters

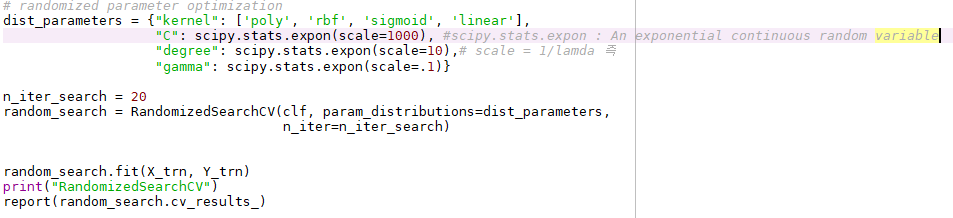
* Define a SVR function as “clf”



* SVRmodel parameters
  + kernel = ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’
  + C = float, optional (default = 1.0)
  + Degree = int, optional (default=3)
  + Gamma = float, optional (default=’auto’)
* Two parameter optimization method
  + Exhaustive grid search

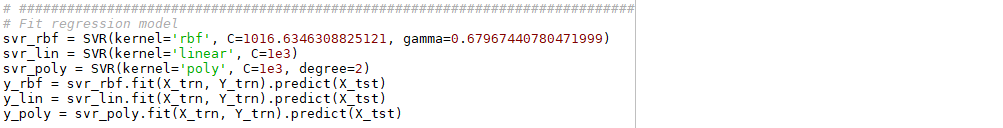


* + Randomized parameter optimization



1. Set and run the SVM model using optimized parameters

* Adjust best parameter to SVR model and validate using your test dataset



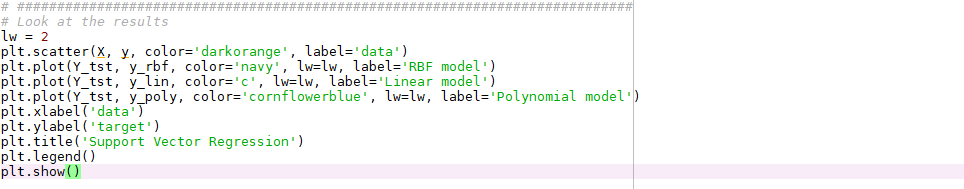
Kernel = ‘rbf’

C = 1016

Gamma = 0.679

(Fill your best parameters for set the model)

* Visualization prediction results



**Assignment**

1. Make SVM/SVR model as changing parameters such as kernel function, sigma using any preferred program. Compare the results and describe the process of finding optimal parameters. Report your results in a document and submit to TA (dhan@unist.ac.kr).