Artificial Intelligence for Remote Sensing Applications
Instructor: Dr. Jungho Im
Presenters: Sumin Park, Seohui Park, Juhyun Lee
Due Date: April 8th, 2018

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
```

2) 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:
     a. Create a logical vector (`indx`) indicating whether an observation is a member of the class.
     b. Train an SVM classifier using the predictor data and `indx`.
     c. Store the classifier in a cell of a cell array.
   - You can set-up parameters such as kernel function, standardization…….

```
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
```

- 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
```

3) 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
```

2) Make SVR model and predicting

```
SVRmodel =
fitrsvm(cali,'PM10','KernelFunction','gaussian','KernelScale','auto','S
tandardize',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
```

2) 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.

```
# 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:
```

- If you successfully run the above code, you can see this tables

```
> table(pred, y)
      y
pred  Cfa Cwa Cwc Dwa Dwc
  Cfa 220   1   0   0   0
  Cwa   0 435   0  25   0
  Cwc   0   0  25   0   0
  Dwa   1   0   0 465   0
  Dwc   0   0   0   0  75
>
```

```
> table(pred_val, y_val)
        y_val
pred_val Cfa Cwa Cwc Dwa Dwc
     Cfa  51   0   0   0   0
     Cwa  16  63   0  25   0
     Cwc   0   0   0   0   2
     Dwa   0  25   0  96   9
     Dwc   0   0  12   0   0
```

3) 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

```
Parameter tuning of ‘svm’:

- sampling method: 10-fold cross validation

- best parameters:
 cost  gamma
 1000 0.0625

- best performance: 0
```

4) 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
```

2) 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)
```

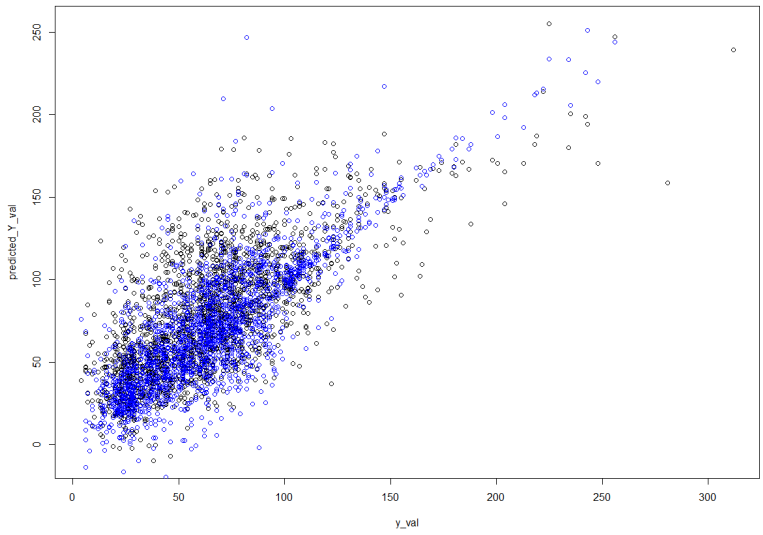
3) Tuning SVR model
   - To find the best parameters for SVR model, tune function is used with parameter range you set.
   - The tuning parameter is conducted by using grid search method.
   - Create new SVR model using the best optimized model by tuning
   - Estimate train and test dataset using new 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

```
#Import Library
import csv
import pandas as pd
import numpy as np
import scipy
from __future__ import print_function

from sklearn import datasets
#from sklearn.model_selection import train_test_split # trainig/testing 데이터셋으로 나눠있지 않을때 이 유틸으로 나눌수있음

from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn import preprocessing
from time import time
from scipy.stats import randint as sp_randint
from scipy.stats.distributions import expon
from sklearn import svm, grid_search
from sklearn.svm import SVC
from io import StringIO
```

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

```
# import training dataset
f_trn = pd.read_csv('~/climate_zone_cali.csv')
f_trn2 = np.array(f_trn)

f_trn.shape # number of rows, columns

X_trn_ins = f_trn2[:,0:-1]
Y_trn = f_trn.target
##X_trn_ins = np.array(X_trn_ins)#.astype('float64')
##Y_trn.astype('str')
# standardization of variables
X_trn = preprocessing.scale(X_trn_ins, axis=0)
# verification of standardization
X_trn.mean(axis=0)

# import testing dataset
f_tst = pd.read_csv('~/climate_zone_vali.csv')
f_tst2 = np.array(f_tst)

f_tst.shape # number of rows, columns

X_tst_ins = f_tst2[:,0:-1]
Y_tst = f_tst.target
##X_tst_ins.astype('float64')
##Y_tst.astype('str')
# standardization of variables
X_tst = preprocessing.scale(X_tst_ins, axis=0)
# verification of standardization
X_tst.mean(axis=0) # axis=0 -> mean of each column, axis=1 -> mean of each row
```

2) Build a SVM model and optimize parameters

- Define a SVM function as “clf”

```
clf = svm.SVC()
```

- 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

```
# exhaustive Grid Search
tuned_parameters = [{'kernel': ['poly'], 'degree': [1,2,3],'C': [1, 10, 100, 1000],'decision_function_shape' : ['ovr', 'ovo']},
                    {'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000],'decision_function_shape' : ['ovr', 'ovo']},
                    {'kernel': ['sigmoid'],'C': [1, 10, 100, 1000],'decision_function_shape' : ['ovr', 'ovo']},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000],'decision_function_shape' : ['ovr', 'ovo']}
                    ]

grid_search = GridSearchCV(clf, param_grid=tuned_parameters)
start = time()
grid_search.fit(X_trn, Y_trn)

print("GridSearchCV took %.2f seconds for %d candidate parameter settings."
      % (time() - start, len(grid_search.cv_results_['params'])))
report(grid_search.cv_results_)
```

✓ Randomized parameter optimization

➢ You can set number of model with randomized parameters (n_iter_search)

```
# randomized parameter optimization
dist_parameters = {"kernel": ['poly', 'rbf', 'sigmoid', 'linear'],
                   "decision_function_shape": ['ovr', 'ovo'],
                   "C": scipy.stats.expon(scale=1000), #scipy.stats.expon : An exponential continuous random variable. 즉 0~1사이의 값을 랜덤하게 추출
                   "degree": scipy.stats.expon(scale=10),# scale = 1/lamda 즉
                   "gamma": scipy.stats.expon(scale=.1)}

# run randomized search
n_iter_search = 20
random_search = RandomizedSearchCV(clf, param_distributions=dist_parameters,
                                   n_iter=n_iter_search)

start = time()
random_search.fit(X_trn, Y_trn)
print("RandomizedSearchCV took %.2f seconds for %d candidates"
      " parameter settings." % ((time() - start), n_iter_search))
report(random_search.cv_results_)
```

- Able to find out best parameters

```
    ...:
    ...:
RandomizedSearchCV took 1.05 seconds for 20 candidates parameter settings.
Model with rank: 1
Mean validation score: 0.775 (std: 0.062)
Parameters: {'C': 477.64582622673549, 'decision_function_shape': 'ovr', 'degree': 4.8713852661482449,
'gamma': 0.019583620717428769, 'kernel': 'rbf'}

Model with rank: 2
Mean validation score: 0.772 (std: 0.062)
Parameters: {'C': 104.94282478431334, 'decision_function_shape': 'ovo', 'degree': 31.249419505836762,
'gamma': 0.033678253614003617, 'kernel': 'rbf'}

Model with rank: 3
Mean validation score: 0.771 (std: 0.030)
Parameters: {'C': 540.60432477440895, 'decision_function_shape': 'ovo', 'degree': 16.851930413052482,
'gamma': 0.15012171187745726, 'kernel': 'rbf'}
```

3) 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

```
# 분류 모델 결과 보여주기

svm_model = SVC(kernel = 'linear', decision_function_shape = 'ovr', C = 1, degree = 3).fit(X_trn, Y_trn)
svm_predictions = svm_model.predict(X_tst)

# model accuracy for X_test
accuracy = svm_model.score(X_tst, Y_tst)

# creating a confusion matrix
cm = confusion_matrix(Y_tst, svm_predictions)
```

- 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

```
#Import Library
import csv
import pandas as pd
import numpy as np
import scipy
import matplotlib.pyplot as plt
import time

from __future__ import print_function
from __future__ import division

from sklearn import datasets
#from sklearn.model_selection import train_test_split # trainig/testing 데이터셋으로 나눠있지 않을때 이 유틸으로 나눌수있음
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.svm import SVC
from sklearn import preprocessing
from time import time
from scipy.stats import randint as sp_randint
from scipy.stats.distributions import expon
from sklearn import svm, grid_search
from sklearn.svm import SVC
from io import StringIO
```

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

```
# import training dataset
f_trn = pd.read_csv('PM10_cal_AIRS.csv')
f_trn2 = np.array(f_trn)

f_trn.shape # number of rows, columns

X_trn_ins = f_trn2[:,0:-1]
Y_trn = f_trn.PM10
##X_trn_ins = np.array(X_trn_ins)#.astype('float64')
##Y_trn.astype('str')
# standardization of variables
X_trn = preprocessing.scale(X_trn_ins, axis=0)
# verification of standardization
X_trn.mean(axis=0)

# import testing dataset
f_tst = pd.read_csv('PM10_val_AIRS.csv')
f_tst2 = np.array(f_tst)

f_tst.shape # number of rows, columns

X_tst_ins = f_tst2[:,0:-1]
Y_tst = f_tst.PM10
##X_tst_ins.astype('float64')
##Y_tst.astype('str')
# standardization of variables
X_tst = preprocessing.scale(X_tst_ins, axis=0)
# verification of standardization
X_tst.mean(axis=0) # axis=0 -> mean of each column, axis=1 -> mean of each row
```

2) Build a SVM model and optimize parameters

- Define a SVR function as “clf”

```
clf = SVR()
```

- 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

```
# exhaustive Grid Search
tuned_parameters = [{'kernel': ['poly'], 'degree': [1,2,3],'C': [1, 10, 100, 1000]},
                    {'kernel': ['rbf'], 'gamma': [1e-3, 1e-4],'C': [1, 10, 100, 1000]},
                    {'kernel': ['sigmoid'],'C': [1, 10, 100, 1000]},
                    {'kernel': ['linear'], 'C': [1, 10, 100, 1000]}
                    ]

grid_search = GridSearchCV(clf, param_grid=tuned_parameters)
grid_search.fit(X_trn, Y_trn)


print("GridSearchCV took")
report(grid_search.cv_results_)
```

  - ✓ Randomized parameter optimization

```
# randomized parameter optimization
dist_parameters = {"kernel": ['poly', 'rbf', 'sigmoid', 'linear'],
                   "C": scipy.stats.expon(scale=1000), #scipy.stats.expon : An exponential continuous random variable
                   "degree": scipy.stats.expon(scale=10),# scale = 1/lamda 즉
                   "gamma": scipy.stats.expon(scale=.1)}

n_iter_search = 20
random_search = RandomizedSearchCV(clf, param_distributions=dist_parameters,
                                   n_iter=n_iter_search)


random_search.fit(X_trn, Y_trn)
print("RandomizedSearchCV")
report(random_search.cv_results_)
```

3) Set and run the SVM model using optimized parameters

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

```
# ##############################################################################
# Fit regression model
svr_rbf = SVR(kernel='rbf', C=1016.6346308825121, gamma=0.67967440780471999)
svr_lin = SVR(kernel='linear', C=1e3)
svr_poly = SVR(kernel='poly', C=1e3, degree=2)
y_rbf = svr_rbf.fit(X_trn, Y_trn).predict(X_tst)
y_lin = svr_lin.fit(X_trn, Y_trn).predict(X_tst)
y_poly = svr_poly.fit(X_trn, Y_trn).predict(X_tst)
```

Kernel = 'rbf'
C = 1016
Gamma = 0.679
(Fill your best parameters for set the model)

- Visualization prediction results

```
# ##############################################################################
# Look at the results
lw = 2
plt.scatter(X, y, color='darkorange', label='data')
plt.plot(Y_tst, y_rbf, color='navy', lw=lw, label='RBF model')
plt.plot(Y_tst, y_lin, color='c', lw=lw, label='Linear model')
plt.plot(Y_tst, y_poly, color='cornflowerblue', lw=lw, label='Polynomial model')
plt.xlabel('data')
plt.ylabel('target')
plt.title('Support Vector Regression')
plt.legend()
plt.show()
```

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).