```
##Part 1. Classification with CarAuction Data

###
------------------------------------------------------------------------
# This dataset contains information of cars purchased at the Auction.
# We will use this file to predict the quality of buying decisions and
visualize decision processes.

# VARIABLE DESCRIPTIONS:
#Auction: Auction provider at which the  vehicle was purchased
#Color: Vehicle Color
#IsBadBuy: Identifies if the kicked vehicle was an avoidable purchase
#MMRCurrentAuctionAveragePrice: Acquisition price for this vehicle in average
condition as of current day
#Size: The size category of the vehicle (Compact, SUV, etc.)
#TopThreeAmericanName:Identifies if the manufacturer is one of the top three
American manufacturers
#VehBCost: Acquisition cost paid for the vehicle at time of purchase
#VehicleAge: The Years elapsed since the manufacturer's year
#VehOdo: The vehicles odometer reading
#WarrantyCost: Warranty price (term=36month  and millage=36K)
#WheelType: The vehicle wheel type description (Alloy, Covers)
###
------------------------------------------------------------------------


# 1. Import the datadet
carAuction <- read.csv(file = "carAuction.csv", stringsAsFactors = FALSE)

# 2. str() shows the structure of data
str(carAuction)

# 3. summary() shows the mean and the five-number statistics indicating the
spread of each column's values
summary(carAuction)

# 4. Change all categorical variables to factors
carAuction$Auction <- factor(carAuction$Auction)
carAuction$Color <- factor(carAuction$Color)
carAuction$IsBadBuy <- factor(carAuction$IsBadBuy)
carAuction$Size <- factor(carAuction$Size)
carAuction$TopThreeAmericanName <- factor(carAuction$TopThreeAmericanName)
carAuction$WheelType <- factor(carAuction$WheelType)
str(carAuction)
summary(carAuction)

# 5. Partition the dataset: 70% for training, 30% for testing
library(caret)
set.seed(1)
train_index <- createDataPartition(carAuction$IsBadBuy, p=0.7, list=FALSE)
datTrain <- carAuction[train_index,]
datTest <- carAuction[-train_index,]

# 6. Check the rows and porportion of target variable for both training and
testing datasets
nrow(datTrain)
```

```
nrow(datTest)
prop.table(table(datTrain$IsBadBuy))
prop.table(table(datTest$IsBadBuy))

# 7. Build svm model with default setting (in default setting, C=1)
library(kernlab)#default setting is C=1
library(rminer)
svm_model <- ksvm(IsBadBuy~.,data=datTrain)
svm_model
#parameter C is used to conrol the trade off between maximazing
#the margin adn minimizing the classification error.
#we build svm model and then we can use it to make prediction on
#the training and testing data.

# Make predictions on both training and tessting sets
prediction_on_train <- predict(svm_model, datTrain)
prediction_on_test <- predict(svm_model, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test,metric=c("ACC","PRECISION","TPR","F1"))


# 8. Build svm model with C=5
svm_model2 <- ksvm(IsBadBuy~.,data=datTrain, C=5)
svm_model2

# Make predictions on both training and tessting sets
prediction_on_train2 <- predict(svm_model2, datTrain)
prediction_on_test2 <- predict(svm_model2, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train2, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test2, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train2,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test2,metric=c("ACC","PRECISION","TPR","F1"))
#with this model, in comparison to the previous one (c=1)we can
#see that the testing and training performance is getting slightly better.

# 8. Build svm model with C=10
svm_model3 <- ksvm(IsBadBuy~.,data=datTrain, C=10)
svm_model3

# Make predictions on both training and tessting sets
prediction_on_train3 <- predict(svm_model3, datTrain)
prediction_on_test3 <- predict(svm_model3, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train3, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test3, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train3,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test3,metric=c("ACC","PRECISION","TPR","F1"))
#when we increase the vlaue of C the model tends to overfit on the training
dataset.
```

```
#The tranining perforamnce got a little bit better, but the testing
#performance slightly worsened.

# 9. Build svm model with C=50
svm_model4 <- ksvm(IsBadBuy~.,data=datTrain, C=50)
svm_model4

# Make predictions on both training and tessting sets
prediction_on_train4 <- predict(svm_model4, datTrain)
prediction_on_test4 <- predict(svm_model4, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train4, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test4, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train4,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test4,metric=c("ACC","PRECISION","TPR","F1"))
# this model is overfitting on the training dataset.

# How dose the cost parameter C impact SVM model performance?
#The parameter C is used to control the balance between
#maximizing the margin and minimizing the classification errors.
#so as we increase the C value the svm model will apply higher cost
#on missclassified instance. Thus, svm will make fewer mistakes on the
#training dataset. Setting a high C value can cause overfit on training
dataset.
#This means that the model might correspond to closely to the training data,
#and the accuracy of prediction on testing data could decrease.

# Which C value provides the best overall performance, C=1, 5, 10 or 50? And
way?
#Note: Compare the accuracy of each model testing data.
# C=5 gives the highest accuracy on the testing data which means it results in
#the best overall performance.
#C=5 89.52 ACC; C=10 89.49 ACC; C=50 87.52 ACC.

# 10. Build MLP model with default setting
#MPL represents multilayer perception.
# MLP's default parameter values of MLP,L=0.3,M=0.2, N=500,H='a'
# L: learning rate with default=0.3
# M: momemtum with default=0.2
# N: number of epochs with default=500
# H <comma seperated numbers for nodes on each layer>
#The hidden nodes to be created on each layer:
# an integer, or the letters 'a' = (attribs + classes) / 2,
#'i' = attribs, 'o' = classes, 't' = attribs + classes)
#for wildcard values, Default = 'a').
library(RWeka)
library(rminer)
MLP <- make_Weka_classifier("weka/classifiers/functions/MultilayerPerceptron")
#line above defines MLP function.
mlp_model <- MLP(IsBadBuy~.,data=datTrain)
#here we building MLP with default settings. we specify what is the target
#variable and the predictors.(we want to use all the predictors here)
summary(mlp_model)

# Make predictions on both training and tessting sets
```

```
prediction_on_train <- predict(mlp_model, datTrain)
prediction_on_test <- predict(mlp_model, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test,metric=c("ACC","PRECISION","TPR","F1"))


# 11. Build MLP model contains two hidden layers: 16 hidden nodes for the
#first layer, and 8 hidden nodes for the second layer. Set N = 100

mlp_model2 <-  MLP(IsBadBuy~.,data=datTrain, control = Weka_control(N=100,
H='16, 8'))
summary(mlp_model2)
#NOTE: N represents the number of itterations on which the model will be
trained.
#The default is n 500, in this case we want to train it on training
#data set 100 times.But the model here will be more complex, because it has
#2 inner layers 16 and 8

# Make predictions on both training and tessting sets
prediction_on_train2 <- predict(mlp_model2, datTrain)
prediction_on_test2 <- predict(mlp_model2, datTest)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain$IsBadBuy,prediction_on_train2, metric="CONF")
mmetric(datTest$IsBadBuy,prediction_on_test2, metric="CONF")
mmetric(datTrain$IsBadBuy,prediction_on_train2,metric=c("ACC","PRECISION","TPR","F1"))
mmetric(datTest$IsBadBuy,prediction_on_test2,metric=c("ACC","PRECISION","TPR","F1"))



# 12. cross validation
# Set up cv parameters
# df: identifies the whole data set by its name
# target: identifies the target variable by its column index in df
# nFolds: indicates the number of folds for cv
# seedVal: carries the seed value for random sampling of instances when
creating folds
# prediction_method: indicates the prediction method - e.g., lm
# metric_list: is a list of evaluation metrics that mmetric should generate

library(matrixStats)
library(knitr)
# Training performance for cross validation
cv_function_train <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)#note 2 cross validation functions,
    #one on training one on testing
```

```
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,train_input)
    return(mmetric(train_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
# Testing performance for cross validation
cv_function_test <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,test_input)
    return(mmetric(test_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}

# 3-fold cv with SVM method
df= carAuction #dataframe
target = 3 #the column # of the target variable
nFolds = 3 #number of folds
seedval = 1
prediction_method = ksvm #the name of the function that we want to use
```

```
metrics_list = c("ACC","PRECISION","TPR","F1")
cv_function_train(df, target, nFolds, seedval, prediction_method,
metrics_list)
cv_function_test (df, target, nFolds, seedval, prediction_method,
metrics_list)
#after you run it, you will see the performance of each folds first on
training
#than on testing.

# 3-fold cv with MLP method
#Note: we use the same code as above but we replace ksvm (in prediction
method)
#with MLP.
# Testing performance for cross validation
#Answer:
############################NOTE: copied from above and  renamed it
accordingly,
#######so that previous results wont be overwritten. Train2 and Test2
cv_function_train2 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)#note 2 cross validation functions,
    #one on training one on testing
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,train_input)
    return(mmetric(train_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
# Testing performance for cross validation
cv_function_test2 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)
```

```
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,test_input)
    return(mmetric(test_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
df= carAuction
target = 3
nFolds = 3
seedval = 1
prediction_method = MLP #instead of ksvm we use MLP
metrics_list = c("ACC","PRECISION","TPR","F1")
cv_function_train2(df, target, nFolds, seedval, prediction_method,
metrics_list)
cv_function_test2 (df, target, nFolds, seedval, prediction_method,
metrics_list)



##Part 2. Numeric Prediction with Insurance Data

'In order for a health insurance company to make money, it needs to collect
more
in yearly premiums than it spends on medical care to its beneficiaries. As a
result,
insurers invest a great deal of time and money in developing models that
accurately
forecast medical expenses for the insured population.

Medical expenses are difficult to estimate because the most costly conditions
are
rare and seemingly random. Still, some conditions are more prevalent for
certain
segments of the population. For instance, lung cancer is more likely among
smokers
than non-smokers, and heart disease may be more likely among the obese.

The goal of this analysis is to use patient data to estimate the average
medical
care expenses for such population segments. These estimates can be used to
create
actuarial tables that set the price of yearly premiums higher or lower,
```

```
depending on the expected treatment costs.'

###
------------------------------------------------------------------------
# The insurance data set has 1338 observations of 7 variables.
# We will use this file to predict the medical expenses.

# VARIABLE DESCRIPTIONS:
#age:       age in years
#sex:       gender
#bmi:       body mass index
#children: how many children do they have?
#smoker:   do they smoke?
#region:   geographic region
#expenses: yearly medical expenses
###
------------------------------------------------------------------------

# 1. Import the datadet
insurance <- read.csv(file = "insurance.csv", stringsAsFactors = FALSE)

# 2. str() shows the structure of data
str(insurance)

# 3. summary() shows the mean and the five-number statistics indicating the
spread of each column's values
summary(insurance)

# 4. Change all categorical variables to factors
insurance$sex <- factor(insurance$sex)
insurance$smoker <- factor(insurance$smoker)
insurance$region <- factor(insurance$region)
str(insurance)
summary(insurance)

# 5. Partition the dataset: 70% for training, 30% for testing
library(caret)
set.seed(1)
train_index <- createDataPartition(insurance$expenses, p=0.7, list=FALSE)
datTrain2 <- insurance[train_index,]
datTest2 <- insurance[-train_index,]

# 6. Build svm models with default setting (in default setting, C=1)
svm_model6 = ksvm(expenses~.,data = datTrain2)#we are using same ksvm
#function as we did in classification. KSVM can be used for both
classificaiton,
#and numeric prediction.
#Question: how to determine whether its classification or numeric prediction?
#Answer: The function will look at your target variables (in this case
#expenses)- if target variable is a factor (IsBadBuy)then this function
#will build svm model for classification. If its numeric varialbe (expenses)
#then ksvm will build ksvm regression and conduct a numeric prediction.

# Make predictions on both training and testing sets
prediction_on_train6 <- predict(svm_model6, datTrain2)
prediction_on_test6 <- predict(svm_model6, datTest2)
```

```
# Generating evaluation metrics on both training and testing data
#note: We do not need CONFUSION MATRIX since we are doing numeric prediction.
mmetric(datTrain2$expenses,prediction_on_train6,metric=c("MAE","RMSE","MAPE","RAE"))
#Also, we dont have accuracy, precision, recall and F measures. Above and
below
#are evaluation metrics for numeric predictions.
mmetric(datTest2$expenses,prediction_on_test6,metric=c("MAE","RMSE","MAPE","RAE"))

# 7. Build svm model with C=5
svm_model7 = ksvm(expenses~.,data = datTrain2, C=5)

# Make predictions on both training and tessting sets
prediction_on_train7 <- predict(svm_model7, datTrain2)
prediction_on_test7 <- predict(svm_model7, datTest2)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain2$expenses,prediction_on_train7,metric=c("MAE","RMSE","MAPE","RAE"))
mmetric(datTest2$expenses,prediction_on_test7,metric=c("MAE","RMSE","MAPE","RAE"))


# 8. Build svm model with C=10
svm_model8 = ksvm(expenses~.,data = datTrain2, C=10)

# Make predictions on both training and tessting sets
prediction_on_train8 <- predict(svm_model8, datTrain2)
prediction_on_test8 <- predict(svm_model8, datTest2)


# Generating evaluation metrics on both training and testing data
mmetric(datTrain2$expenses,prediction_on_train8,metric=c("MAE","RMSE","MAPE","RAE"))
mmetric(datTest2$expenses,prediction_on_test8,metric=c("MAE","RMSE","MAPE","RAE"))



# Which C value provides the best performance?
##########Answer:
#Comparing testing performance of the three models (above) and checking which
#model provides lowest MAE, RMSE, MAPE, RAE, error numbers.
mmetric(datTest2$expenses,prediction_on_test6,metric=c("MAE","RMSE","MAPE","RAE"))#C=1
mmetric(datTest2$expenses,prediction_on_test7,metric=c("MAE","RMSE","MAPE","RAE"))#C=5
mmetric(datTest2$expenses,prediction_on_test8,metric=c("MAE","RMSE","MAPE","RAE"))#C=10
#######MOdel 2 (C=5)has the lowest error values on the testing data.
#Therefore, the model with C=5, second one, has the best performance.

# How is SVM performed compared to mean estimator?
####We should look at RAE on the testing data as it provides the model's
#effectiveness compared to a mean estimator.
########Answer:
#Based on RAE.
#SVM Model 6(C=1) and 8(C=10)make about 31% of the errors that are made by
#the mean estimator. Model 7 (C=5), best performance, makes about 30% of the
#errors made by mean estimator.
#All three of these models are much more effective compared to the mean
#estimator, since they make considerably less errors than the mean estimator.
```

```
# Which evaluation metric tells you the percentage error?
#ANSWER####   MAPE evaluation metrics gives the PERCENTAGE ERROR.

# Assume that you will lose each dollar your models prediction misses due to
#an over-estimation or under-estimation. Which evaluation metric you should
use?
######ANSWER:
#We should use MAE since it helps to measures an average error. It is
basically
#an average of the difference of predicted patient expenses (in this case)and
#the actual expenses. That is its an average error of predicting expenses. So,
#for model 7, MAE is 2713 so we will lose $2713 for each error.
mmetric(datTest2$expenses,prediction_on_test7,metric=c("MAE"))

# Assume that the penalty for an erroneous prediction increases with the
#difference between the actual and predicted values.
#Which evaluation metric you should use?
#####Answer:
#We would use RMSE (root mean squared error).
#It gives relative high weight to large errors. It means that in this
scenario,
#I would pick the model with lowest RMSE since it would decrease the chances
#of making large errors.Model7 outperforms the other two, since RMSE is
smaller.

#NOTE: RMSE will always be higher than RAE.

# 9. Build MLP model contains two hidden layers: 8 hidden nodes for the first
#layer, and 4 hidden nodes for the second layer. Set N = 100
mlp_model10 <-  MLP(expenses~.,data=datTrain2, control = Weka_control(N=100,
H='8, 4'))
summary(mlp_model10)

# Make predictions on both training and testing sets
prediction_on_train10 <- predict(mlp_model10, datTrain2)
prediction_on_test10 <- predict(mlp_model10, datTest2)

# Generating evaluation metrics on both training and testing data
mmetric(datTrain2$expenses,prediction_on_train10,metric=c("MAE","RMSE","MAPE","RAE"))
mmetric(datTest2$expenses,prediction_on_test10,metric=c("MAE","RMSE","MAPE","RAE"))


# 10. 3-fold cv on insurance data with SVM method
# Set up cv parameters
# df: identifies the whole data set by its name
# target: identifies the target variable by its column index in df
# nFolds: indicates the number of folds for cv
# seedVal: carries the seed value for random sampling of instances when
creating folds
# prediction_method: indicates the prediction method - e.g., lm
# metric_list: is a list of evaluation metrics that mmetric should generate
cv_function_train5 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
```

```
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)#note 2 cross validation functions,
    #one on training one on testing
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,train_input)
    return(mmetric(train_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
# Testing performance for cross validation
cv_function_test5 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,test_input)
    return(mmetric(test_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
df= insurance
target = 7
nFolds = 3
```

```
seedval = 1
prediction_method = ksvm
metrics_list = c("MAE","RMSE","MAPE","RAE")
cv_function_train5(df, target, nFolds, seedval, prediction_method,
metrics_list)
cv_function_test5 (df, target, nFolds, seedval, prediction_method,
metrics_list)


# 11. 3-fold cv on insurance data with MLP method
# Set up cv parameters
# df: identifies the whole data set by its name
# target: identifies the target variable by its column index in df
# nFolds: indicates the number of folds for cv
# seedVal: carries the seed value for random sampling of instances when
creating folds
# prediction_method: indicates the prediction method - e.g., lm
# metric_list: is a list of evaluation metrics that mmetric should generate
cv_function_train6 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)#note 2 cross validation functions,
    #one on training one on testing
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,train_input)
    return(mmetric(train_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
# Testing performance for cross validation
cv_function_test6 <- function(df, target, nFolds, seedVal, prediction_method,
metrics_list)
{
  # create folds
  set.seed(seedVal)
  folds = createFolds(df[,target],nFolds)
  # perform cross validation
  cv_results <- lapply(folds, function(x)
```

```
  {
    test_target <- df[x,target]
    test_input  <- df[x,-target]

    train_target <- df[-x,target]
    train_input <- df[-x,-target]

    prediction_model <- prediction_method(train_target~.,train_input)
    pred<- predict(prediction_model,test_input)
    return(mmetric(test_target,pred,metrics_list))
  })
  # generate means and sds and show cv results, means and sds using kable
  cv_results_m <- as.matrix(as.data.frame(cv_results))
  cv_mean<- as.matrix(rowMeans(cv_results_m))
  cv_sd <- as.matrix(rowSds(cv_results_m))
  colnames(cv_mean) <- "Mean"
  colnames(cv_sd) <- "Sd"
  cv_all <- cbind(cv_results_m, cv_mean, cv_sd)
  kable(t(cv_all),digits=2)
}
df= insurance
target = 7
nFolds = 3
seedval = 1
prediction_method = MLP
metrics_list = c("MAE","RMSE","MAPE","RAE")
cv_function_train6(df, target, nFolds, seedval, prediction_method,
metrics_list)
cv_function_test6 (df, target, nFolds, seedval, prediction_method,
metrics_list)
```