Classification of Fitness of Drive for Drivers with Multiple Sclerosis

Yi Cheng, Wenqing He

August 10, 2017

1 Introduction

Multiple Sclerosis (MS) is a disease of the central nervous system and can cause symptoms that may affect MS patients' ability to drive. One of the most common symptoms is cognitive impairment (CI) which occurs in roughly 50% of MS patients[1]. The Minimal Assessment of Cognitive Function in MS (MACFIMS) is a valid approach to test CI with a battery of 12 tests in 8 different types, see Appendix A[4].

Apparently drivers with MS are more likely to cause traffic accidents if their cognitive functions are impaired. Intuitively MS patients with severe CI symptoms should be limited to drive for the sake of traffic safety. Meanwhile, patients with mild symptoms should be allowed to drive so that their living quality can be promised. The problem is how to distinguish between the two groups wisely. It is hypothesized that the Symbol Digit Modalities Test (sdo), Paced Auditory Serial Addition Test (rp2, rp3), Delis - Kaplan Executive Function System Sorting Test (cccs, ccds) and the Brief Visuospatial Memory Test - Revised (bvtd, bvtt) will be the best indicators of whether a driver is fit to drive.

The objective of this study is to find the best classification model with the best predictive power to decide which patients are fit to drive and which are not.

2 Preliminary Analysis

The original data set contains 38 observations with 12 independent variables and one binary dependent variable. Each variable represents a certain MACFIMS test result as shown in Appendix A. The response variable represents whether the patient passes driving test or not. One observation without response variable was removed. Another observation has missing value in variable **scw**. Due to small sample size, whether to remove the observation depends on the importance of the **scw** variable. Throughout the study models are

trained and compared with both 36 observations (without **scw**) and 37 observations (with **scw**). Of all the 37 observations, 8 observations are positive cases (Fail) and 29 observations are negative cases (Pass). The event rate is 21.62%.

Since some MACFIMS tests are similar, pairwise correlations between variables are detected. If we set the criterion as correlation coefficient ≥ 0.7 to be considered as correlated, the detected correlated variable pairs are shown in Table 1.

Pairs	Correlation Coefficient
ccds : cccs	0.8947
bvtt : bvtd	0.8922
cvtd : cvtt	0.7896
scw : cvtt	0.7485
rp2 : rp3	0.7144

Table 1: Correlated variable pairs

As an attempt to visually examine the test score difference between two groups, Figure 1 shows a series of box plots for all test results separated by two groups. From the plot, only cwt, bvtt and bvtd tests reveal obvious score differences between fail group and pass group with median score in the pass group outside the interquartile range of the fail group.

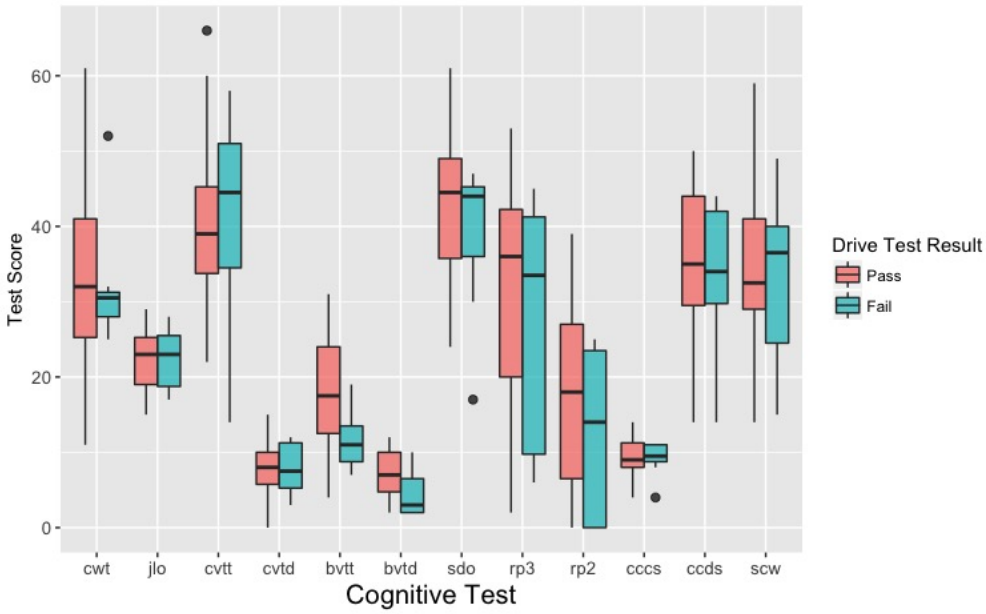

Figure 1: Boxplots of MACFIMS test scores for pass and fail groups

Further Wilcoxon test is used to examine the test score difference between two groups. The result is shown in Table 2. The most significant test is the bvtt test (p-value = 0.0455). Both box plot and Wilcoxon test suggested that test **bvtt** is potentially the most useful predictor in constructing models.

Test Variable	Wilcoxon Test P-Value
cwt	0.8490
jlo	0.8633
cvtt	0.5171
cvtd	0.9390
bvtt	0.0455
bvtd	0.0629
sdo	0.3804
rp3	0.5548
rp2	0.2577
cccs	0.8020
ccds	0.6749
scw	0.9848

Table 2: Wilcoxon test for MACFIMS test score difference

3 Methods

In order to find the best classification model, different classification methods as well as different variable combinations are tried and their performances compared. The first and second combinations are 36 observations with all 12 variables and 37 observations with all variables except **scw** respectively. The purpose is to examine the importance of **scw** variable and, if possible, use larger sample size to build models to achieve better results. The third combination is made due to correlation problem, since it is uncertain whether any of the classification method would be sensitive to variable correlations. Within each correlated variable pairs, variable with larger Wilcoxon test p-value is thrown away. The final combination contains 7 variables. The last combination is constructed to verify the original hypothesis, stating that 4 tests are the best indicators of a fit or unfit drivers.

A previous study[5] has applied logistic model to the data set with 33 observaitons. The final model was obtained by fitting data with four hypothesized main inputs (bvtt, sdo, rp2, ccds) and all two-way and three-way interactions and conducting bidirectional elimination stepwise variable selection. As an attempt to the model prediction accuracy, this study builds logistic model using same method with updated 37 observations. The model perfomance is compared with other methods as well.

3.1 Classification Methods

The first attempt is made on simple Decision Tree Model. For each variable combination, a single tree is built. A classification tree is built top-down from a root node and involves splitting the data into subsets that contain instances with similar labels. The rule of tree splitting is to minimize the classification error rate. The algorithm is usually employed with greedy strategy that grows a decision tree by making a series of optimum decisions about which variable to use for tree split. When a tree is built, each leaf node is assigned a class label by the most commonly occurring class of observations in that node. Each observations are predicted as belonging to that class.

After applying tree model, it is often useful to apply Tree Bootstrap Aggregating (Bagging) to reduce the variance of single tree model. In tree bagging, B different bootstrapped data sets are generated with replacement. Then for each bootstrapped data set, a simple decision tree is built in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. Then the prediction for a given point x can be obtained by average all the predictions[3]:

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x).$$

The idea for classification tree bagging is similar, the overall prediction of an observation is obtained by taking the most commonly occurring class among the B predictions. By applying tree bagging, for each variable combination, the optimum number of trees to build is tuned.

Intuitively after tree bagging, Random Forest method is applied. Random forest method resembles tree bagging in its "bagging" idea but provides an improvement over tree bagging in that it can decorrelates the trees. In random forest, each time a split in a tree is considered, a random selection of m predictors is chosen as split candidates from the full set of p predictors. Each split is allowed to use only one of the m predictors. The optimum number of variable limits is tuned as well.

Eventually Support Vector Machine (SVM) is applied in an attempt to improve model performance. SVM method tries to find a hyperplane that separates the classes in feature space and makes the biggest margin(M)

between the two classes. The hyperplance has the form:

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \cdots + \beta_p X_p = 0$$

When the data points are not separable, a solution can alwasys be found by softening the margin and letting some points standing on the wrong sides. This introduces cost parameter(C) into the model. In general, the method attempts to solve the following problem:

$$\begin{aligned} &\underset{\beta_0,\beta_1,\ldots,\beta_p,\epsilon_1,\ldots,\epsilon_n}{\text{maximize}} \quad M \\ &\text{subject to } \sum_{j=1}^{p} \beta_j^2 = 1, \\ &\qquad y_i(\beta_0 + \beta_1 x_{i1} + \beta_2 x_{i,2} + \cdots + \beta_p x_{i,p}) \geq M(1-\epsilon_i), \\ &\qquad \epsilon_i \geq 0, \sum_{i=1}^{n} \leq C, \end{aligned}$$

After comparing linear, polynomial and radial kernels in terms of prediction accuracy, radial basis function kernel is selected to fit SVM models. For each variable combination, the parameters are tuned by using grid search to achieve the highest prediction accuracy. The predicted class of each observation is decided by the sign of discriminant function.

3.2 Evaluation Metrics

Training AUC The area under the curve (AUC) is often used as a quality summary of a ROC curve. Higher AUC score indicates better classification performance. In this study training AUC is obtained by predicting training data originally used to train the model. For all the classification methods except SVM, the ROC score rule is based on the predicted class probabilities. The ROC score rule for SVM model is based on the decision values obtained from discriminant function.

Testing AUC Apparently training AUC is not a useful indicator of prediction accuracy, especially when a model faces overfitting problem. Hence additionally in this study testing AUC is calculated. It is obtained by calculating cross validated AUCs (K=5), repeating with 5 different seeds and taking average. Considering the imbalanced data set with only 8 positive cases, stratified cross validation is used where cases and controls are assigned in proportion to the overall case and control ratio within each fold .

Misclassification Rate The misclassification rate is calculated for each method and each variable combination. It is the sum of misclassified observations divided by all observations.

Cross - Validated Error Cross-validated error is often deemed as a common choice to estimate model test error. Similar with testing AUC, in this study Stratified cross-validation is used because of the data imbalance problem. For better comparison among different models, three different cross-validated errors are calculated, namely 5-fold, 10-fold and leave-one-out error rate. The errors are obtained by repeating with 5 different seeds and taking average.

For each model, in addition to cross-validated errors, confusion matrix obtained from leave-one-out cross validation is presented as well for model comparison purpose.

4 Results

4.1 Logistic Model

Metrics	Value
Training AUC	0.9009
Testing AUC	0.5416
Training Misclassification Rate	0.1621
5-Fold CV Error	0.3943
10-Fold CV Error	0.3800
LOOCV	0.3784
Cutoff	0.2717

Table 3: Performance summary for logistic model

	Actual Fail	Actual Pass
Predicted Fail	1	7
Predicted Pass	7	22

Table 4: Confusion matrix for logistic model obtained from LOOCV

4.2 Decision Tree

The model fitting began with all 12 variables to see if variable **scw** is important in constructing models. It turned out that **scw** variable is not useful since it is not used as split variables for tree building, as seen in Figure 2. Therefore tree models are trained with other three variable combinations for comparison purpose.

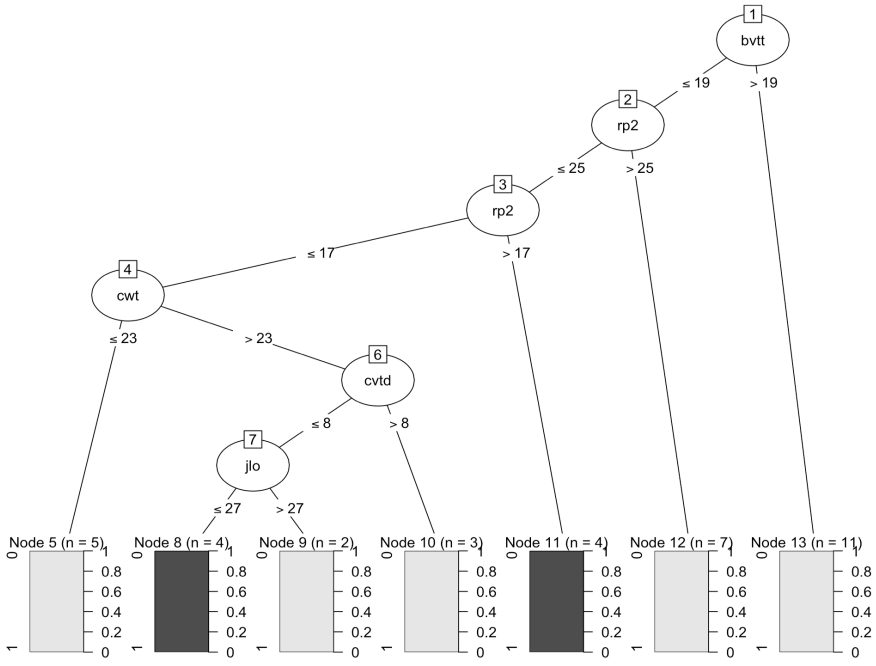

Figure 2: Tree structure using 36 observations

Decision Tree	11 variables	7 variables	4 variables
Training AUC	1.0000	0.9957	0.9138
Testing AUC	0.5898	0.6052	0.5852
Training Misclassification Rate	0.0000	0.0270	0.1081
5-Fold CV Error	0.3171	0.3195	0.2586
10-Fold CV Error	0.3067	0.2900	0.2650
LOOCV	0.3243	0.3514	0.2432

Table 5: Performance summary for tree models

11 variables	Actual Fail	Actual Pass
Predicted Fail	2	9
Predicted Pass	6	20

7 variables	Actual Fail	Actual Pass
Predicted Fail	2	8
Predicted Pass	6	21

4 variables	Actual Fail	Actual Pass
Predicted Fail	0	2
Predicted Pass	8	27

Table 6: Confusion matrices for tree models obtained from LOOCV

4.3 Tree Bagging

Tree Bagging	12 variables	11 variables	7 variables	4 variables
Training AUC	1.0000	1.0000	1.0000	1.0000
Testing AUC	0.6313	0.6377	0.7173	0.6790
Training Misclassification Rate	0.0000	0.0000	0.0000	0.0000
5-Fold CV Error	0.3204	0.2848	0.3081	0.3207
10-Fold CV Error	0.3217	0.2983	0.2883	0.3214
LOOCV	0.3333	0.3243	0.3243	0.3243

Table 7: Performance summary for tree bagging models

12 varaibles	Act.Fail	Act.Pass
Pre.Fail	2	3
Pre.Pass	6	25

11 varaibles	Act.Fail	Act.Pass
Pre.Fail	1	6
Pre.Pass	7	23

7 varaibles	Act.Fail	Act.Pass
Pre.Fail	1	3
Pre.Pass	7	26

4 varaibles	Act.Fail	Act.Pass
Pre.Fail	1	4
Pre.Pass	7	25

Table 8: Confusion matrices for tree bagging models obtained from LOOCV

4.4 Random Forest

Random Forest	12 variables	11 variables	7 variables	4 variables
Training AUC	1.0000	1.0000	1.0000	1.0000
Testing AUC	0.6957	0.7167	0.7067	0.7063
Training Misclassification Rate	0.0000	0.0000	0.0000	0.0000
5-Fold CV Error	0.3143	0.2598	0.2293	0.2805
10-Fold CV Error	0.3667	0.2583	0.2200	0.2833
LOOCV	0.3143	0.2703	0.2162	0.2973

Table 9: Performance summary for random forest models

12 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	2
Pre.Pass	8	26

11 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	1
Pre.Pass	8	28

7 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	2
Pre.Pass	8	27

4 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	3
Pre.Pass	8	26

Table 10: Confusion matrices for random forest models obtained from LOOCV

4.5 Support Vector Machine

SVM	12 variables	11 variables	7 variables	4 variables
Training AUC	0.5893	0.6466	0.7112	0.5259
Testing AUC	0.7214	0.7173	0.7162	0.7108
Training Misclassification Rate	0.2220	0.2162	0.2162	0.2162
5-Fold CV Error	0.2250	0.2129	0.2129	0.2129
10-Fold CV Error	0.2250	0.2083	0.2083	0.2083
LOOCV	0.2222	0.2162	0.2162	0.2162

Table 11: Performance summary for support vector machine models

12 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	0
Pre.Pass	8	28

11/7/4 varaibles	Act.Fail	Act.Pass
Pre.Fail	0	0
Pre.Pass	8	29

Table 12: Confusion matrices for support vector machine models obtained from LOOCV

4.6 Discussion

The logistic model has the highest cross validation error among all methods. This suggests that the previous classification method is weak in prediction accuracy and is not the ideal model for this study. Examining other model results carefully, clearly each method and model has its own concerns. Decision tree models have overfitting problems. The training AUCs are high while respective testing AUCs are the lowest among all methods. The predictive power is week for all variable combinations. The use of tree bagging and random forest does not improve prediction accuracy much. Both methods suffered from severe overfitting problems, since their training AUCs equal 1. The random forest models do show improvement upon tree bagging models in terms of predictiion accuracy. But the result is still not ideal. Support vector machine models seems to have improved prediction accuracy but their misclassification rates are dangerously high. As a matter of fact, SVM models are insensitive. From the confusion matrices obtained from leave-one-out cross validation, the recall are zero for all SVM models built. Due to imbalanced data, the event rate is only 0.2162 (0.2222 when 36 observations are used). This means if the recall is zero, a model is not a good classifier even if its cross-validated errors are the lowest among all methods.

5 Model Improvement

5.1 Methods

The problem goes that when a data set is imbalanced, the classifier tends to only predict the majority class data. The features of the minority class are treated as noise and are often ignored. Hence there is a high probability of misclassification of the minority class as compared to the majority class. Two approaches are considered together in an attempt to improve SVM model in terms of sensitivity. One idea is to use class-weighted SVM models. It means we put higher penalty on misclassifying minority class so that the minority class is less likely to be treated as noise. The second idea is to conduct variable selections for SVM models based on some criteria. Different criteria are tried and respective models are built and compared to find the best SVM model.

The criteria used include the following:

Variable Importance In random forest models, variable importance (VI) can be calculated for each variable based on the rule of mean decrease in accuaracy. This means variables with higher variable importance contribute more accuracy to the model than variables with lower variable importance. In this study both top 2 and top 3 variables are picked to fit the class-weighted SVM models.

The selected variable combinations are **cwt + jlo** and **cwt + jlo + cvtt**.

Tree Splits The variables used for decision tree splits are considered deterministic for data classification. Therefore it can be argued that the the first several variables used as tree splits are useful in constructing models. In this study both top 2 and top 3 variables are picked to fit the class-weighted SVM models.

The selected combinations are **bvtt + rp2** and **cwt + bvtt + rp2**.

Change of Testing AUC Variables are selected based on metric of change of testing AUC with and without removing each variable[2]. The importance of each variable can be determined by considering how the performance is influenced without that variable. If removing a variable deteriorates the performance, the variable is considered important. On the other hand, if the performance improves after removing a variable, the variable should be removed. The seletion process starts with all 11 variables and gradually removes unimportant variables till the testing AUC cannot be further improved. Testing AUC is obtained by calculating cross validated(K=5) AUCs, repeating with 5 different seeds and taking average.

The selected combination is **cwt + jlo + cvtt + ccds**.

Highest Testing AUC with bvtt Previously by conducting two-sample Wilcoxon test and visually examining boxplots, it is very clear that **bvtt** is potentially the most useful variable for doing classification. Therefore the method picks whichever one or two variables that, combined with **bvtt**, has the highest testing AUC.

The selected combinations are **cwt + bvtt** and **cwt + bvtt + rp3**.

5.2 Results

The model performance is summarized in Table 13.

Variable combinations	I	II	III	IV	V	VI	VII
Training AUC	1.0000	0.5603	0.7328	0.8922	0.5086	0.9870	0.9526
Testing AUC	0.7900	0.7867	0.7633	0.7800	0.8147	0.7987	0.8240
Training Misclassification Rate	0.0000	0.2162	0.2162	0.1892	0.2162	0.0540	0.0811
5-Fold CV Error	0.2626	0.2130	0.2130	0.2127	0.2130	0.2063	0.1950
10-Fold CV Error	0.2117	0.2067	0.2067	0.2135	0.2067	0.1976	0.1905
LOOCV	0.2324	0.2162	0.2162	0.2188	0.2162	0.2046	0.1866

- I : Top 2 variables with highest variable importance : cwt + jlo
- II : Top 3 variables with highest variable importance : cwt + jlo + cvtt
- III : First 2 variables used as tree splits : bvtt + rp2
- IV : First 3 variables used as tree splits : bvtt + rp2 + cwt
- V : Variables selected by change of testing AUC : cwt + jlo + cvtt + ccds
- VI : One variable combined with bvtt having the highest testing AUC : cwt + bvtt
- VII : Two variables combined with bvtt having the highest testing AUC : cwt + bvtt + rp3

Table 13: Performance summary for class-weighted SVM with different variable selection methods

From Table 13 it is clear that the best two SVM model are obtained by method of searching variables which has the highest test AUC when combined with bvtt. The best model has variables: **cwt + bvtt + rp3**. It has the highest Testing AUC and lowest cross validated errors, which are lower than the event rate(21.62%) as well.

Table 14 shows the confusion matrix obtained from leave-one-out cross validation for the best model. Compared with SVM models that do not use class - weighted method, the class-weighted SVM model can handle the data imbalance problem very well. For the best SVM model, precision = 0.545, sensitivity = 0.750.

	Act.Fail	Act.Pass
Pre.Fail	6	5
Pre.Pass	2	24

Table 14: Confusion matrix for the best SVM model obtained from LOOCV

In an attempt to understand the roles each variable plays in the classifier, 2D and 3D plots are plotted by using the best two SVM models, see Figure 3 - 4. Whether a MS patience is qualified enough to drive can be understood by visually examining the decision boundary in plots. Clearly even though these two models have so far the best model performance in terms of predictive power, and that the cross validated error is lower than the event rate, it is hard to interpret the decision boundary from a practical point of view. With only 37 valid observations, the data points in the plots are scattered allover and unable to show clustered pattern as expected. No doubt if more data points are available, the decision boundary as well as the model itself will become more interpretable.

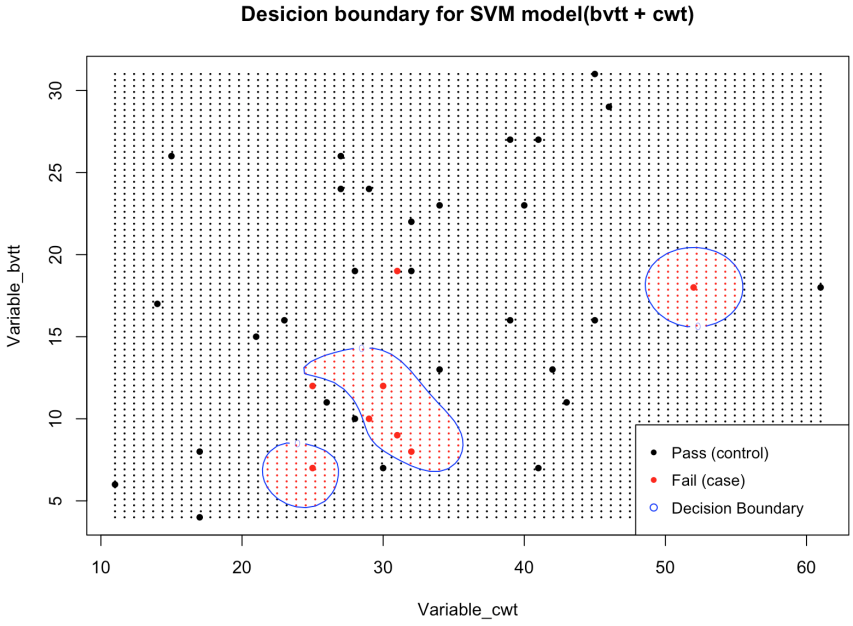

Figure 3: Decision boundary for SVM model with variable: cwt, bvtt

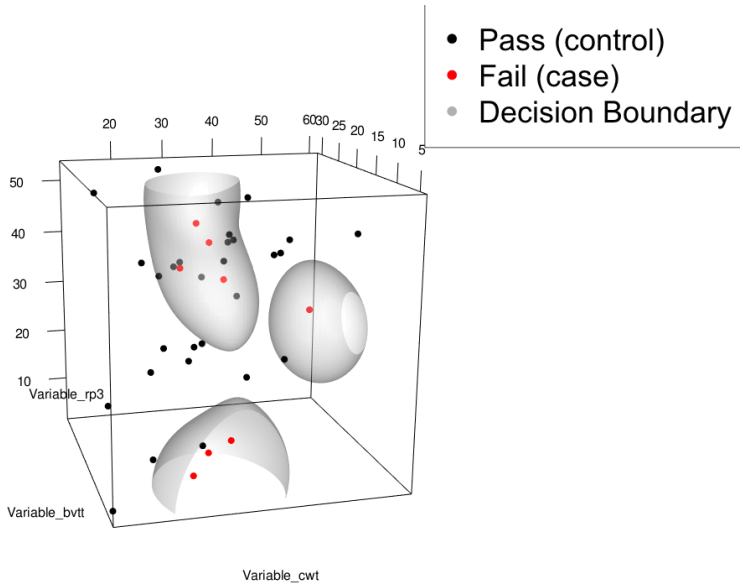

Figure 4: Decision boundary for the best SVM model with variable: cwt, bvtt, rp3

6 Conclusions and Limitations

The analysis begins with studying data itself. By examining box plot and conducting two sample Wilcoxon test, it is clear that there is not a big difference in test scores between pass group and fail group. The only potentially useful variable is bvtt which relates to patients' learning ability and memory. After correlation detection, four different variable combinations are tried on four classification methods to find the model with best prediction accuracy. The results suggest that decision tree model, tree bagging and random forest, despite their low misclassification rates, are not ideal models in terms of predictive power. The support vector machine method has better prediction accuracy but suffers from data imbalance problem. Variable selection methods and class-weighted SVM method are then applied. Eventually the best model is found to contain variables **cwt**, **bvtt** and **rp3** with cross-validated leave-one-out error at 18.66% and testing AUC at 0.8240.

Due to small sample size and data imbalance problem, the study finds it hard to improve the cross-validated errors further. Considering the fact that the event rate is 21.62%, surely the achieved result still has room for improvement. If more MS patients observations are available, by using the same technique, it is very likely that a stronger classifier with higher prediction accuracy can be trained. Another concern is that the decision boundary obtained by fitting model with only 37 observations is too abstract to be interpretable. This situation is believed to get improved if more observations are available and more advanced variable

selection methods are used.

References

[1] Ralph HB Benedict, John DeLuca, Glenn Phillips, Nicholas LaRocca, Lynn D Hudson, Richard Rudick, and Multiple Sclerosis Outcome Assessments Consortium. Validity of the symbol digit modalities test as a cognition performance outcome measure for multiple sclerosis. *Multiple Sclerosis Journal*, 23(5):721–733, 2017.

[2] Yin-Wen Chang and Chih-Jen Lin. Feature ranking using linear svm. In *Causation and Prediction Challenge*, pages 53–64, 2008.

[3] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[4] Yelena Lapshin. *Detecting Cognitive Dysfunction in Multiple Sclerosis: Assessing the Validity of a Computer Generated Battery.* PhD thesis, 2013.

[5] Robert Taylor and Wenqing He. Fitness to drive prediction of drivers with multiple sclerosis. unpublished master report, Western University, 2016.

Appendices

A

Table 15: Variables used in the study from the MACFIMS test

Variable: cwt	
Test:	Controlled Oral Word Association Test
Domain:	Language and other domains
Variable: jlo	
Test:	Judgment of Line Orientation Test
Domain:	Visuospatial ability
Variable: cvtt	
Test:	California Verbal Learning Test - II - Total Recall
Domain:	Learning and memory
Variable: cvtd	
Test:	California Verbal Learning Test - II - Delayed Recall
Domain:	Learning and memory
Variable: bvtt	
Test:	Brief Visuospatial Memory Test - Revised - Total Recall
Domain:	Learning and memory
Variable: bvtd	
Test:	Brief Visuospatial Memory Test - Revised - Delayed Recall
Domain:	Learning and memory
Variable: sdo	
Test:	Symbol Digit Modalities Test
Domain:	Processing speed/working memory
Variable: rp3	

Test:	Paced Auditory Serial Addition Test - 3 seconds
Domain:	Processing speed/working memory
Variable: rp2	
Test:	Paced Auditory Serial Addition Test - 2 seconds
Domain:	Processing speed/working memory
Variable: cccs	
Test:	Delis-Kaplan Executive Function System Card Sorting Test - Sorting Score
Domain:	Executive function
Variable: ccds	
Test:	Delis-Kaplan Executive Function System Card Sorting Test - Description Score
Domain:	Executive function
Variable: scw	
Test:	Stroop Colour-Word Test
Domain:	Selective Attention and Cognitive Flexibility

B R code

```
# Data Input & Cleaning
MSdata <- read.csv("/Users/chengyi/Desktop/drivers/MS\ Driving.csv", header = TRUE)
MSdata.37 <- MSdata[-2,] # delete obs without response
#add pass fail column
test <- as.factor(MSdata.37[,"GRS.2X.CDRS"]==1)
levels(test)[1] <- 0 #pass
levels(test)[2] <- 1 #fail
MSdata.37$Test <- test
#pull test scores out
input <- MSdata.37[,c("cwt","jlo","cvtt","cvtd",
                      "bvtt","bvtd","sdo","rp3","rp2","cccs","ccds","scw")]
driver <- cbind(test, input)
# dataframe containing hypothsized useful variables
driver_lo <- driver[c(1,6,8,10,12)]
# dataframe containing 11 varaibes (no scw)
driver_scw <- driver[,-13]
# dataframe with 36 obs
driver_36 <- driver[-22,]
k <- c(5,10,37) # CV fold
```

```
k36 <- c(5,10,36)
CV_Folds5 <- createMultiFolds(y, k = 5, times = 5)
CV_Folds10 <- createMultiFolds(y, k = 10, times = 5)
seed <- 45
see <- seq(100,300,10)
see_red <- seq(100,300,50)

# Detect Correlation
library(dplyr)
library(reshape2)
cors <- as.matrix(cor(driver_36[,-1]))
rs <- arrange(melt(cors), -abs(value))
topcor <- dplyr::filter(rs, value > 0.7) # throw away: cvtd, bvtd, rp3, cccs, scw
# dataframe containing 7 variables
driver_cor <- driver[,-c(5,7,9,11,13)]

# Boxplot
```{r,results="asis", echo=FALSE, warning=FALSE,message=FALSE}
library(ggplot2)
library(reshape2)
data.melt <- melt(driver_36,id.var="test")
ggplot(data.melt,aes(x=variable,y=value)) +
  geom_boxplot(aes(fill=test),alpha=0.7, outlier.colour = "black",outlier.size=2) +
  ylab("Test Score") +
  xlab("Cognitive Test") +
  scale_fill_discrete(name="Drive Test Result",breaks=c("0","1"),labels=c("Pass","Fail")) +
  theme(axis.text=element_text(size=10),axis.title.x=element_text(size=15))

# Use two-sample Wilcoxon test
library(xtable)
x <- c(2:13)
w.test <- lapply(x,FUN=function(k)
          {wilcox.test(driver_36[,k]~driver_36[,1],alternative="two.sided")})
pval.w <- sapply(1:12,FUN=function(k){w.test[[k]]$p.value})
test.names <- c("cwt","jlo","cvtt","cvtd","bvtt","bvtd","sdo","rp3","rp2","cccs","ccds","scw")
t.table <- as.data.frame(cbind(test.names, round(pval.w,4)))
names(t.table) <- c("Test", "Wilcoxon test p-value")
x.t.table <- xtable(t.table,
                    caption="Wilcoxon test for MACFIMS test score difference.")
print(x.t.table,include.rownames=FALSE,table.placement="H")

# Define Functions
# functions for LOOCV confusion tables
 CVError_logi_loo <- function(daTa, K, seed, cf){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.pre <- numeric(K)
  for (k in 1:K) {
    fit <- glm(test~ bvtt + sdo + rp2 + ccds + bvtt:rp2 + bvtt:ccds + sdo:rp2 +
    sdo:ccds + rp2:ccds + sdo:rp2:ccds, family = binomial, data = daTa[-index[[k]],])
    pred <- ifelse(predict(fit, newdata = daTa[index[[k]],], type="response") > cf, 1, 0)
    all.pre[k] <- pred
```
```

```
 }
 all.pre
}
CVError_tree_loo <- function(daTa, K){
 library(caret)
 index <- createFolds(daTa$test, k = K)
 all.pre <- numeric(K)
 for (k in 1:K) {
   library(C50)
   fit <- C5.0(test~., data=daTa[-index[[k]],])
   pred <- predict.C5.0(fit, newdata = daTa[index[[k]],])
   pred <- ifelse(pred == 0, 0, 1)
   all.pre[k] <- pred
 }
 all.pre
}
CVError_bag_loo <- function(daTa, K, nbag){
 library(caret)
 index <- createFolds(daTa$test, k = K)
 all.pre <- numeric(K)
 for (k in 1:K) {
   library(ipred)
   fit <- bagging(test~., data=daTa[-index[[k]],], nbagg = nbag)
   pred <- predict(fit, newdata = daTa[index[[k]],])
   pred <- ifelse(pred == 0, 0, 1)
   all.pre[k] <- pred
 }
 all.pre
}
CVError_rf_loo <- function(daTa, K){
 library(caret)
 index <- createFolds(daTa$test, k = K)
 all.pre <- numeric(K)
 for (k in 1:K) {
   library(e1071)
   library(randomForest)
   fit <- best.randomForest(test~., data=daTa[-index[[k]],],
                            mtry = c(1:2), ntree = 100*seq(5,10,1))
   pred <- predict(fit, newdata = daTa[index[[k]],])
   pred <- ifelse(pred == 0, 0, 1)
   all.pre[k] <- pred
 }
 all.pre
}
CVError_svm_loo <- function(daTa, K){
 library(caret)
 index <- createFolds(daTa$test, k = K)
 all.pre <- numeric(K)
 for (k in 1:K) {
   library(e1071)
   fit <- svm(test~., data=daTa[-index[[k]],], scale=TRUE,
              kernel="radial", gamma =3.051758e-05, cost = 0.03125)
   pred <- predict(fit, newdata = daTa[index[[k]],])
```

```
    pred <- ifelse(pred == 0, 0, 1)
    all.pre[k] <- pred
  }
  all.pre
 }
 CVError_wsvm_loo <- function(daTa, K, seed){
  library(caret)
  index <- createFolds(daTa$test, k = K, seed)
  all.pre <- numeric(K)
  library(e1071)
  set.seed(seed)
  fit.o <- best.svm(test~., data=daTa, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)),
                    class.weights= c("0" = 1, "1" = 3))
  ga <- fit.o$gamma
  co <- fit.o$cost
  library(e1071)
  for (k in 1:K) {
  fit <- svm(test~., data = daTa[-index[[k]],],scale=TRUE,
             kernel="radial",gamma = ga, cost = co,
             class.weights= c("0" = 1, "1" = 3))
  pred <- predict(fit, newdata = daTa[index[[k]],])
  pred <- ifelse(pred == 0, 0, 1)
  all.pre[k] <- pred
  }
  all.pre
}
# SVM_stratified CV error, weighted
CVError_svm_w <- function(daTa, K, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.err <- numeric(K)
  library(e1071)
  set.seed(seed)
  fit.o <- best.svm(test~., data=daTa, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)),
                    class.weights= c("0" = 1, "1" = 3))
  ga <- fit.o$gamma
  co <- fit.o$cost
  library(e1071)
  for (k in 1:K) {
  fit <- svm(test~., data = daTa[-index[[k]],],scale=TRUE, kernel="radial",
                  gamma = ga, cost = co,
                  class.weights= c("0" = 1, "1" = 3))
  pred <- predict(fit, daTa[index[[k]],])
  pred <- as.factor(pred)
  levels(pred) <- c("0","1")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  cm <- table(pred, true)
  err <- 1-(cm[1,1]+cm[2,2])/sum(cm)
  all.err[k] <- err
```

```
  }
  mean(all.err)
}
CVError_svm <- function(daTa, K, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.err <- numeric(K)
  library(e1071)
  set.seed(seed)
  fit.o <- best.svm(test~., data=daTa, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)))
  ga <- fit.o$gamma
  co <- fit.o$cost
  library(e1071)
  for (k in 1:K) {
  fit <- svm(test~., data = daTa[-index[[k]],],scale=TRUE, kernel="radial",
                  gamma = ga, cost = co)
  pred <- predict(fit, daTa[index[[k]],])
  pred <- as.factor(pred)
  levels(pred) <- c("0","1")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  cm <- table(pred, true)
  err <- 1-(cm[1,1]+cm[2,2])/sum(cm)
  all.err[k] <- err
  }
  mean(all.err)
}
CVError_svm_s <- function(daTa, K, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.err <- numeric(K)
  library(DMwR)
  newdata <- SMOTE(test ~ ., daTa, perc.over = 200)
  library(e1071)
  set.seed(seed)
  fit.o <- best.svm(test~., data=newdata, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)))
  ga <- fit.o$gamma
  co <- fit.o$cost
  for (k in 1:K) {
  library(DMwR)
  newdatak <- SMOTE(test ~ ., daTa[-index[[k]],], perc.over = 200)
  library(e1071)
  fit <- svm(test~., data = newdatak, scale=TRUE, kernel="radial",
                  gamma = ga, cost = co)
  pred <- predict(fit, daTa[index[[k]],])
  pred <- as.factor(pred)
  levels(pred) <- c("0","1")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
```

```
  cm <- table(pred, true)
  err <- 1-(cm[1,1]+cm[2,2])/sum(cm)
  all.err[k] <- err
  }
  mean(all.err)
}
# SVM_AVG testing AUC, k=5
CVAUC_svm_w <- function(daTa, K=5, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  library(e1071)
  fit.o <- best.svm(test~., data=daTa, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)),
                    class.weights= c("0" = 1, "1" = 3))
  ga <- fit.o$gamma
  co <- fit.o$cost
  all.auc <- numeric(K)
  library(e1071)
  for (k in 1:K) {
  fit <- svm(test~., data = daTa[-index[[k]],],scale=TRUE, kernel="radial",
                  gamma = ga, cost = co, probability = TRUE,
                  class.weights= c("0" = 1, "1" = 3))
  pred <- predict(fit, daTa[index[[k]],], probability = TRUE, decision.values = TRUE)
  psvm <- attr(pred, "decision.values")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.svm <- roc(true~as.vector(psvm), smooth = FALSE))
  }
  mean(all.auc)
}
CVAUC_svm <- function(daTa, K=5, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  library(e1071)
  fit.o <- best.svm(test~., data=daTa, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)))
  ga <- fit.o$gamma
  co <- fit.o$cost
  all.auc <- numeric(K)
  library(e1071)
  for (k in 1:K) {
  fit <- svm(test~., data = daTa[-index[[k]],],scale=TRUE, kernel="radial",
                  gamma = ga, cost = co, probability = TRUE)
  pred <- predict(fit, daTa[index[[k]],], probability = TRUE, decision.values = TRUE)
  psvm <- attr(pred, "decision.values")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.svm <- roc(true~as.vector(psvm), smooth = FALSE))
  }
```

```
  mean(all.auc)
}
CVAUC_svm_s <- function(daTa, K=5, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  library(DMwR)
  newdata <- SMOTE(test ~ ., daTa, perc.over = 200)
  library(e1071)
  fit.o <- best.svm(test~., data=newdata, scale=TRUE, kernel="radial",
                    gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)))
  ga <- fit.o$gamma
  co <- fit.o$cost
  all.auc <- numeric(K)
  for (k in 1:K) {
    library(DMwR)
    newdatak <- SMOTE(test ~ ., daTa[-index[[k]],], perc.over = 200)
    library(e1071)
    fit <- svm(test~., data = newdatak, scale=TRUE, kernel="radial",
                  gamma = ga, cost = co)
  pred <- predict(fit, daTa[index[[k]],], probability = TRUE, decision.values = TRUE)
  psvm <- attr(pred, "decision.values")
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.svm <- roc(true~as.vector(psvm), smooth = FALSE))
  }
  mean(all.auc)
}
# RF_AVG testing AUC, k=5
CVAUC_rf <- function(daTa, K=5, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  vari <- ncol(daTa) - 1
  library(e1071)
  fit.o <- best.randomForest(test~., data = daTa,
                             mtry = c(1:vari), ntree = 100*seq(5,15,1))
  m <- fit.o$mtry
  n <- fit.o$ntree
  all.auc <- numeric(K)
  library(randomForest)
  for (k in 1:K) {
  fit <- randomForest(test~., data = daTa[-index[[k]],], mtry = m, ntree = n)
  prf <- predict(fit, daTa[index[[k]],], type = "prob")[,2]
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.rf <- roc(true~as.vector(prf), smooth = FALSE))
  }
  mean(all.auc)
}
# Tree_AVG testing AUC, k=5
```

```
CVAUC_tree <- function(daTa, K=5, seed){
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.auc <- numeric(K)
  library(C50)
  for (k in 1:K) {
  fit <- C5.0(test~., data = daTa[-index[[k]],], method = "class")
  prf <- predict(fit, daTa[index[[k]],], type = "prob")[,2]
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.tree <- roc(true~as.vector(prf), smooth = FALSE))
  }
  mean(all.auc)
}
# Bag_AVG testing AUC, k=5
CVAUC_bag <- function(daTa, K=5, t, seed) {
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.auc <- numeric(K)
  library(ipred)
  for (k in 1:K) {
  fit <- bagging(test~., data = daTa[-index[[k]],], nbagg = t, coob = FALSE)
  pbag <- predict(fit, daTa[index[[k]],], type = "prob")[,2]
  true <- as.factor(daTa$test[index[[k]]])
  levels(true) <- c("0","1")
  library(pROC)
  all.auc[k] <- auc(roc.bag <- roc(true~as.vector(pbag), smooth = FALSE))
  }
  mean(all.auc)
}
# Logit_AVG testing AUC, k=5
CVAUC_logi <- function(daTa, K=5, seed) {
  library(caret)
  set.seed(seed)
  index <- createFolds(daTa$test, k = K)
  all.auc <- numeric(K)
  for (k in 1:K) {
  fit <- glm(test~ bvtt + sdo + rp2 + ccds + bvtt:rp2 + bvtt:ccds + sdo:rp2 +
               sdo:ccds + rp2:ccds + sdo:rp2:ccds,
             family = binomial, data = daTa[-index[[k]],])
  library(ROCR)
  prob.main <- predict(fit, newdata= daTa[index[[k]],], type="response")
  pred.main <- prediction(prob.main, daTa$test[index[[k]]])
  perf.main <- performance(pred.main, measure = "tpr", x.measure = "fpr")
  all.auc[k] <- performance(pred.main, measure = "auc")@y.values[[1]]
  }
  mean(1-all.auc)
}
```

```r
#Logistic Model
# training AUC
library(ROCR)
prob.main <- predict(modfinal, newdata = driver, type="response")
pred.main <- prediction(prob.main, driver[,1])
perf.main <- performance(pred.main, measure = "tpr", x.measure = "fpr")
plot(perf.main)
performance(pred.main, measure = "auc")@y.values[[1]]
# Misclassification
pre <- ifelse(predict(modfinal, newdata= driver, type="response") >= cutoff, 1,0)
table(pre, driver$test)
# Testing Accuracy
library(caret)
library(caTools)
y <- driver_scw$test
set.seed(seed)
train(test ~ bvtt + sdo + rp2 + ccds + bvtt:rp2 + bvtt:ccds +
    sdo:rp2 + sdo:ccds + rp2:ccds + sdo:rp2:ccds, data = driver_scw,
    method="glm", family = "binomial", tuneLength=10,
    trControl = trainControl(method='repeatedcv', index=CV_Folds5))
set.seed(seed)
train(test ~ bvtt + sdo + rp2 + ccds + bvtt:rp2 + bvtt:ccds +
    sdo:rp2 + sdo:ccds + rp2:ccds + sdo:rp2:ccds, data = driver_scw,
    method="glm", family = "binomial", tuneLength=10,
    trControl = trainControl(method='repeatedcv', index=CV_Folds10))
set.seed(seed)
train(test ~ bvtt + sdo + rp2 + ccds + bvtt:rp2 + bvtt:ccds +
    sdo:rp2 + sdo:ccds + rp2:ccds + sdo:rp2:ccds, data = driver_scw,
    method="glm", family = "binomial", tuneLength=10,
    trControl = trainControl(method='LOOCV', repeats = 5))
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_logi(driver_scw, 5, s)))
# Confusion matrix from LOO
pre_logi_loo <- CVError_logi_loo(driver_scw, 37, 840, 0.5)
table(as.factor(pre_logi_loo), driver_scw$test)

# Tree Model
# 36 obs, 12 variables
library(C50)
tree_36 <- C5.0(test~.,data=driver_36)
plot(tree_36)  # there is no scw
# 37 obs, 11 variables
library(C50)
tree <- C5.0(test~.-scw,data=driver)
summary(tree)
# CV error
c50Grid <- expand.grid(.trials = c(1:9),.model = "tree",.winnow = FALSE)
set.seed(seed)
train(driver_scw[,-1],y, method="C5.0", tuneGrid = c50Grid,
                  trControl = trainControl(method='repeatedcv', index=CV_Folds5))
set.seed(seed)
train(driver_scw[,-1],y, method="C5.0", tuneGrid = c50Grid,
                  trControl = trainControl(method='repeatedcv', index=CV_Folds10))
set.seed(seed)
```

```
train(driver_scw[,-1],y, method="C5.0",tuneGrid = c50Grid,
                  trControl = trainControl(method='LOOCV', repeats = 5))
# training AUC
library(pROC)
tree.pred <- predict.C5.0(tree, driver, type = "prob")[,2]
comtree <- cbind(as.numeric(driver[,1])-1,tree.pred)
colnames(comtree) <- c("test","response")
roc.tree <- roc(test~response, data= comtree, smooth=FALSE)
auc(roc.tree)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_tree(driver_scw, 5, s)))
# Confusion matrix from LOO
pre_tree_loo11 <- CVError_tree_loo(driver_scw, 37)
table(as.factor(pre_tree_loo11), driver_scw$test)

# 37obs, 7 variables
set.seed(seed)
tree_cor <- C5.0(test~., data = driver_cor)
summary(tree_cor)
plot(tree_cor)
# CV error
set.seed(seed)
train(driver_cor[,-1],y, method="C5.0",tuneGrid = c50Grid,
                  trControl = trainControl(method='repeatedcv', index=CV_Folds5))
set.seed(seed)
train(driver_cor[,-1],y, method="C5.0",tuneGrid = c50Grid,
                  trControl = trainControl(method='repeatedcv', index=CV_Folds10))
set.seed(seed)
train(driver_cor[,-1],y, method="C5.0",tuneGrid = c50Grid,
                  trControl = trainControl(method='LOOCV', repeats = 5))
# training AUC
library(pROC)
tree.pred <- predict.C5.0(tree_cor, driver, type = "prob")[,2]
comtree <- cbind(as.numeric(driver[,1])-1,tree.pred)
colnames(comtree) <- c("test","response")
roc.tree <- roc(test~response, data= comtree, smooth=FALSE)
auc(roc.tree)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_tree(driver_cor, 5, s)))
# Confusion matrix from LOO
pre_tree_loo7 <- CVError_tree_loo(driver_cor, 37)
table(as.factor(pre_tree_loo7), driver_cor$test)

# 37 obs, 4 variables
library(C50)
tree_vs <- C5.0(test~.,data=driver_lo)
summary(tree_vs)
plot(tree_vs)
# CV
set.seed(seed)
train(driver_lo[,-1],y, method="C5.0",tuneGrid = c50Grid,
                  trControl = trainControl(method='repeatedcv', index=CV_Folds5))
set.seed(seed)
```

```
train(driver_lo[,-1],y, method="C5.0",tuneGrid = c50Grid,
                 trControl = trainControl(method='repeatedcv', index=CV_Folds10))
set.seed(seed)
train(driver_lo[,-1],y, method="C5.0", tuneGrid = c50Grid,
                 trControl = trainControl(method='LOOCV', repeats = 5))
# training AUC
library(pROC)
tree.pred_vs <- predict.C5.0(tree_vs, driver_lo, type = "prob")[,2]
comtree_vs <- cbind(as.numeric(driver_lo[,1])-1,tree.pred_vs)
colnames(comtree_vs) <- c("test","response")
roc.tree_vs <- roc(test~response, data= comtree_vs, smooth=FALSE)
auc(roc.tree_vs)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_tree(driver_lo, 5, s)))
# Confusion matrix from LOO
pre_tree_loo4 <- CVError_tree_loo(driver_lo, 37)
table(as.factor(pre_tree_loo4), driver_lo$test)


# Tree Bagging
library(ipred)
# 12 variables 36 obs
err_36 <- double(90) # tune parameter
for(n in 11:100){
  set.seed(3)
  bag.fit <- bagging(test~., data = driver_36, nbagg = 2*n, coob=TRUE)
  err_36[n-10] <- bag.fit$err
}
plot(2*(11:100), err_36, type="b")
# take n = 24
set.seed(seed)
ip_36 <- bagging(test~., data = driver_36, coob =TRUE, nbagg=24)
# Misclassification rate
table(predict(ip_36, driver_36), driver_36$test)
# training AUC
ip.prob <- predict(ip_36, driver_36, type = "prob")
ip_36.rocr <- prediction(ip.prob[,2], driver_36$test)
ip_36.perf <- performance(ip_36.rocr, "tpr","fpr")
performance(ip_36.rocr, measure = "auc")@y.values
# CV error
set.seed(seed)
bag_m5 <- train(driver_36[,-1],driver_36[,1], method="treebag", tuneLength = 5,
                 trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-bag_m5$results[2]
set.seed(seed)
bag_m10 <- train(driver_36[,-1],driver_36[,1], method="treebag", tuneLength = 5,
                 trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-bag_m10$results[2]
set.seed(seed)
bag_mloo <- train(driver_36[,-1],driver_36[,1], method="treebag", tuneLength = 5,
                 trControl = trainControl(method='LOOCV', repeats = 5))
1-bag_mloo$results[2]
# testing AUC
```

```
mean(sapply(see_red, FUN = function(s) CVAUC_bag(driver_36, 5, 24, s)))
# Confusion matrix from LOO
pre_bag_loo12 <- CVError_bag_loo(driver_36, 36, 24)
table(as.factor(pre_bag_loo12), driver_36$test)

# 11 variables 37 obs
err_37 <- double(90)
for(n in 11:100){
  set.seed(seed)
  bag.fit <- bagging(test~., data = driver_scw, nbagg = 2*n, coob=TRUE)
  err_37[n-10] <- bag.fit$err
}
plot(2*(11:100), err_37, type="b")
# take n = 50
set.seed(seed)
ip_scw <- bagging(test~., data = driver_scw, coob =TRUE, nbagg = 50)
# Misclassification rate
table(predict(ip_scw, driver_scw), driver_scw$test)
# training AUC
ip_scw.prob <- predict(ip_scw, driver_scw, type = "prob")
ip_scw.rocr <- prediction(ip_scw.prob[,2], driver_scw$test)
ip_scw.perf <- performance(ip_scw.rocr, "tpr","fpr")
performance(ip_scw.rocr, measure = "auc")@y.values  # auc = 1
# CV error
set.seed(seed-2)
bag.scw_m5 <- train(driver_scw[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-bag.scw_m5$results[2]
set.seed(seed-2)
bag.scw_m10 <- train(driver_scw[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-bag.scw_m10$results[2]
set.seed(seed-2)
bag.scw_mloo <- train(driver_scw[,-1], y, method="treebag", tuneLength = 10,
                trControl = trainControl(method='LOOCV', repeats = 5))
1-bag.scw_mloo$results[2]
# testing AUC
mean(sapply(c(50, 40, 130, 100, 203), FUN = function(s) CVAUC_bag(driver_scw, 5, 50, s)))
# Confusion matrix from LOO
pre_bag_loo11 <- CVError_bag_loo(driver_scw, 37, 50)
table(as.factor(pre_bag_loo11), driver_scw$test)

# 7 vairalbes
err_cor <- double(90)
for(n in 11:100){
  set.seed(seed)
  bag.fit <- bagging(test~., data = driver_cor, nbagg = 2*n, coob=TRUE)
  err_cor[n-10] <- bag.fit$err
}
plot(2*(11:100), err_cor, type="b")
# take n = 46
set.seed(seed)
ip_cor <- bagging(test~., data = driver_cor, coob =TRUE, nbagg=46)
```

```
# Misclassification rate
table(predict(ip_cor, driver_cor), driver_cor$test)
# training AUC
ip_cor.prob <- predict(ip_cor, driver_cor, type = "prob")
ip_cor.rocr <- prediction(ip_cor.prob[,2], driver_cor$test)
ip_cor.perf <- performance(ip_cor.rocr, "tpr","fpr")
performance(ip_cor.rocr, measure = "auc")@y.values
# CV error
set.seed(seed)
bag.cor_m5 <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-bag.cor_m5$results[2]
set.seed(seed)
bag.cor_m10 <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-bag.cor_m10$results[2]
set.seed(seed)
bag.cor_mloo <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='LOOCV', repeats = 5))
1-bag.cor_mloo$results[2]
# testing AUC
mean(sapply(c(50, 220, 130, 180, 203), FUN = function(s) CVAUC_bag(driver_cor, 5, 46, s)))
# Confusion matrix from LOO
pre_bag_loo7 <- CVError_bag_loo(driver_cor, 37, 46)
table(as.factor(pre_bag_loo7), driver_cor$test)

# 4 variables
err_lo <- double(90)
for(n in 11:100){
  set.seed(seed)
  bag.fit <- bagging(test~., data = driver_lo, nbagg = 2*n, coob=TRUE)
  err_lo[n-10] <- bag.fit$err
}
plot(2*(11:100), err_cor, type="b")
# take n = 54
set.seed(seed)
ip_lo <- bagging(test~., data = driver_lo, coob =TRUE, nbagg=54)
# Misclassification rate
table(predict(ip_lo, driver_lo), driver_lo$test)
# training AUC
ip_lo.prob <- predict(ip_lo, driver_lo, type = "prob")
ip_lo.rocr <- prediction(ip_lo.prob[,2], driver_lo$test)
ip_lo.perf <- performance(ip_lo.rocr, "tpr","fpr")
performance(ip_lo.rocr, measure = "auc")@y.values  # auc = 1
plot(ip_lo.perf)
# CV error
set.seed(seed-1)
bag.lo_m5 <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-bag.lo_m5$results[2]
set.seed(seed-1)
bag.lo_m10 <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-bag.lo_m10$results[2]
```

```
set.seed(seed-1)
bag.lo_mloo <- train(driver_cor[,-1], y, method="treebag", tuneLength = 5,
                trControl = trainControl(method='LOOCV', repeats = 5))
1-bag.lo_mloo$results[2]
# testing AUC
mean(sapply(c(50, 220, 130, 180, 203), FUN = function(s) CVAUC_bag(driver_lo, 5, 54, s)))
# Confusion matrix from LOO
pre_bag_loo4 <- CVError_bag_loo(driver_lo, 37, 54)
table(as.factor(pre_bag_loo4), driver_lo$test)

#RF
library(e1071)
library(randomForest)
set.seed(seed)
rf_36 <- best.randomForest(test~., data = driver_36, mtry = c(1:12),
                           ntree = 100*seq(5,15,1), proximity = TRUE)
table(predict(rf_36, driver_36), driver_36$test)
# CV error
set.seed(seed)
rf_m5 <- train(driver_36[,-1],driver_36[,1], method="rf", tuneLength = 11,
               preProc = c("center", "scale"),
               trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-max(rf_m5$results[,2])
set.seed(seed)
rf_m10 <- train(driver_36[,-1],driver_36[,1], method="rf", tuneLength = 11,
               preProc = c("center", "scale"),
               trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-max(rf_m10$results[,2])
set.seed(seed)
rf_mloo <- train(driver_36[,-1],driver_36[,1], method="rf", tuneLength = 11,
               preProc = c("center", "scale"),
               trControl = trainControl(method='LOOCV', repeats = 5))
1-max(rf_mloo$results[,2])
# training AUC
prf_36 <- predict(rf_36, driver_36, type = "prob")[,2]
library(pROC)
roc.rf_36 <- roc(driver_36$test~prf_36, smooth = FALSE)
auc(roc.rf_36)
# testing AUC
mean(sapply(see_red, FUN = function(s) CVAUC_rf(driver_36, 5, s)))
# Confusion matrix from LOO
pre_rf_loo12 <- CVError_rf_loo(driver_36, 36)
table(as.factor(pre_rf_loo12), driver_36$test)

# 11 variables
set.seed(seed)
rf_37 <- best.randomForest(test~., data = driver_scw, mtry=c(1:11),
                           ntree=100*seq(5,15,1), proximity = TRUE)
table(predict(rf_37, driver_scw), driver_scw$test) # mis = 0
# CV error
set.seed(seed)
rf.scw_m5 <- train(driver_scw[,-1], y, method="rf", tuneLength = 11,
               preProc = c("center", "scale"),
```

```
                trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-max(rf.scw_m5$results[,2])
set.seed(seed)
rf.scw_m10 <- train(driver_scw[,-1], y, method="rf", tuneLength = 11,
                preProc = c("center", "scale"),
                trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-max(rf.scw_m10$results[,2])
set.seed(seed)
rf.scw_mloo <- train(driver_scw[,-1], y, method="rf", tuneLength = 11,
                preProc = c("center", "scale"),
                trControl = trainControl(method='LOOCV', repeats = 5))
1-max(rf.scw_mloo$results[,2])
# training AUC
prf_37 <- predict(rf_37, driver_scw, type = "prob")[,2]
library(pROC)
roc.rf_37 <- roc(driver_scw$test~prf_37, smooth = FALSE)
auc(roc.rf_37)
# testing AUC
mean(sapply(see_red, FUN = function(s) CVAUC_rf(driver_scw, 5, s)))
# Confusion matrix from LOO
pre_rf_loo11 <- CVError_rf_loo(driver_scw, 37)
table(as.factor(pre_rf_loo11), driver_scw$test)

# 7 variables
set.seed(seed)
rf_cor <- best.randomForest(test~., data = driver_cor, mtry = c(1:7),
                            ntree = 100*seq(5,15,1), proximity = TRUE)
table(predict(rf_cor, driver_cor), driver_cor$test)
importance(rf_cor, type = 1) # cwt jlo cutt butt sdo rp2 ccds
# CV error
set.seed(seed)
rf.cor_m5 <- train(driver_cor[,-1], y, method="rf", tuneLength = 11,
                preProc = c("center", "scale"),
                trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-max(rf.cor_m5$results[,2])
set.seed(seed)
rf.cor_m10 <- train(driver_cor[,-1], y, method="rf", tuneLength = 11,
                preProc = c("center", "scale"),
                trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-max(rf.cor_m10$results[,2])
set.seed(seed)
rf.cor_mloo <- train(driver_cor[,-1], y, method="rf", tuneLength = 11,
                preProc = c("center", "scale"),
                trControl = trainControl(method='LOOCV', repeats = 5))
1-max(rf.cor_mloo$results[,2])
# training AUC
prf_cor <- predict(rf_cor, driver_cor, type = "prob")[,2]
library(pROC)
roc.rf_cor <- roc(driver_cor$test~prf_cor, smooth = FALSE)
auc(roc.rf_cor)
# testing AUC
mean(sapply(see_red, FUN = function(s) CVAUC_rf(driver_cor, 5, s)))
# Confusion matrix from LOO
```

```
pre_rf_loo7 <- CVError_rf_loo(driver_cor, 37)
table(as.factor(pre_rf_loo7), driver_cor$test)

# 4 variables
set.seed(seed)
rf_lo <- best.randomForest(test~., data = driver_lo, mtry = c(1:4),
                           ntree = 100*seq(5,15,1), proximity = TRUE)
table(predict(rf_lo, driver_lo), driver_lo$test)
# CV error
set.seed(seed)
rf.lo_m5 <- train(driver_lo[,-1], y, method="rf", tuneLength = 11,
                  preProc = c("center", "scale"),
                  trControl = trainControl(method='repeatedcv', index=CV_Folds5))
1-max(rf.lo_m5$results[,2])
set.seed(seed)
rf.lo_m10 <- train(driver_lo[,-1], y, method="rf", tuneLength = 11,
                  preProc = c("center", "scale"),
                  trControl = trainControl(method='repeatedcv', index=CV_Folds10))
1-max(rf.lo_m10$results[,2])
set.seed(seed)
rf.lo_mloo <- train(driver_lo[,-1], y, method="rf", tuneLength = 11,
                  preProc = c("center", "scale"),
                  trControl = trainControl(method='LOOCV', repeats = 5))
1-max(rf.lo_mloo$results[,2])
# training AUC
prf_lo <- predict(rf_lo, driver_lo, type = "prob")[,2]
library(pROC)
roc.rf_lo <- roc(driver_lo$test~prf_lo, smooth = FALSE)
auc(roc.rf_lo)
# testing AUC
mean(sapply(see_red, FUN = function(s) CVAUC_rf(driver_lo, 5, s)))
# Confusion matrix from LOO
pre_rf_loo4 <- CVError_rf_loo(driver_lo, 37)
table(as.factor(pre_rf_loo4), driver_lo$test)

#SVM
library(e1071)
# 12 variables 36 obs
set.seed(seed)
svm_36 <- best.svm(test~., data = driver_36, scale=TRUE, kernel="radial",
          gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)),probability = TRUE)
table(predict(svm_36, driver_36), driver_36$test)
# CV error
set.seed(seed)
svm_m5 <- train(driver_36[,-1], driver_36[,1], method="svmRadial",
                tuneLength=10, preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds5))
1 - max(svm_m5$results[,3])
set.seed(seed)
svm_m10 <- train(driver_36[,-1], driver_36[,1], method="svmRadial",
                 tuneLength=10, preProc = c("center", "scale"),
                 trControl=trainControl(method='repeatedcv', index=CV_Folds10))
1 - max(svm_m10$results[,3])
```

```
set.seed(seed)
svm_mloo <- train(driver_36[,-1], driver_36[,1], method="svmRadial",
                  tuneLength=10, preProc = c("center", "scale"),
                  trControl=trainControl(method='LOOCV', repeats = 5))
1 - max(svm_mloo$results[,3])
# training AUC
psvm_36 <- attr(predict(svm_36, driver_36, probability = TRUE,
                        decision.values = TRUE), "decision.values")
library(pROC)
roc.svm_36 <- roc(yTe_36~as.vector(psvm_36), smooth = FALSE)
auc(roc.svm_36)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_svm(driver_36, K=5, s)))
# Confusion matrix from LOO
pre_svm_loo12 <- CVError_svm_loo(driver_36, 36)
table(as.factor(pre_svm_loo12), driver_36$test)

# 11 predictors
svm_37 <- best.svm(test~., data = driver_scw, scale=TRUE, kernel="radial",
          gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)), probability = TRUE)
table(predict(svm_37, driver_scw), driver_scw$test)
# CV errors
set.seed(seed)
svm.scw_m5 <- train(driver_scw[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds5))
1 - max(svm.scw_m5$results[,3])
set.seed(seed)
svm.scw_m10 <- train(driver_scw[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds10))
1 - max(svm.scw_m10$results[,3])
set.seed(seed)
svm.scw_mloo <- train(driver_scw[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='LOOCV', repeats = 5))
1 - max(svm.scw_mloo$results[,3])
# training AUC
psvm_37 <- attr(predict(svm_37, driver_scw, probability = TRUE,
                        decision.values = TRUE), "decision.values")
library(pROC)
roc.svm_37 <- roc(yTe~as.vector(psvm_37), smooth = FALSE)
auc(roc.svm_37)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_svm(driver_scw, K=5, s)))
# Confusion matrix from LOO
pre_svm_loo11 <- CVError_svm_loo(driver_scw, 37)
table(as.factor(pre_svm_loo11), driver_scw$test)

# 7 variables
svm_cor <- best.svm(test~., data = driver_cor, scale=TRUE, kernel="radial",
           gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)), probability = TRUE)
table(predict(svm_cor, driver_cor), driver_cor$test)
```

```
# CV error
set.seed(seed)
svm.cor_m5 <- train(driver_cor[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds5))
1 - max(svm.cor_m5$results[,3])
set.seed(seed)
svm.cor_m10 <- train(driver_cor[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds10))
1 - max(svm.cor_m10$results[,3])
set.seed(seed)
svm.cor_mloo <- train(driver_cor[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='LOOCV', repeats = 5))
1 - max(svm.cor_mloo$results[,3])
# Training AUC
psvm_cor <- attr(predict(svm_cor, driver_cor, probability = TRUE,
                         decision.values = TRUE), "decision.values")
library(pROC)
roc.svm_cor <- roc(yTe_cor~as.vector(psvm_cor), smooth = FALSE)
auc(roc.svm_cor)
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_svm(driver_cor, K=5, s)))
# Confusion matrix from LOO
pre_svm_loo7 <- CVError_svm_loo(driver_cor, 37)
table(as.factor(pre_svm_loo7), driver_cor$test)

# 4 variables
svm_lo <- best.svm(test~., data = driver_lo, scale=TRUE, kernel="radial",
          gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)), probability = TRUE)
table(predict(svm_lo, driver_lo), driver_lo$test)
# CV error
set.seed(seed)
svm.lo_m5 <- train(driver_lo[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds5))
1 - max(svm.lo_m5$results[,3])
set.seed(seed)
svm.lo_m10 <- train(driver_lo[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='repeatedcv', index=CV_Folds10))
1 - max(svm.lo_m10$results[,3])
set.seed(seed)
svm.lo_mloo <- train(driver_lo[,-1], y, method="svmRadial", tuneLength=10,
                preProc = c("center", "scale"),
                trControl=trainControl(method='LOOCV', repeats = 5))
1 - max(svm.lo_mloo$results[,3])
# training AUC
psvm_lo <- attr(predict(svm_lo, driver_lo, probability = TRUE,
                        decision.values = TRUE), "decision.values")
library(pROC)
roc.svm_lo <- roc(yTe_lo~as.vector(psvm_lo), smooth = FALSE)
auc(roc.svm_lo)
```

```r
# testing AUC
mean(sapply(see, FUN = function(s) CVAUC_svm(driver_lo, K=5, s)))
# Confusion matrix from LOO
pre_svm_loo4 <- CVError_svm_loo(driver_lo, 37)
table(as.factor(pre_svm_loo4), driver_lo$test)

# Change of Testing AUC (weighted svm)
Cauc <- function(x) {
 mean(sapply(see_red, FUN = function(s) CVAUC_svm(driver_scw[,-x], K=5, s)))
}
   # Round 1
auc_all <- mean(sapply(see_red, FUN = function(s) CVAUC_svm(driver_scw, K=5, s)))
#0.732
auc_result1 <- numeric(11)
auc_result1 <- sapply(2:12, FUN = function(x) Cauc(x))
# calculate which variable to remove(have greatest increase in auc)
diff1 <- auc_result1 - auc_all
remove1 <- which.max(diff1) # 5, index=6
index <- order(diff1, decreasing = TRUE)
names(driver_scw)[-1][index]
#"bvtd" "cccs" "bvtt" "cvtt" "rp3"  "cwt"  "rp2"  "cvtd" "sdo"  "jlo"  "ccds"
   # Round 2
auc_all2 <- auc_result1[remove1] # 0.7453333
auc_result2 <- sapply(c(2:6,8:12), FUN = function(x) Cauc(c(remove1+1, x)))
diff2 <- auc_result2 - auc_all2
remove2 <- which.max(diff2)
index2 <- order(diff2, decreasing = TRUE)
names(driver_scw)[-c(1,7)][index2]
#  "bvtt" "cwt"  "cvtd" "cvtt" "rp2"  "sdo"  "rp3"  "jlo"  "cccs" "ccds"
   # Round 3
auc_all3 <- auc_result2[remove2] # 0.7380000
auc_result3 <- sapply(c(2:5,8:12), FUN = function(x) Cauc(c(6,7, x)))
diff3 <- auc_result3 - auc_all3
remove3 <- which.max(diff3) #
index3 <- order(diff3, decreasing = TRUE)
names(driver_scw)[-c(1,6,7)][index3]
# "rp2"  "sdo"  "cccs" "ccds" "cvtd" "jlo"  "rp3"  "cvtt" "cwt"
   # Round 4
auc_all4 <- auc_result3[remove3] # 0.7986667
auc_result4 <- sapply(c(2:5,8:9,11,12), FUN = function(x) Cauc(c(6,7,10, x)))
diff4 <- auc_result4 - auc_all4
remove4 <- which.max(diff4) #
index4 <- order(diff4, decreasing = TRUE)
names(driver_scw)[-c(1,6,7,10)][index4]
# "sdo"  "rp3"  "ccds" "cccs" "cwt"  "cvtd" "jlo"  "cvtt"
   # Round 5
auc_all5 <- auc_result4[remove4] # 0.8046667
auc_result5 <- sapply(c(2:5,9,11,12), FUN = function(x) Cauc(c(6,7,8,10, x)))
diff5 <- auc_result5 - auc_all5
remove5 <- which.max(diff5) #
index5 <- order(diff5, decreasing = TRUE)
names(driver_scw)[-c(1,6,7,8,10)][index5]
# "cccs" "rp3"  "ccds" "cwt"  "cvtd" "jlo"  "cvtt"
  # Round 6
```

```
auc_all6 <- auc_result5[remove5] # 0.8306667
auc_result6 <- sapply(c(2:5,9,12), FUN = function(x) Cauc(c(6,7,8,10,11, x)))
diff6 <- auc_result6 - auc_all6
remove6 <- which.max(diff6) #
index6 <- order(diff6, decreasing = TRUE)
names(driver_scw)[-c(1,6,7,8,10,11)][index6]
#"rp3"  "ccds" "cvtd" "cwt"  "jlo"  "cvtt"
  # Round 7
auc_all7 <- auc_result6[remove6] # 0.8513333
auc_result7 <- sapply(c(2:5,12), FUN = function(x) Cauc(c(6:11, x)))
diff7 <- auc_result7 - auc_all7
remove7 <- which.max(diff7) #
index7 <- order(diff7, decreasing = TRUE)
names(driver_scw)[-c(1,6:11)][index7]
# "cvtd" "cwt"  "jlo"  "ccds" "cvtt"
  # Round 8
auc_all8 <- auc_result7[remove7] # 0.8513333
auc_result8 <- sapply(c(2:4,12), FUN = function(x) Cauc(c(5:11, x))) # end

# Highest testingAUC with bvtt
aucmean <- numeric(13)
for(i in c(2:5,7:13)) {
  va <- c(1,6,i)
  aucmean[i] <- mean(sapply(see_red,
                            FUN = function(s) CVAUC_svm_w(driver_scw[va], K=5, s)))
}
which.max(aucmean) # choose cwt
aucmean2 <- numeric(13)
for(i in c(9:10)) {
  va <- c(1,2,6,i)
  aucmean2[i] <- mean(sapply(see_red,
                             FUN = function(s) CVAUC_svm_w(driver_scw[va], K=5, s)))
}
which.max(aucmean2)


# Class weighted model performance metrics
va <- c(1,2,6,9) # indices of variables selected, different methods try different values
set.seed(seed)
svm_w_vi2 <- best.svm(test~., data = driver_scw[,va], scale=TRUE, kernel="radial",
             gamma = 2^(seq(-15,15,3)), cost = 2^(seq(-5,15,2)),probability = TRUE,
             class.weights= c("0" = 1, "1" = 3))
yHe <- driver_scw$test
table(predict(svm_w_vi2, driver_scw[,va]), yHe)
# training AUC
psvm_w_vi2 <- attr(predict(svm_w_vi2, driver_scw[,va], probability = TRUE,
                           decision.values = TRUE), "decision.values")
library(pROC)
roc.svm_w_vi2 <- roc(yHe~as.vector(psvm_w_vi2), smooth = FALSE)
auc(roc.svm_w_vi2)
# CV error
CVEsvm_w_vi2 <- apply(sapply(see, FUN = function(s)
                      sapply(k, FUN = function(x)
```

```
                        CVError_svm_w(driver_scw[va], x, s))),1,mean)
CVEsvm_w_vi2
# testing AUC (only try K =5)
mean(sapply(see_red, FUN = function(s)
            CVAUC_svm_w(driver_scw[va], K=5, s)))
# Confusion matrix from LOO for the best model
pre_wsvm_loo <- CVError_wsvm_loo(driver_scw[,va], 37,seed+320)
table(as.factor(pre_wsvm_loo), driver_scw$test)
```