Assesment

2022-12-20

Environement settings

```
library(VSURF)
library(rpart) # for CART
library(rpart.plot)
library(randomForest)
library(MASS) # for variable selection
library(glmnet) # for computing linear model with Lasso penalization
library(tidyverse) # coding
library(broom) # for nice display
library(caret) # To compute metrics for binary classification
```

Data preparation

Loading the data

```
data('toys')
```

The object `toys` is composed by `toys$x` and `toys$y`.In `toys$x`, the first 6 are influential variables (Genuer and al.) and the rest are noise variables. `toys$y` is the response variable. We have 100 observations, 200 features (`toys$x`) and 1 response with a binary output {-1, 1} (`toys$y`). Let's put all of this into one data frame.

Merge to create a data frame

```
toys <- cbind(toys$x, toys$y)
```

Split the data

80 random observation for training 20 for testing

```
set.seed(122) # set seed to reproduce same results
train <- sample(1:nrow(toys), 80)
test <- setdiff(1:nrow(toys), train)
```

Define X_train, y_train, X_test, y_test.

X_train and X_test are the predictors and made by the first 200 columns of the data frame. y_train and y_test are the respective responses and made by the last column of the data frame.

```
X_train <- toys[train, 1:200]
y_train <- toys[train, 201]
X_test <- toys[test, 1:200]
y_test <- toys[test, 201]
```

Fit a linear model and variable selection

Variable selection with elastic net regularization

Compute a cross validation to find the best hyper parameter lambda in a Lasso regression: $\alpha = 1$. $family =' binomial'$ stands for binary output.

```
cv.glmnet(as.matrix(X_train), as.matrix(y_train), family = "binomial", alpha = 1)%>%
  print()
```

```
##
## Call:  cv.glmnet(x = as.matrix(X_train), y = as.matrix(y_train), family = "binomial",      alpha = 1)
##
## Measure: Binomial Deviance
##
##       Lambda Index Measure      SE Nonzero
## min 0.003952   100 0.05265 0.01327      12
## 1se 0.006593    89 0.06463 0.01395      11
```

The cross validation suggests if we look at the first `se` we should take $\lambda = 0.006593$.

```
#Fit the model
model_glm <- glmnet(X_train, y_train, alpha = 1, family = 'binomial', lambda = 0.006593)
```

```
model_glm$beta@i
```

```
##  [1]   0   1   2   3   4   5  31  36  93 158 178
```

```
model_glm$beta@p
```

```
## [1]  0 11
```

Out of 200 hundred variables, only 11 remains in the final model after the Lasso regularization:

V1, V2, V3, V4, V5, V6, V32, V37, V94, V159, V179 (shift of one compare to the output of `model_glm$beta@i`).

We can notice the first 6 variables have been captured which is relevant from the data description. However it has aslo captured 5 other features which should be noise. It could lead to an overfit.

Model evaluation

Check the distribution of the output

As we have sample randomly, we may have an imbalanced training data set with a prevalence of one of levels of y_train ,{-1} or {1}. We must consider this point to choose the appropriate metric.

```
cat('Number of occurences of value 1 in the whole data set toys : ', table(toys[,201])['1'], '\n',
'Number of occurences of value -1 in the whole data set toys : ', table(toys[,201])['-1'], '\n',
   '-------------------------------------------------------------------------------------------','\n',
'Number of occurences of value 1 in the training data set : ', table(y_train)['1'], '\n',
'Number of occurences of value -1 in the training data set : ', table(y_train)['-1'], '\n')
```

```
## Number of occurences of value 1 in the whole data set toys :  56
##  Number of occurences of value -1 in the whole data set toys :  44
##  -------------------------------------------------------------------------------------------
##  Number of occurences of value 1 in the training data set :  44
##  Number of occurences of value -1 in the training data set :  36
```

In both the initial data set and the training one we have a light imbalanced data set. As we are dealing with a binary output, it could be interesting to use the F1 score, which will captured both the recall and the precision. Moreover, we do not have any context which implies any interpretation on the output {-1} or {1}, so we do not really care which of the recall or the precision is the most important. In that sense, F1 score seems to be the most appropriate metric.

```
# Apply prediction with the fitted model
pred_lasso <- predict(model_glm, as.matrix(X_test), type = "class")
# Change the output into a factor with two levels -1 and 1, same as `y_test`
pred_lasso <- factor(pred_lasso, levels = c(-1, 1))
# Print out metrics
confusionMatrix(pred_lasso, y_test, mode = "everything", positive="1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1  8  0
##         1   0 12
##
##                Accuracy : 1
##                  95% CI : (0.8316, 1)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 3.656e-05
##
##                   Kappa : 1
##
##  Mcnemar's Test P-Value : NA
##
##             Sensitivity : 1.0
##             Specificity : 1.0
##          Pos Pred Value : 1.0
##          Neg Pred Value : 1.0
##               Precision : 1.0
```

```
##                  Recall : 1.0
##                      F1 : 1.0
##              Prevalence : 0.6
##          Detection Rate : 0.6
##    Detection Prevalence : 0.6
##       Balanced Accuracy : 1.0
##
##        'Positive' Class : 1
##
```

We have only TRUE predictions, no errors at all. This is clearly an excellent model with a $F1_{glm} = 1$.

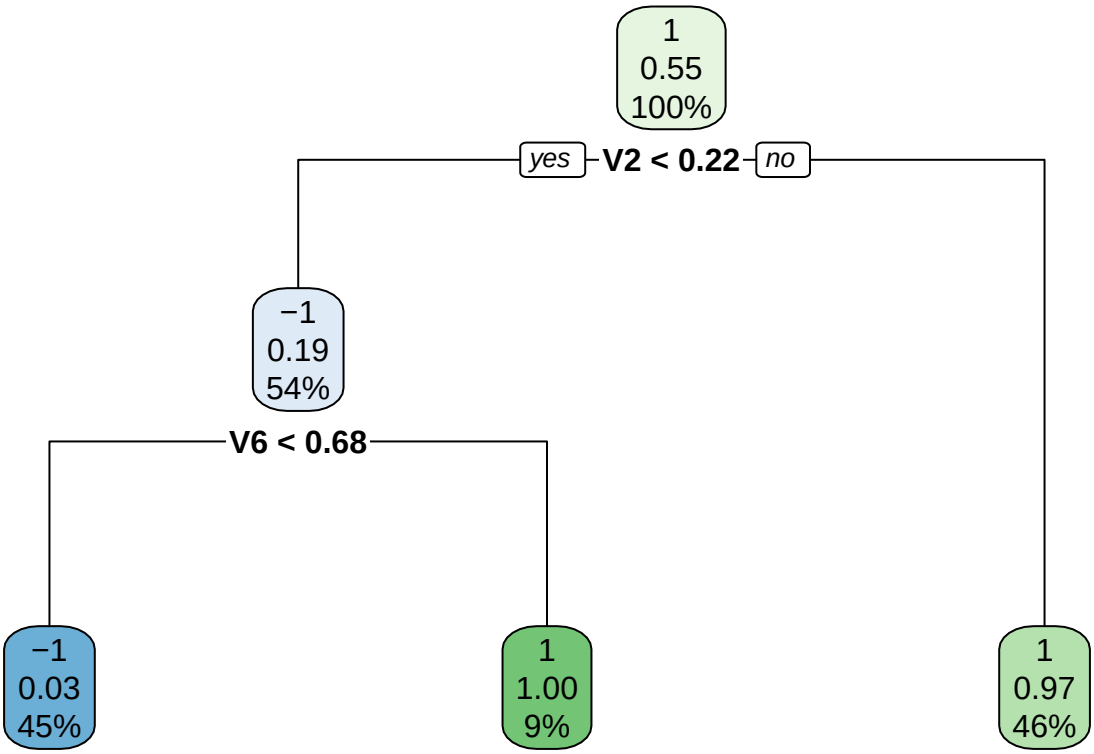
But we have to remember the model have captured noise features, and if we use another sample, it might over fit, and get worst results.

Fit a CART model

```
#Fit a CART model
model_cart <- rpart(y_train ~ ., data = X_train, method = "class", cp=0.01)
```

Features importance and tree plot

```
rpart.plot(model_cart)
```

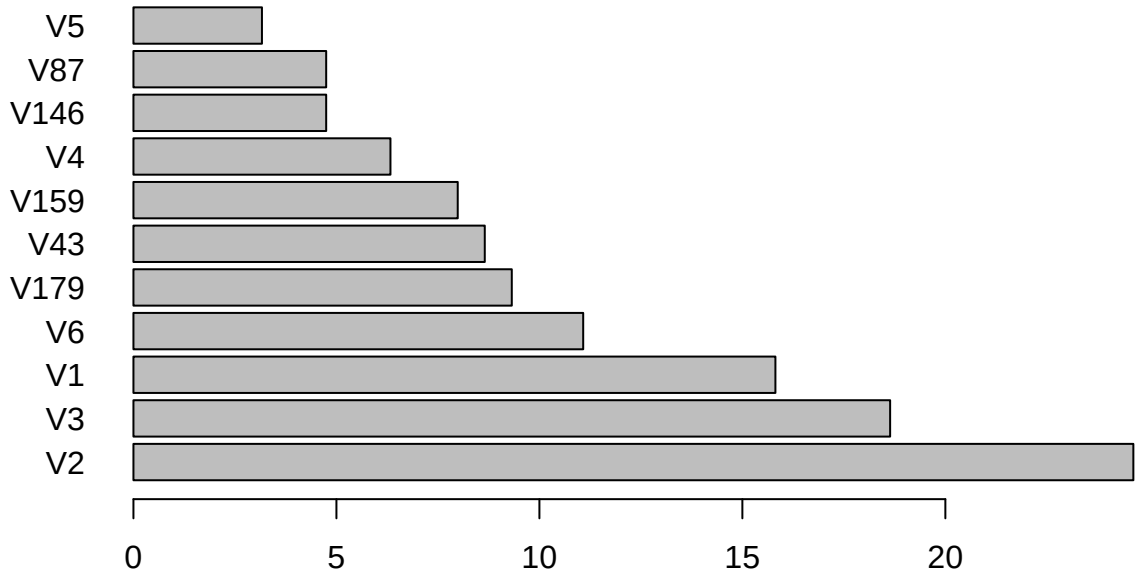


This time only two nodes have been enough to process and sort the output into three leaves.

For this example, conducting a variable selection with a CART algorithm, seems to be very quick.

We can also have a look onto the variable importance plot.

```
barplot(model_cart$variable.importance,horiz = T,las=1)
```



The first 4 more important variables are from the first 6 of the initial model, which is relevant.

Model evaluation

```
#compute the prediction
pred_cart <- rpart.predict(object = model_cart, newdata = X_test, type = 'class')

# Print out metrics
confusionMatrix(pred_cart, y_test, mode = "everything", positive="1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1  6  0
##         1   2 12
##
##                Accuracy : 0.9
##                  95% CI : (0.683, 0.9877)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.003611
##
##                   Kappa : 0.7826
##
##  Mcnemar's Test P-Value : 0.479500
```

```
##
##             Sensitivity : 1.0000
##             Specificity : 0.7500
##          Pos Pred Value : 0.8571
##          Neg Pred Value : 1.0000
##               Precision : 0.8571
##                  Recall : 1.0000
##                      F1 : 0.9231
##              Prevalence : 0.6000
##          Detection Rate : 0.6000
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.8750
##
##        'Positive' Class : 1
##
```

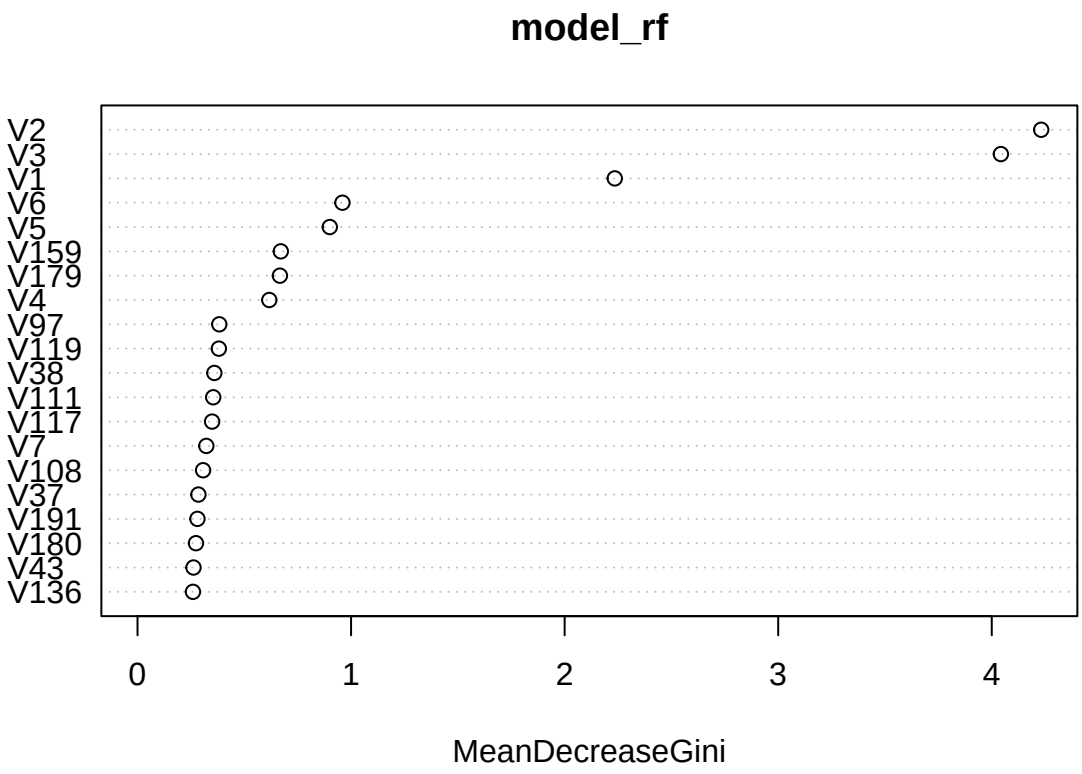
On twenty predictions only two are falses (False positive as the positive class is 1). We have a $F1_{CART} = 0.9231$. It is a bit less than the one from glm, but less variables were needed, the risk to over fit on new data is lower.

Fit a random Forest

To be consistent with the `VSURF` model, we will explicitly ask 2000 trees, which is the default value in `VSURF`.

```
model_rf <- randomForest(y_train ~ ., data = X_train, xtest= X_test, ytest = y_test, ntree = 2000)
```

```
varImpPlot(model_rf,n.var = 20 )
```



It is interesting to notice the top five are all in the six first variables of the initial data set. This model, while it is heavy, captured quiet well the true feature importance.

Model Evaluation

```
# Predictions
pred_rf <- model_rf$test$predicted

# Print out metrics
confusionMatrix(pred_rf, y_test, mode = "everything", positive="1")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction -1  1
##         -1  6  0
##         1   2 12
##
##                Accuracy : 0.9
##                  95% CI : (0.683, 0.9877)
##     No Information Rate : 0.6
##     P-Value [Acc > NIR] : 0.003611
##
##                   Kappa : 0.7826
```

```
##
##  Mcnemar's Test P-Value : 0.479500
##
##             Sensitivity : 1.0000
##             Specificity : 0.7500
##          Pos Pred Value : 0.8571
##          Neg Pred Value : 1.0000
##               Precision : 0.8571
##                  Recall : 1.0000
##                      F1 : 0.9231
##              Prevalence : 0.6000
##          Detection Rate : 0.6000
##    Detection Prevalence : 0.7000
##       Balanced Accuracy : 0.8750
##
##        'Positive' Class : 1
##
```

We have the same errors with the random forest and the CART procedure. We have $F1_{RF} = 0.9231$.

Let's now conduct a VSURF procedure, which is processing a variable selection on the RF.

Fit with VSURF

VSURF package process a variable selection into three steps based on `CART` and `randomForest` algorithm.

```
model_vsurf <- VSURF(x=X_train, y= y_train )
```

```
## Thresholding step
## Estimated computational time (on one core): 2.8 sec.
##   |                                                                              |
## Interpretation step (on 18 variables)
## Estimated computational time (on one core): between 0 sec. and  0 sec.
##   |                                                                              |
## Prediction step (on 5 variables)
## Maximum estimated computational time (on one core): 1 sec.
##   |                                                                              |
```

Step 1:Preliminary elimination:

It will rank the variables by their importance in a descending order. Then as the standard deviation for useless variables is lower than for important ones, a threshold is compute and only the most important ones remains (with a higher standard deviation VI)

```
model_vsurf$varselect.thres
```

```
##  [1]   3   2   6   1   5   4 179 119 159 180  37  97 191 136 157 163 146  94
```

Step 2: Variable selection

It will compute several random Forest, starting from the remaining variables of step 1, and it will keep the model with the smallest `Out Of Bag` error.

```
model_vsurf$varselect.interp
```

```
## [1] 3 2 6 1 5
```

Only V3, V2, V6, V1 and V5 remains for the interpretation. It makes sense, as only the first 6 are influential variables. All the others have been built to be noise variables.

We can already make some prediction from this model, but we may have some redundancy.

Step 3:

Thus last step is to avoid hypothetical redundancy between the remaining variables from the second step. The goal is to find a smaller subset of variables but still good enoug for predictions.

```
model_vsurf$varselect.pred
```

```
## [1] 3 6 1 5
```

Only 4 variables remains in this last step.

Metrics

It can be now interesting to evaluate both models, the one out of the interpretation step with 5 variables and the other one after the prediction step with only 4 variables.

```
#Predictions
pred_vsurf <- predict(model_vsurf, X_test)

# Print out metrics
m_int<- confusionMatrix(pred_vsurf$interp, y_test, mode = "everything", positive="1")
m_pred<- confusionMatrix(pred_vsurf$pred, y_test, mode = "everything", positive="1")

m_int$table
```

```
##            Reference
## Prediction -1  1
##         -1  7  0
##         1   1 12
```

```
m_pred$table
```

```
##            Reference
## Prediction -1  1
##         -1  7  0
##         1   1 12
```

```r
# Print F1 scores
cat('F1 score after the interpreation step: F1=', m_int$byClass['F1'], '\n',
  'F1 score after the prediction step: F1=', m_pred$byClass['F1'], '\n')
```

```
## F1 score after the interpreation step: F1= 0.96
##  F1 score after the prediction step: F1= 0.96
```

We can see from this result, that both model have the same results whereas the model with predictions require only 4 variables instead of 5 from the interpretation model. We have $F1_{VSURF} = 0.96$

Cross validation

Let's repeat this `VSURF` procedure 50 times, each time with a random sampling.

```r
  cv <- function(){
    '
    Train and test a VSURF model and get the results:
    variables selected and F1 score alongside into a dataframe
    '

    #Split the data 80/20
    train <- sample(1:nrow(toys), 80)
    test <- setdiff(1:nrow(toys), train)
    X_train <- toys[train, 1:200]
    y_train <- toys[train, 201]
    X_test <- toys[test, 1:200]
    y_test <- toys[test, 201]
    # Fit the model
    model_vsurf <- VSURF(x=X_train, y= y_train, mtry = 100 )
    # Predict
    pred_vsurf <- predict(model_vsurf, X_test)
    # Save the model
    model_vsurf
    # Save the variables selected
    variables <- model_vsurf$varselect.pred
    # Put the variables selected into a list of one element into a string
    variables_list <- paste(variables, collapse = ",")

    # Get the F1 score
    m_pred<- confusionMatrix(pred_vsurf$pred, y_test, mode = "everything", positive="1")
    F1 <- m_pred$byClass['F1']

    # Build a dataframe with the variables selected and the F1 score
    df <- data.frame(variables_selected = variables_list, F1 = F1)

    return(df)
}
```

Compute the dataframe

```
df <- data.frame()
for (i in 1:50){
  df <- rbind(df, cv())
}
```

Comments

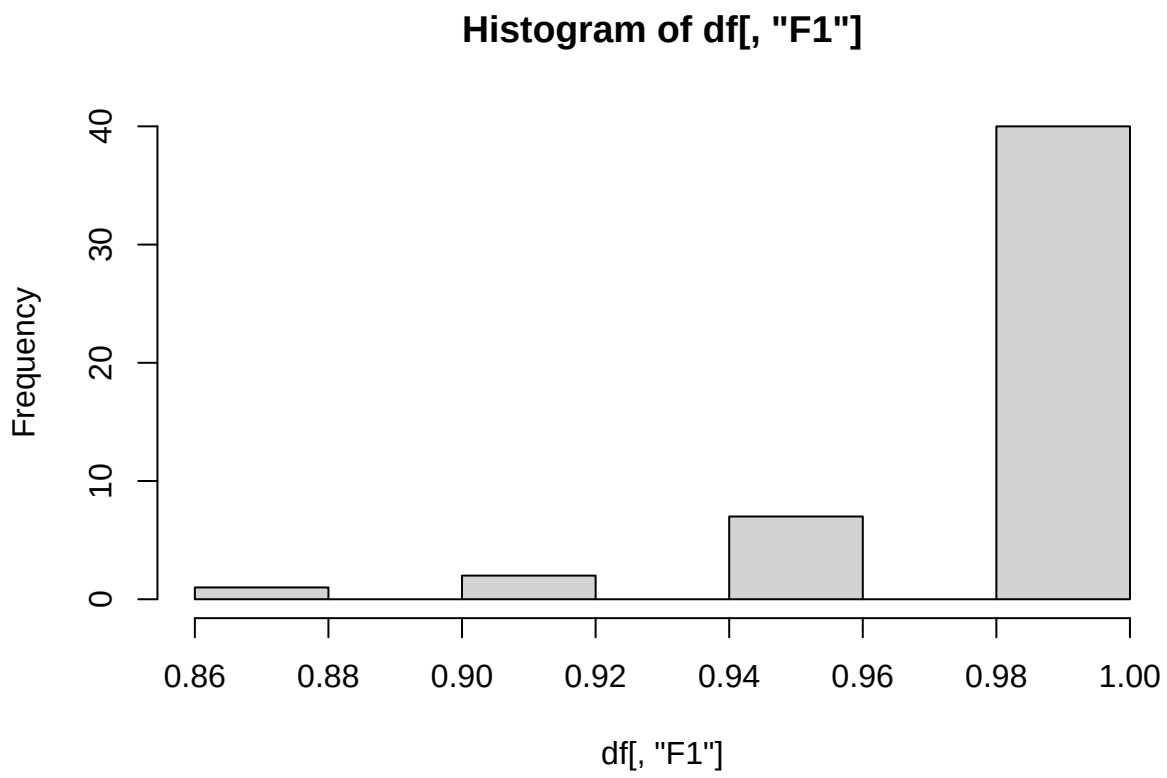
```
head(df)
```

```
##     variables_selected   F1
## F1               3,6,5 1.00
## F11              3,6,5 1.00
## F12              3,6,5 1.00
## F13                3,6 1.00
## F14            3,6,1,5 0.96
## F15              3,6,5 1.00
```

Each line match for one model (50). The first column **variables_selected** stands for the variables selected after the final step prediction, and the second column **F1** is the F1 score of this model on the test set.

We can have a look onto the the different value of F1 score to check how well the model predicts the results.

```
hist(df[, "F1"])
```



We can clearly see, most of the time it is able to catch a $F1_{VSURF} = 1$.

Finally let's have a look into a table of frequency to understand which models end up with a $F1_{VSURF} = 1$.

```
table( df[,1], df['F1'])
```

```
##
##                0.875 0.909090909090909 0.947368421052632 0.952380952380952
##   2,3,1,6,5        0                 0                 0                 0
##   2,3,5,21         0                 0                 0                 0
##   2,3,5,6,1        0                 0                 0                 0
##   2,3,6            0                 0                 0                 0
##   2,3,6,5,1        0                 0                 0                 0
##   2,5              1                 0                 0                 0
##   2,6              0                 0                 0                 0
##   2,6,1            0                 1                 0                 0
##   2,6,5            0                 0                 0                 0
##   2,6,5,1,143      0                 0                 0                 0
##   3,1,6            0                 0                 0                 0
##   3,1,6,5          0                 0                 0                 0
##   3,2,6            0                 0                 0                 0
##   3,2,6,5          0                 0                 0                 0
##   3,2,6,5,4        0                 0                 1                 0
##   3,5,1,6          0                 1                 0                 0
##   3,5,6            0                 0                 0                 0
##   3,6              0                 0                 0                 0
##   3,6,1            0                 0                 0                 0
##   3,6,1,5          0                 0                 0                 1
##   3,6,5            0                 0                 0                 0
##   3,6,5,1          0                 0                 1                 0
##
##                0.956521739130435 0.96  1
##   2,3,1,6,5                    0    0  1
##   2,3,5,21                     0    0  1
##   2,3,5,6,1                    0    0  1
##   2,3,6                        0    0  1
##   2,3,6,5,1                    0    0  1
##   2,5                          0    0  0
##   2,6                          1    0  0
##   2,6,1                        0    0  0
##   2,6,5                        0    0  1
##   2,6,5,1,143                  1    0  0
##   3,1,6                        0    0  1
##   3,1,6,5                      0    0  1
##   3,2,6                        0    0  1
##   3,2,6,5                      0    0  2
##   3,2,6,5,4                    0    0  0
##   3,5,1,6                      0    0  0
##   3,5,6                        0    0  3
##   3,6                          0    0  3
##   3,6,1                        0    1  2
##   3,6,1,5                      0    1  1
##   3,6,5                        0    0 19
##   3,6,5,1                      0    0  1
```

Finally, many models even with only 3 variables were able to get $F1_{VSURF} = 1$. Each time, all the variables selected are among the first six $V_i, i = 1, ...6$.

Conclusion

The linear model was good to predict the results, $F1_{glm} = 1$, but we have some reserve regarding future overfitting has it has captured noise variables.
The CART algorithm has done a really good job to process a variable selection (only 2 splits needed), and the result is still really good: $F1_{CART} = 0.9231$.
The randomForest, in this case has no interest compare to CART has it has the same result, $F1_{RF} = 0.9231$, but used all variables to process all the trees.
Finally, VSURF is the lightest model, some models out of the cross validation required only 3 variables, but still were able to get $F1_{VSURF} = 1$.
In that sense, if we have to put a model into production, we might consider a VSURF model.