Report: Study of the Visa Premier Dataset

Mai-Anh Dang

January 30, 2018

1. Introduction

This report is a part of Big Data Course at Toulouse School of Economics. The interested problem is the prediction of customer behavior for a bank product - the *visa premier card*, to optimize the advertisement. The original data comes from a real dataset of *Caisse d'Epargne*. We would predict with the model of logistic regression, Random Forest and Support Vector Machine.

2. Exploratory Approach

2.1. Loading the dataset

To load data, we use these programs:

Program `lecture_visprem.R`

- The input file is `vispremR.txt`, the ouput is `visprem.txt`
- **Key Task**: Factorisation Qualitative variable
- Other data preprocessing tasks: i) Aggregate the days of debit; ii) Drop the observation by inappropiate ages, bancaires (`G29G30S-G47G48S`), term account or certificate (`NBCATS`, `NBBECS`)

Program `transf_visprem.R`

- The input file is `visprem.txt`, the ouput is `vispremt.txt`
- **Key Task**: Transform Quantitative variable into logarithm forms
- Other data preprocessing tasks: i) Group factor variables into fewer levels: `PCSPQ` (from 9 levels to 5 levels), `FAMIQ` (from 7 levels to 3 levels); ii) Treating too high `RELAT` values; iii) Replace `NA` values of `ROCNB` by 0, while delete observation with no values of `DMVTP`

Program `code_visprem.R`

- The input file is `visprempt.txt`, the ouput is `vispremv.txt`
- **Key Task**: Create the categorical-version (factor variables) of numeric variables, by `cut()` function, dividing the range of numeric variables into intervals (by thresholds or quantiles) and using these intervals as categories of variables.
- For `FAMIQ == "Finc"`, it is assigned randomly to either `"Fseu"` or `"Fcou"` by uniform distribution, mimic the proportion of `Fseu`:`Fcou` in the observed data ($P(FAMIG = "Fseu") \approx 0.45$)
- Create `familr` and `sexer`, which are numeric-version of `FAMIQ` and `SEXEQ`
- Re-arrange the table with: quantitative variables (information in numeric format) first, then qualitative variables (information in factor format), and finally `CARVP` variable.

Identify several features of the dataset:

```
nrow(vispremv) # number of col
```

```
## [1] 1063
```

```
ncol(vispremv) # number of variables
```

```
## [1] 55
```

```r
varquant = names(vispremv[sapply(vispremv,is.numeric)]) # features in quantities
varquant
```

```
##  [1] "familr" "sexer"  "RELAT"  "AGER"   "OPGNBL" "MOYRVL" "TAVEPL"
##  [8] "ENDETL" "GAGETL" "GAGECL" "GAGEML" "KVUNB"  "QSMOY"  "QCREDL"
## [15] "DMVTPL" "BOPPNL" "FACANL" "LGAGTL" "VIENB"  "VIEMTL" "UEMNB"
## [22] "XLGNB"  "XLGMTL" "YLVNB"  "YLVMTL" "ROCNB"  "NPTAG"  "ITAVCL"
## [29] "HAVEFL" "JNBJDL"
```

```r
varqual = names(vispremv[sapply(vispremv,is.factor)]) # features in quantities
varqual
```

```
##  [1] "SEXEQ"  "FAMIQ"  "PCSPQ"  "kvunbq" "vienbq" "uemnbq" "xlgnbq"
##  [8] "ylvnbq" "rocnbq" "nptagq" "endetq" "gagetq" "facanq" "lgagtq"
## [15] "havefq" "ageq"   "relatq" "qsmoyq" "opgnbq" "moyrvq" "dmvtpq"
## [22] "boppnq" "jnbjdq" "itavcq" "CARVP"
```

2.2. Training set - Test set

First, the purpose of this work is to build a predictive model, which should accurately classify the new (unobserved data). Meanwhile, the classifier built in a data set is chosen by its accuracy in this whole data set, which does not tell us how well the classification models perform in unseen data. Consequently, there exist **over-fitting** problems. In other words, rather than generalizing to fit the new data, the models are specialized the structure of obsvered training data.

To solve that, we would like to split the whole data set into: i) **Training set** (to construct the classification models); ii) **Test set** (to assess the performance of predictive model in unseen data).

The above sub-sampling procedure is for the purpose of splitting the data into training and test sets. By constructing, the test set should be independent with the training set, but have the same distribution with the training set. Thus, we randomly draw 200 observations from the total observation to create the test set, and the disjoint remaining part is the training set.

```r
# size of training sets
nrow(visappt)
```

```
## [1] 863
```

```r
# size of test sets
nrow(vistest)
```

```
## [1] 200
```

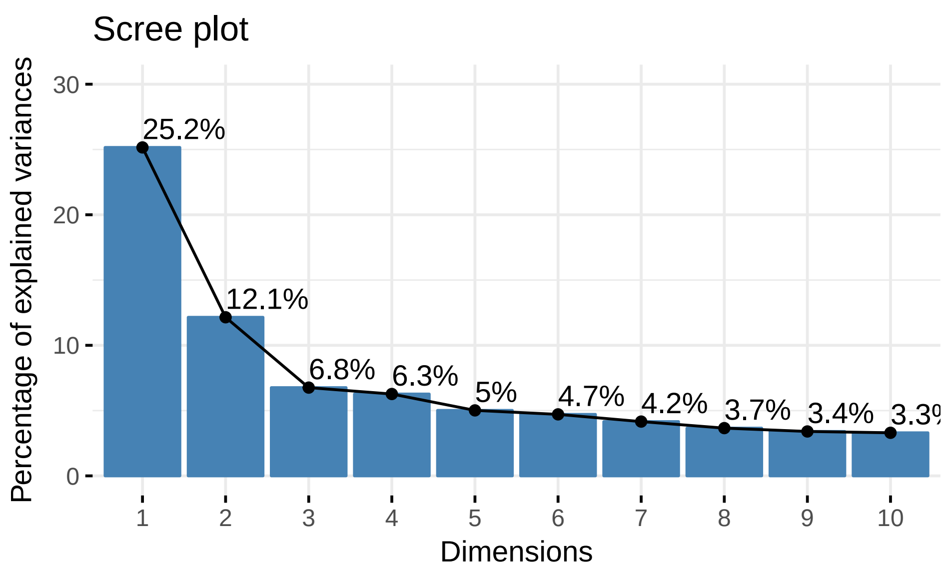
There are different rule-of-thumb for the ratio of training and test set size: 80:20, 66:34, 70:30. The increasing size of training set wil increase the ability of model to generalize, while the increasing size of test set enable us to have more data to test the robustness of the prediction.

In this case, we have 1069 obs., spliting into 863 obs. for the training, and 200 obs. for testing (80:20 for training:testing). The size of test sets is relatively enough to check how well the model perform. This could be a good starting point, also we would do cross-validation later.

2.3. Descriptive results

The high-dimensional characteristics may cause difficulties for the descriptive/visualize summary. Principal Component Analysis (PCA) would explain the variation of data by principal components, which are combination of original variables.

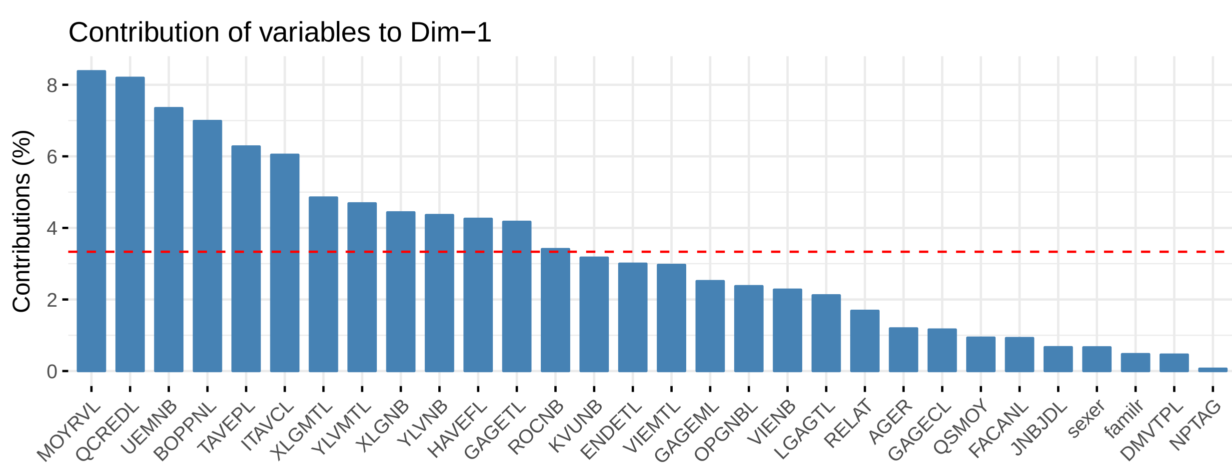
```r
# PCA is applied for numeric variables only, data is standardize (by center and scale)
pca.visapptr <- prcomp(visapptr[,-1], scale. =  TRUE, center = TRUE)

# Eigenvalues / Variances
factoextra::fviz_eig(pca.visapptr, addlabels = TRUE, ylim = c(0, 30))
```



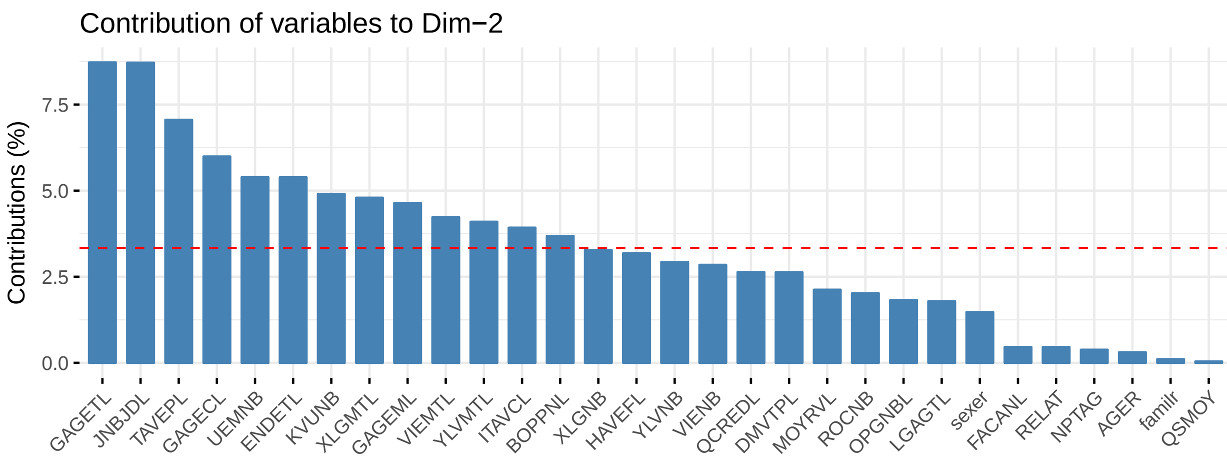
In this analysis, PC1 and PC2 explains 25.5% and 12.2% of the variation, respectively. Up to first 7 PCs, only 64.4% is explained.

We might also be interested in the contribution of each variables to important PC (say PC1 and PC2) in the plots below. The red dashed line indicates the expected average contribution. The more important variables in explaining the variability in the data set would have higher contribution to PCs.

```r
# contributions of variables to PC1
factoextra::fviz_contrib(pca.visapptr, choice = "var", axes=1)
```




```r
# contributions of variables to PC2
factoextra::fviz_contrib(pca.visapptr, choice = "var", axes=2)
```



3. Classification with CART

3.1. On Qualitative Variables

The `rpart` algorithm will recursively split the dataset, by finding the best split (j, s) over set T, attributes $x_1, ..., x_p$ until the terminate node. The best split is determined the best to minimize the impurity of splitted groups, or in other words, the largest possible reduction of heterogeneity (as pure as possible) in the predicted response variables.

The best split (j, s) solves: $argmin_{j,s} C(j, s) = N_1 Q_1 + N_2 Q_2$. Where Q_i is the impurity measurement, and N_i is number of observations in two splited groups.

The `"information"` is the spliting methods with the form of impurity measure: $Q_m = -\sum_{k=1}^{K} \hat{p}_{mk} log \hat{p}_{mk}$.

Furthermore, this alogirthm might face the over-growing problem that too complex tree would reduce the bias within the training sample, but have higher variance (trade-off of local bias and local variance). This leads to the over-fitting problem. Hence, we need a **stopping criterior**, which is when the improve in bias cannot compensate for the cost of complexity of the model.

Cost-complexity criterion: $C_\alpha = \sum_{m=1}^{M} N_m Q_m + \alpha M$ Where: M is the number of splits (represent the complexity of the tree), and $alpha$ is the penalty paramter for the complexity.

`cp` is the complexity parameter in `rpart`, which is scaled version of α over the misclassifcation rate of the overall data. By setting the `cp`, we set the rules for splits not meaningful, hence overcome the over-growing issues. For smaller values of `cp`, the tree grows deeper (more complex).

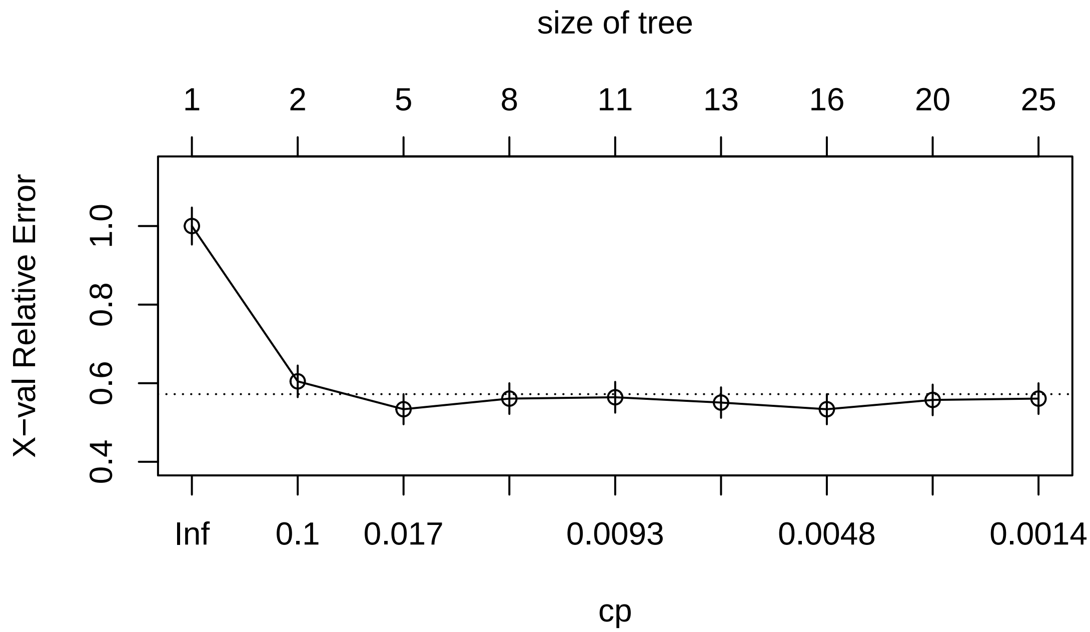
Conclusion for predicting well the visa variable: In this study, we have a large number of predictors, which makes the over-growing problem more likely. It might leads to the situation that we may have a very fitted model for the training set, but performs poorly in the test set. The `cp` value plays a crucial role in the shape of tree, as it should balance the reduction in bias and the increasing of complexity. It would be better if `cp` is selected by cross-validation procedure, rather than choosing arbitrarily.

3.2. Pruning

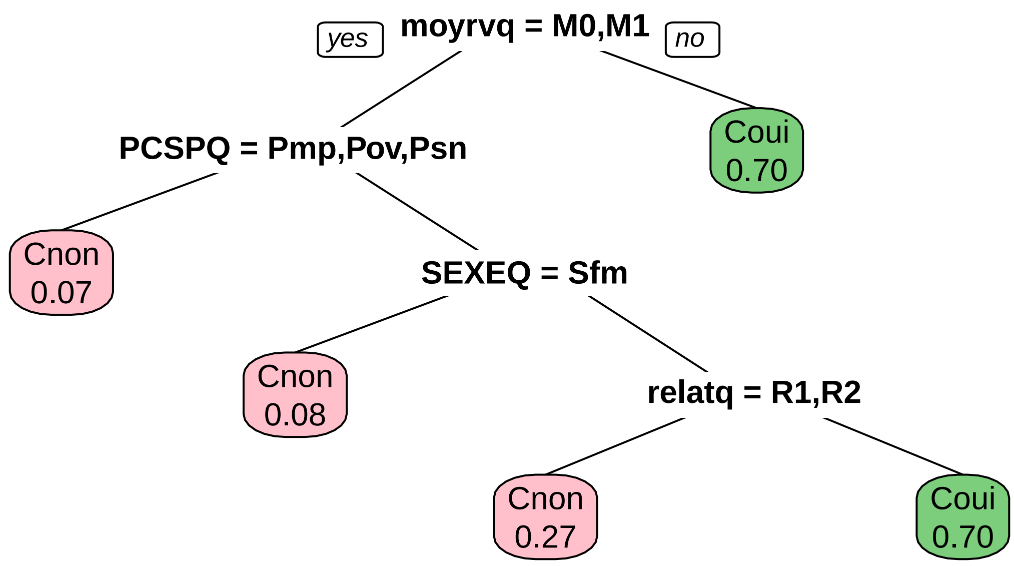
The decision on the size of tree could be convenient advised by the `plotcp()`, which plots the cross-validated relative error versus `cp`. The smallest tree with the best relative error have size = 5, and cp = 0.017.

We adopt it to prune our tree to obtain the `vis.treeq.cut`.

```r
# graphs of cross-validation results of different cp prunings of the tree
plotcp(vis.treeq)
```



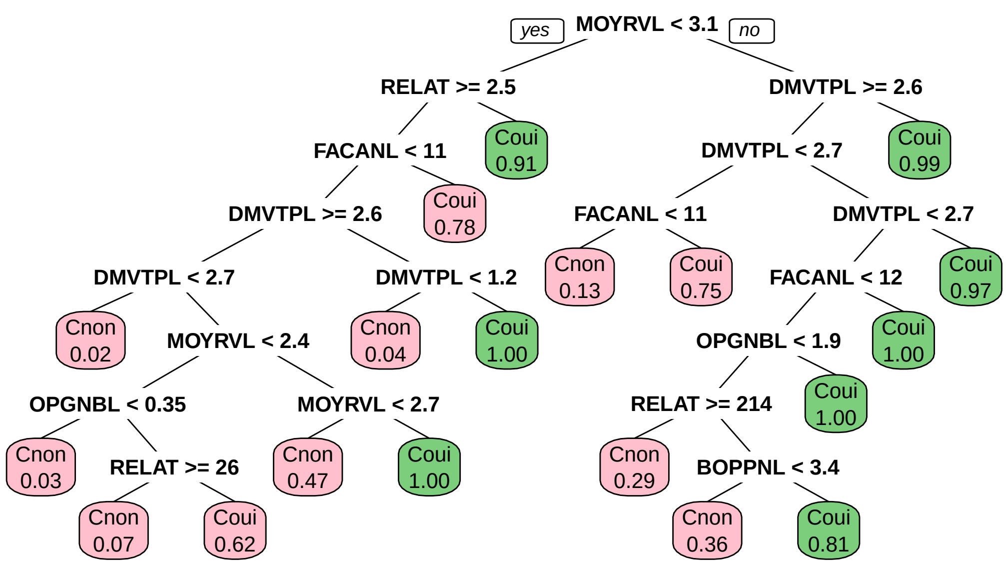

```r
# from the graphs, cp=0.017 is the optimal
vis.treeq.cut=prune(vis.treeq,cp=0.017)
# plot the tree after appropriate pruning
rpart.plot::prp(vis.treeq.cut, extra = 6,
                # report the predicted prob. of class
                box.palette=c("pink", "palegreen3")[vis.treeq.cut$frame$yval])
```



3.3. On Quantitative Variables

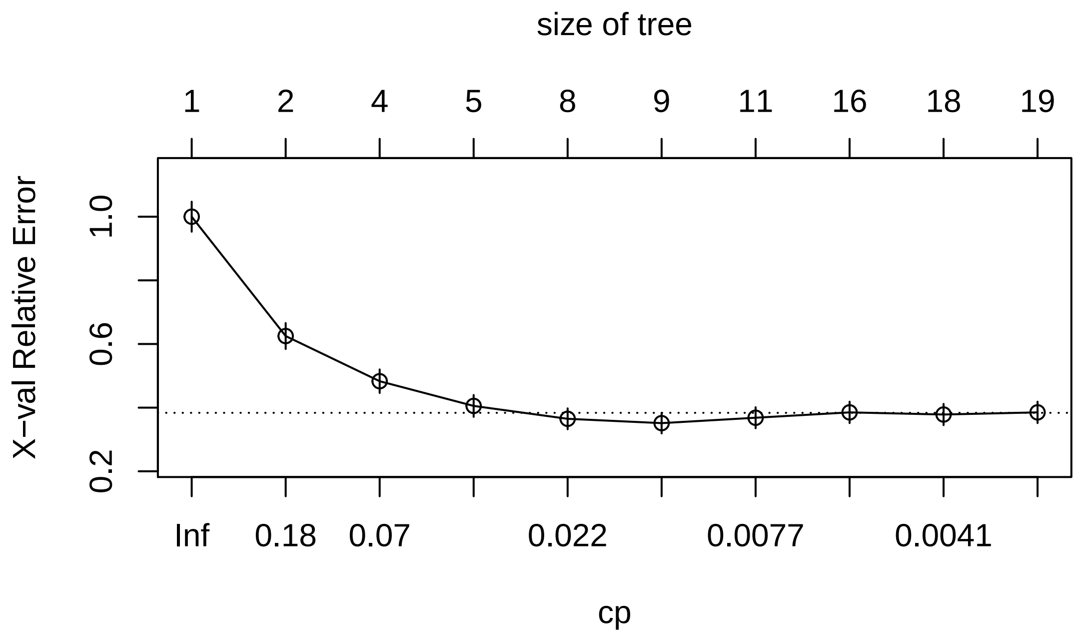
Different from the previous part, now we build the tree with quantitative variables (continuous input attributes). At each node of the tree, the data will be splitted by a continuous attribute, and a certain threshold. For instance, in the first node, the attribute is `MOYRVL` and the threshold is 3.1, the observation with the values below 3.1, is *yes* group, otherwise they belong to *no* group. For the remaining of theoretical part, it is similar to the previous cas in qualitative variables.

```
# We extract the quantitative variables
visapptr=visappt[,c("CARVP",varquant)] # traing sets
vistestr=vistest[,c("CARVP",varquant)] # test set

# Build the tree with the spliting method of 'information'
# and the pre-pruning at cp=0.001
set.seed(1234)
vis.treer=rpart(CARVP~.,data=visapptr,
                parms=list(split="information"),cp=0.001)

# graphical presentation of tree, at cp=0.001
rpart.plot::prp(vis.treer, extra = 6,
                box.palette=c("pink", "palegreen3")[vis.treeq.cut$frame$yval])
```

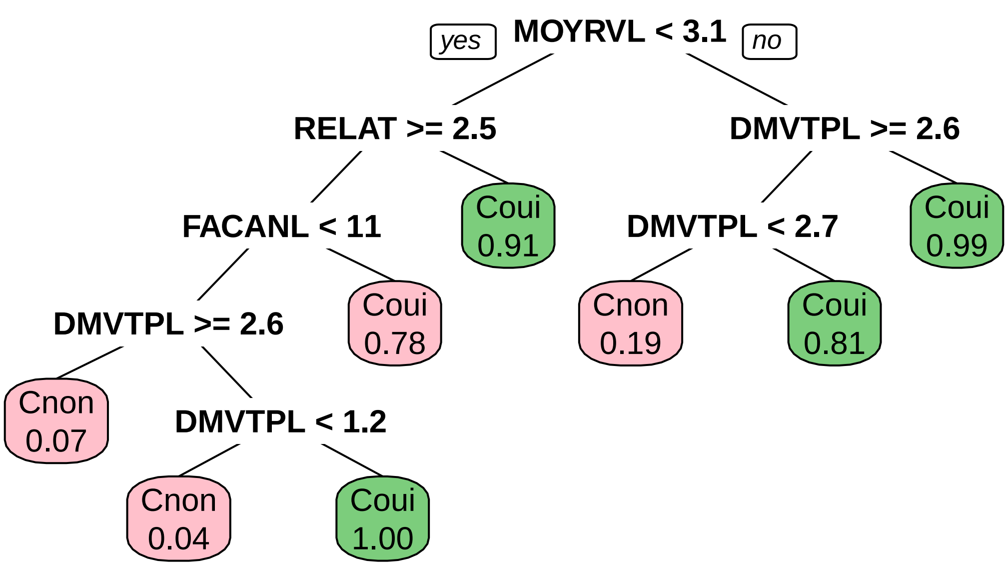


The tree (as in the graph) grows very deep, as we put the `cp` quite low. To avoid the over-fitting, it is motivated to prune the tree at the optimal `cp` value. The various values of cp and corresponding relative error is presented in the graph. The best value of `cp` which gives the lowest relative error is 0.022. We choose that value to 'prune' the tree.

```
# plot the cp to check the optimal cp by cross-valiation relative error
plotcp(vis.treer)
```




```
# From the best value of cp=0.022, we prune the tree
vis.treer.cut=prune(vis.treer,cp=0.022)

# tree after pruning
rpart.plot::prp(vis.treer.cut, extra = 6,
                box.palette=c("pink", "palegreen3")[vis.treeq.cut$frame$yval])
```



3.4. Important variables

The classification tree will "drop" the variables not significantly relevent to the decision outcomes. We could say that the attributes kept in the tree are important factors. From the graphical representation of the tree, it seems that important variables for the prediction are: `MOYRVL`, `DMVTPL`, `RELAT`, `FACANL`

3.5. Pruning with Cross-validation

The cross-valiation (CV) is the procedure to assess the accuracy of the prediction in an unseen data set (how it generalize). The CV is the method enabing us to define the "test" dataset, from different subsets of the sample. The principal of CV is partitioning the sample of all observations in the data set to complementary subsets, using one subset as *training set* and the other as a *validation set*. This step is repeated for several times, for different ways of participating training set and validation set.

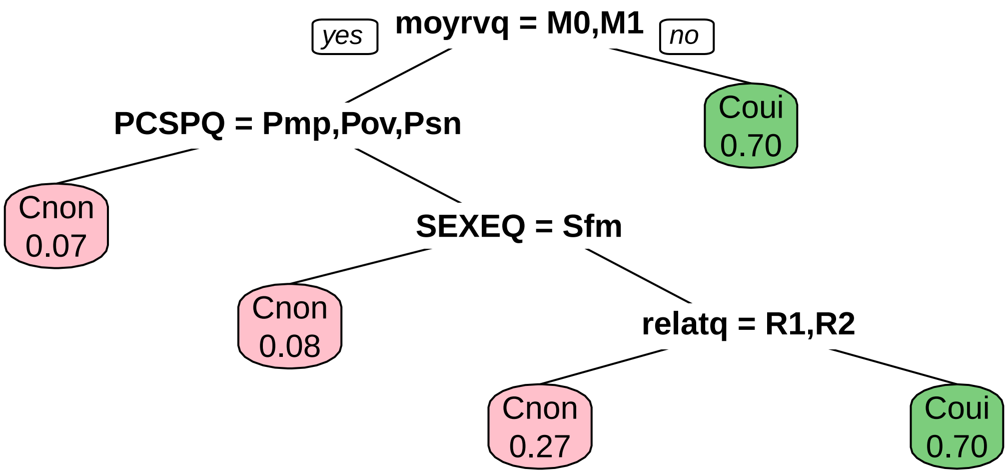
In this particular situation, it randomly divides the data into 10 subsets (`xval=10`). Out of this 10 subsets, one set is used as test set, the remaining 9 subsets are used as training data. The process repeats for 10 times, each subset is used once as test set.

By this CV procedure, all observations are used in both training and testing. It is a powerful techniques when the number of observations in the data is limited, the significance of predictive modeling in training set and testing capacities in test set are unfavorable due to the small number of observations in both sets.

```
# return cross-validated prediction
set.seed(1234)
xmat=xpred.rpart(vis.treeq, xval=10, # xval: number of cv group
                 cp = seq(0.05,0.001,length=20)) # set of cp values

# return the false values
xerr = as.integer(visapptq$CARVP)!=xmat
error = apply(xerr,2,sum)/nrow(xerr)

# return the cp with the smallest error rate
id = which(error == min(error))[1]
cp_list = seq(0.05,0.001,length=20) # set of tested values
cp.op = cp_list[id]
cp.op  # to obtain best cp
```

```
## [1] 0.02678947
```

```
# build the tree with optimal pruning
set.seed(1234)
vis.treeq.cut2=rpart(CARVP~.,data=visapptq,
                     parms=list(split="information"),cp=cp.op)

# graphical tree
rpart.plot::prp(vis.treeq.cut2, extra = 6,
                box.palette=c("pink", "palegreen3")[vis.treeq.cut$frame$yval])
```



3.6. Prediction on the test set

```
# prediction of tree by qualitative attributes
pred.vistestq=predict(vis.treeq.cut2,
                      newdata=vistestq,type="class")
# confusion matrix
table(pred.vistestq,vistestq$CARVP)
```

```
##
## pred.vistestq Cnon Coui
##          Cnon  113   15
##          Coui   26   46
```

```
# prediction of tree by quantitative attributes
pred.vistestr=predict(vis.treer.cut,
                      newdata=vistestr,type="class")
# confusion matrix
table(pred.vistestr,vistestr$CARVP)
```

```
##
## pred.vistestr Cnon Coui
##          Cnon  133   14
##          Coui    6   47
```

We compute the error classification on the test set (created in section 2.2, `n=200`). The output is the confusion matrix, in which we present the table of predicted results by the tree models with the actual values in the test set.

- There are two possible predicted class: `Cnon` and `Coui`
- For the tree with qualitative input variables `vis.treeq.cut2`, there are 113 `Cnon` predictions by the model which are truly `Cnon` (true negative), and 46 accurately predicted `Coui` (true positive). *The accuracy rate is 79.5%.*
- For the tree with quantitative input variables `vis.treer.cut`, there are 133 true `Cnon` predictions, and 47 true `Coui` prediction. *The accuracy rate is 90%.*

Generally, the `vis.treeq.cut` performs better than `vis.treeq.cut2`.

4. Random Forest

4.1. Remainders

Principal: Random Forest (RF) grows tree for different random sampling sets of the orginal data. It combines the results of individual trees (weak leaners) to improve the generalization ability of the model (strong learners).

Important steps:

1. Sampling the observations of the training set from the original data. This is the training set for growing a tree.
2. Randomly choose k (`mtry`) out of p input variables ($k << p$), k variables are considered at each node for the best split
3. By the random training data and features set, we grow a tree. Different from tree classification, there is no prunning, each tree in the forest is grown fully.
4. Repeat step1-3, many tree are growns.
5. From the RF, the final prediction is ensembled from the individual prediction of each tree by "majority vote" rule.

The major improvement of RF comparing to CARTs: is the reduction of over-fitting. Reaching the final prediction by the "majority vote", RF could combine the strengths and compensate the weakness by the complementaty of different trees. By the diversity, it captures the repeated important pattern and recognize the subtle interesting pattern in the data set.

Important parameters:

- **`mtry`**: as discussed before, `mtry` is the number of variables at each node, to be selected for the best split. Normally, $mtry = int(sqrt(p))$, but it should be check the cross-validation for the optimal value.
- **`ntree`**: the number of trees in the forest. The higher `ntree`, the accuracy would be improve in some certain, but the more computationally expensive to construct the RF. Therefore, we want to balance the `ntree` to the optimal points that the further increasing number of tree is not significantly economical (diminishing return).
- **`nodesize`**: The minimum number of observations at the terminal nodes. This parameter will control the depth of tree. The smaller `nodezie`, the deeper tree. But, with lower tree depth, the tree might fail to realize subtle patterns of the data.

4.2. Random Forest with R

```r
set.seed(1234)
fit2 = randomForest::randomForest(CARVP~.,data=visapptr,
                 xtest=vistestr[,varquant],ytest=vistestr[,"CARVP"],
                 importance=TRUE,norm.vote=FALSE,ntree=500)
print(fit2)
```

```
##
## Call:
##  randomForest(formula = CARVP ~ ., data = visapptr, xtest = vistestr[,      varquant], ytest = vistes
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 5
##
```

```
##         OOB estimate of  error rate: 10.66%
## Confusion matrix:
##      Cnon Coui class.error
## Cnon  521   46  0.08112875
## Coui   46  250  0.15540541
##                 Test set error rate: 12.5%
## Confusion matrix:
##      Cnon Coui class.error
## Cnon  127   12  0.08633094
## Coui   13   48  0.21311475
```

We have the confusion matrix in OBB and Test set. For each tree, after the sampling for the training data, there are remaining observations ("Out-of-Bag") which are used to estimate the accuracy of prediction. The error rate are about 11% and 12%, respectively in the OBB and test set.

Overfitting is the situation that the prediction by the model is too specific for the training data (fitting very closely) but lack of generalization (fitting poorly for the new unseen data set). It exists because the criterion used for selecting the model is based on the training data, while its purpose is to predict well in unobserved dat.

It happens when the complexity of the model is high, which makes it lack of flexibility, and so difficult to fit the new data set. The typical example is in the model with too many predictors. Another example is in the classification tree with high depth (not appropriate pruning).

In general, the RF is more robust to overfitting than other model. But, in this case, the important tuning parameters has not optimized. In fact, the small default values of mtry (`mtry=5`) could cause the overfitting, as the model is less flexible than it should be when the set of considered attributes at each node is limited. We will try the cross-validation to choose the optimal paramter in next step.

4.3. Optimization with Cross-validation

```r
# return the best mtry and ntree
set.seed(1234)
res=tune(randomForest, CARVP~., data=visapptr,
         tunecontrol=tune.control(sampling="cross",cross=10),
         ranges=list(mtry=c(10,17,24),ntree=c(200,400,600)))

param = res$best.parameters

# using the optimal tuning paramters for RF
set.seed(1234)
fit3=randomForest(CARVP~.,data=visapptr,
                  xtest=vistestr[,varquant],ytest=vistestr[,"CARVP"],
                  importance=TRUE,norm.vote=FALSE,ntree=param$ntree, mtry=param$mtry)
print(fit3)
```

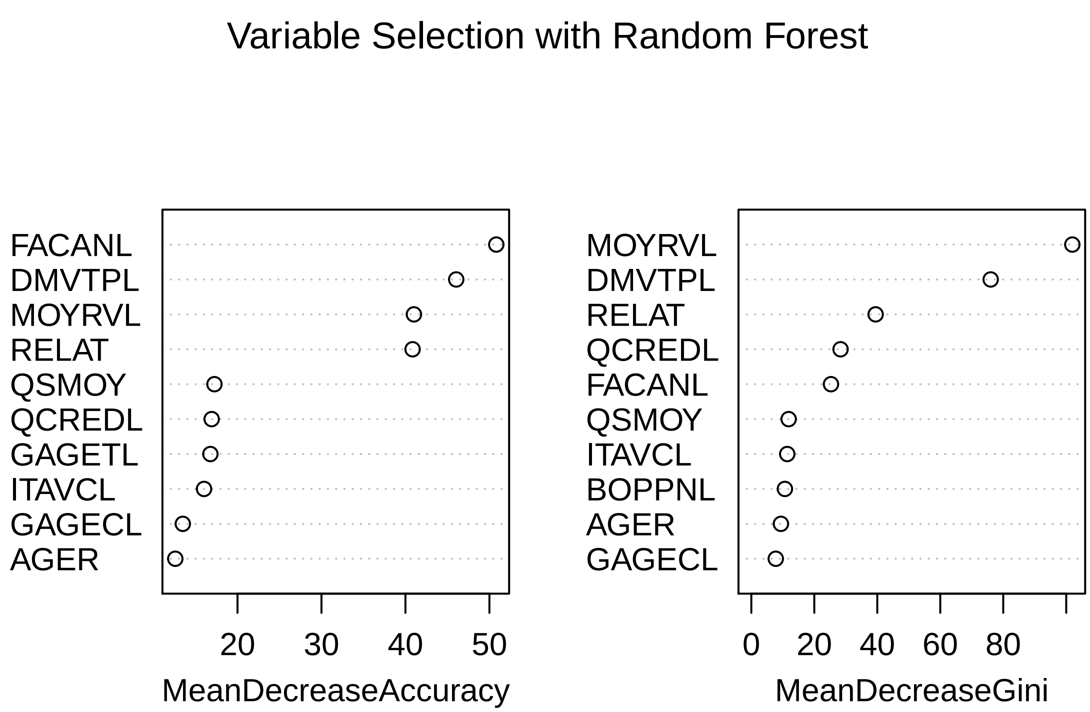
```
##
## Call:
##  randomForest(formula = CARVP ~ ., data = visapptr, xtest = vistestr[,      varquant], ytest = vistes
##                Type of random forest: classification
##                      Number of trees: 400
## No. of variables tried at each split: 24
##
##         OOB estimate of  error rate: 10.08%
## Confusion matrix:
```

```
##      Cnon Coui class.error
## Cnon  531   36  0.06349206
## Coui   51  245  0.17229730
##                Test set error rate: 10.5%
## Confusion matrix:
##      Cnon Coui class.error
## Cnon  131    8  0.05755396
## Coui   13   48  0.21311475
```

The misclassification error rate is about 10.5% on the test set, improved comparing to the previous RF. We predict 131 true `Cnon` out of 139, and 48 true `Coui` out of 61 in the test set. The random forest produce a better prediction than un-tunning RF and the previous optimal classification tree for qualitative variables.

4.4. Variable Selection with RF

```
varImpPlot(fit3, sort=TRUE, n.var=min(10, nrow(fit3$importance)),
           main = "Variable Selection with Random Forest")
```



The variable selection is available in the RF because each tree in the RF selects the best available split based on its set of features at nodes (random sampled). The available tool in R package enables us to plot the importance of variables by:

- **Mean Decrease Accuracy:** the mean decrease accuracy over all out-of-bag cross-validation predictions, when a given variable is removed. The variables with higher mean decrease accuracy is more important.
- **Mean Decrease Gini:** Gini importance measures the average gain of purity by splits at the given variable. If the feature is relevent, it will better split nodes into purer. Meanwhile, the split at unimportant variables will not increase the purity.

Similar to our first glance in the importance of tree **Section 3.4**. `MOYRVL`, `DMVTPL`, `RELAT` and `FACANL` are in the top important variables in both terms of Mean Decrease Accuracy and Mean Decrease Gini, while the order of importance is slightly different.

5. Logistic Regression

5.1. On the Visa Premier Dataset

Regression method is to fit a particular family of functions (graphical as line or curve) to data, to represent the relationship between the dependent variables and input variables.

Logistic Regression is indeed a regression method, the family of function in this case is logistic function. Our dependent variable is binary `Cnon` and `Coui`. The logistic function to fit the training data has the shape of S-curve with the outcome values between 0 and 1, as the probability to belong to one class.

Logistic funtion has the form: $f(x) = \frac{1}{1+e^{-index}}$ The value of index determined the value of $f(x)$ varied between 0 and 1, the index is constructed as the linear combination of input variables, with the coefficients β (weights of input variables): $index = X'\beta$.

Our interest is to estimate the coefficient β, then we could used the estimated logistic function with estimated coefficients to predict the dependent variable `CARVP` by the future input variables.

The estimated parameters of the logistic model is obtained by the **maximum likelihood methods**. The objective function is to maximize the log likelihood of the paramteres giving the observed outcomes and input variables in the training data set: $L(\beta; x, y)$. It is equivalent to minimizing the negative log-likelihood, which is the Cost function: $J(\beta; x, y)$.

The coefficients are chosen to minimize the classification error of logistic model comparing to the actual observations in the training data, which is quantifies by **Cost function:** $\min_\beta J(\beta)$.

The technical algorithm behind that is the Gradient Descent. Analytically, to obtain the optimal value of the objective function, we will look for the derivative with respect to the coefficients to be equal to zero (First-order condition). It is also analytically difficult. Instead, we use the Gradient Descent method that: Start at a tentative starting values of parameters, then iteratively update the coefficients by calculating the gradient of the objective function to get as close as possible to zero. By that, we identify the optimal coeficients for the minimum errors.

```
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:dplyr':
##
##     select
```

```
var=names(vispremv)
varquant=var[1:30]
varqual=var[31:54]
visapptq=visappt[,c("CARVP",varqual)]
vistestq=vistest[,c("CARVP",varqual)]

#vraisemblance.
visa.logit=glm(CARVP ~.,data=visapptq,
               family=binomial, na.action=na.omit)
```

5.2. Variable Selection

We use the `anova()` to compute variance analysis to compare among models. Given a sequence of objects, `anova` tests the model againts one anther by the specific model. In the below table, we could see that some model terms does not have significant results. It means that they are not relevent in the appropriate model.

```
anova(visa.logit,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     862    1109.82
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## FAMIQ   1    5.630       860    1005.80 0.0176606 *
## PCSPQ   4  119.732       856     886.07 < 2.2e-16 ***
## kvunbq  1   36.084       855     849.98 1.890e-09 ***
## vienbq  1   10.885       854     839.10 0.0009692 ***
## uemnbq  2    6.951       852     832.15 0.0309452 *
## xlgnbq  2    1.377       850     830.77 0.5024544
## ylvnbq  2    1.701       848     829.07 0.4272685
## rocnbq  1    8.475       847     820.59 0.0035997 **
## nptagq  1    8.735       846     811.86 0.0031215 **
## endetq  1    1.731       845     810.13 0.1883148
## gagetq  1   13.928       844     796.20 0.0001900 ***
## facanq  1    8.317       843     787.88 0.0039272 **
## lgagtq  1    0.003       842     787.88 0.9585077
## havefq  1    4.714       841     783.17 0.0299227 *
## ageq    2    1.199       839     781.97 0.5489655
## relatq  2   27.518       837     754.45 1.058e-06 ***
## qsmoyq  2   34.776       835     719.67 2.809e-08 ***
## opgnbq  2   19.222       833     700.45 6.700e-05 ***
## moyrvq  2   67.145       831     633.31 2.628e-15 ***
## dmvtpq  2   32.202       829     601.10 1.017e-07 ***
## boppnq  2    5.156       827     595.95 0.0759118 .
## jnbjdq  2    0.914       825     595.03 0.6331772
## itavcq  2   14.429       823     580.60 0.0007360 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
visa.logit=glm(CARVP ~.,data=visapptq,
               family=binomial, na.action=na.omit)

visa.step <-step(visa.logit) # step-wise regression??, by AIC
```

```
anova(visa.step,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
```

```
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                      862    1109.82
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## PCSPQ   4  125.289       857     886.14 < 2.2e-16 ***
## kvunbq  1   36.149       856     849.99 1.828e-09 ***
## uemnbq  2   11.959       854     838.03 0.0025297 **
## nptagq  1   11.217       853     826.81 0.0008106 ***
## endetq  1    2.684       852     824.13 0.1013519
## gagetq  1   15.526       851     808.60 8.136e-05 ***
## facanq  1    9.503       850     799.10 0.0020511 **
## havefq  1    9.744       849     789.36 0.0017990 **
## relatq  2   20.203       847     769.15 4.102e-05 ***
## qsmoyq  2   38.961       845     730.19 3.465e-09 ***
## opgnbq  2   20.382       843     709.81 3.752e-05 ***
## moyrvq  2   70.040       841     639.77 6.181e-16 ***
## dmvtpq  2   33.396       839     606.37 5.599e-08 ***
## itavcq  2   16.484       837     589.89 0.0002634 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

By the step-wise AIC-based procedure, we obtain much better model, after eliminating irrelevant terms.

The principal of step-wise variable selection in the logistic regression is that: Start with the full model consisting of all input variables, at each following stage certain variables would be removed or added in the model in the manner to obtain better AIC. At the end, it will return the best model in terms of minimizing AIC.

The AIC (Akaike Information Criterion) is the indications of relative quality of statistical models for a given training data.

$$AIC = -2log(likelihood) + 2k$$

Where: $likelihood$ represent the fitness of the model: $nlog\left(\frac{RSS}{n}\right)$, while k is the number of coefficients involved in the model.

AIC represents the tradeoff of the goodness of fit and the simplicity of the model. In the larger model, the RSS will reduce and the $likelihood$ increase. Yet, higher k will increase the magnitude of AIC. The better model will balance this trade-off and obtain minimum AIC.

5.3. Calibration with Cross Validation

Now, we do the average performance of the logistic regression with a CV algorithm. Starting with the full model, we iterate the process:

1. Run the logistic model with the set of input variables
2. Compute the CV prediction error
3. By anova, figure out the irrelevant variable
4. Eliminate the irrelevant variable. Repeat 1-3.

The iteration stops when the CV classification rate is worse, comparing to the previous step.

```r
library(boot)

# Iteration 1
visa1.logit=glm(CARVP~SEXEQ+PCSPQ+kvunbq+uemnbq+nptagq+
                  endetq+gagetq+facanq+havefq+relatq+qsmoyq+opgnbq+
                  moyrvq+dmvtpq+boppnq+jnbjdq+itavcq, data=visapptq,
                family=binomial, na.action=na.omit)

set.seed(1234)
cv.glm(visapptq,visa1.logit,K=10)$delta[1] # CV estimated prediction error
```

```
## [1] 0.1191089
```

```r
anova(visa1.logit,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     862    1109.82
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## PCSPQ   4  125.289       857     886.14 < 2.2e-16 ***
## kvunbq  1   36.149       856     849.99 1.828e-09 ***
## uemnbq  2   11.959       854     838.03 0.0025297 **
## nptagq  1   11.217       853     826.81 0.0008106 ***
## endetq  1    2.684       852     824.13 0.1013519
## gagetq  1   15.526       851     808.60 8.136e-05 ***
## facanq  1    9.503       850     799.10 0.0020511 **
## havefq  1    9.744       849     789.36 0.0017990 **
## relatq  2   20.203       847     769.15 4.102e-05 ***
## qsmoyq  2   38.961       845     730.19 3.465e-09 ***
## opgnbq  2   20.382       843     709.81 3.752e-05 ***
## moyrvq  2   70.040       841     639.77 6.181e-16 ***
## dmvtpq  2   33.396       839     606.37 5.599e-08 ***
## boppnq  2    4.485       837     601.89 0.1061979
## jnbjdq  2    1.414       835     600.48 0.4931785
## itavcq  2   15.099       833     585.38 0.0005263 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Iteration 2: Drop jnbjdq
visa2.logit=glm(CARVP~SEXEQ+PCSPQ+kvunbq+uemnbq+nptagq+
                  gagetq+facanq+havefq+relatq+qsmoyq+opgnbq+moyrvq+dmvtpq+
                  boppnq+endetq+itavcq, data=visapptq,
                family=binomial, na.action=na.omit)
set.seed(1234)
cv.glm(visapptq,visa2.logit,K=10)$delta[1]
```

```
## [1] 0.1190024
```

```r
anova(visa2.logit,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     862    1109.82
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## PCSPQ   4  125.289       857     886.14 < 2.2e-16 ***
## kvunbq  1   36.149       856     849.99 1.828e-09 ***
## uemnbq  2   11.959       854     838.03 0.0025297 **
## nptagq  1   11.217       853     826.81 0.0008106 ***
## gagetq  1   13.344       852     813.47 0.0002592 ***
## facanq  1    9.499       851     803.97 0.0020553 **
## havefq  1    9.418       850     794.55 0.0021483 **
## relatq  2   21.086       848     773.47 2.638e-05 ***
## qsmoyq  2   37.442       846     736.02 7.407e-09 ***
## opgnbq  2   19.120       844     716.90 7.051e-05 ***
## moyrvq  2   70.202       842     646.70 5.698e-16 ***
## dmvtpq  2   32.290       840     614.41 9.734e-08 ***
## boppnq  2    4.295       838     610.12 0.1167789
## endetq  1    8.227       837     601.89 0.0041268 **
## itavcq  2   14.389       835     587.50 0.0007505 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Iteration 3: Drop boppnq
visa3.logit=glm(CARVP~SEXEQ+PCSPQ+kvunbq+uemnbq+nptagq+
                  gagetq+facanq+havefq+relatq+qsmoyq+opgnbq+moyrvq+dmvtpq+
                  endetq+itavcq, data=visapptq,
                family=binomial, na.action=na.omit)
set.seed(1234)
cv.glm(visapptq,visa3.logit,K=10)$delta[1] # BEST!!!
```

```
## [1] 0.1184061
```

```r
anova(visa3.logit,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     862    1109.82
```

```
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## PCSPQ   4  125.289       857     886.14 < 2.2e-16 ***
## kvunbq  1   36.149       856     849.99 1.828e-09 ***
## uemnbq  2   11.959       854     838.03 0.0025297 **
## nptagq  1   11.217       853     826.81 0.0008106 ***
## gagetq  1   13.344       852     813.47 0.0002592 ***
## facanq  1    9.499       851     803.97 0.0020553 **
## havefq  1    9.418       850     794.55 0.0021483 **
## relatq  2   21.086       848     773.47 2.638e-05 ***
## qsmoyq  2   37.442       846     736.02 7.407e-09 ***
## opgnbq  2   19.120       844     716.90 7.051e-05 ***
## moyrvq  2   70.202       842     646.70 5.698e-16 ***
## dmvtpq  2   32.290       840     614.41 9.734e-08 ***
## endetq  1    8.037       839     606.37 0.0045827 **
## itavcq  2   16.484       837     589.89 0.0002634 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# Iteration 4: Drop endetq
visa4.logit=glm(CARVP~SEXEQ+PCSPQ+kvunbq+uemnbq+nptagq+
                  gagetq+facanq+havefq+relatq+qsmoyq+opgnbq+moyrvq+dmvtpq+
                  itavcq, data=visapptq,
                  family=binomial, na.action=na.omit)
set.seed(1234)
cv.glm(visapptq,visa4.logit,K=10)$delta[1] # worse
```

```
## [1] 0.1199755
```

```r
anova(visa4.logit,test="Chisq")
```

```
## Analysis of Deviance Table
##
## Model: binomial, link: logit
##
## Response: CARVP
##
## Terms added sequentially (first to last)
##
##
##        Df Deviance Resid. Df Resid. Dev  Pr(>Chi)
## NULL                     862    1109.82
## SEXEQ   1   98.388       861    1011.43 < 2.2e-16 ***
## PCSPQ   4  125.289       857     886.14 < 2.2e-16 ***
## kvunbq  1   36.149       856     849.99 1.828e-09 ***
## uemnbq  2   11.959       854     838.03 0.0025297 **
## nptagq  1   11.217       853     826.81 0.0008106 ***
## gagetq  1   13.344       852     813.47 0.0002592 ***
## facanq  1    9.499       851     803.97 0.0020553 **
## havefq  1    9.418       850     794.55 0.0021483 **
## relatq  2   21.086       848     773.47 2.638e-05 ***
## qsmoyq  2   37.442       846     736.02 7.407e-09 ***
## opgnbq  2   19.120       844     716.90 7.051e-05 ***
## moyrvq  2   70.202       842     646.70 5.698e-16 ***
## dmvtpq  2   32.290       840     614.41 9.734e-08 ***
## itavcq  2   15.415       838     599.00 0.0004495 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```r
# stop when the CV estimater prediction error is worse.
```

We have gone through 4-iteration, with the CV rates are: 0.11911, 0.11900, **0.11841**, and 0.11997, respectively. We stopped at Iteration 3, when the CV rate become worse in next iteration. The best model is `visa3.logit`.

We use this best model to predict on the test set, classifying into `Cnon` and `Coui` at the threshold of probability 0.5. The confusion matrix is below. The accuracy is 78.5%, while the error rate in test set is 21.5%, which is better than the prediction of tree classification on qualitative inpur variables.

```r
# Use your model to make predictions
p.vistestq = predict(visa3.logit, newdata = vistestq, type = "response")
p.vistestq = as.numeric(p.vistestq > 0.5) %>% factor(labels = c("Cnon", "Coui"))


# use caret and compute a confusion matrix
caret::confusionMatrix(data = p.vistestq, reference = vistestq$CARVP)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction Cnon Coui
##       Cnon  121   25
##       Coui   18   36
##
##                Accuracy : 0.785
##                  95% CI : (0.7215, 0.8398)
##     No Information Rate : 0.695
##     P-Value [Acc > NIR] : 0.00286
##
##                   Kappa : 0.476
##  Mcnemar's Test P-Value : 0.36020
##
##             Sensitivity : 0.8705
##             Specificity : 0.5902
##          Pos Pred Value : 0.8288
##          Neg Pred Value : 0.6667
##              Prevalence : 0.6950
##          Detection Rate : 0.6050
##    Detection Prevalence : 0.7300
##       Balanced Accuracy : 0.7303
##
##        'Positive' Class : Cnon
##
```

6. Support Vector Machine

6.1. First run

The SVM aims to maximize the statistical efficiency of classification, by maximize the **"margin"** classification. Margin, in fact, is the distance between a data point to the hyperplane to seperate the data set.

Kernel: The linear equation of this hyperplane is: $\langle\beta, x\rangle + \beta_0 = 0$. In fact, the SVM classifier is not likely linear, hence we enable non-linear SVMs by kernel function $K(x, x')$.
The classifier is:

$$f(x) = sign\left(\sum_{i=1}^{N} \alpha_i y_i K(x_i, x') + \beta_0\right)$$

By kernelization, we send the non-seperable data into higher-dimensional, which enables the classification.

Gamma: It is the free parameters of kernel function, represents how far the influence of a support vector. Small `gamma` means wide-spread influences, leading to high bias and low variance, and vice-versa.

Cost: The objective function is:

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N} \xi_i$$

$$st. y_i(\beta' h(x_i) + \beta_0) \geq 1 - \xi_i$$

$h(.)$ is the mapping to higher dimensional Euclidean Space. ξ_i is the slack variables which allows some irrelevent data points could be ignored,for better over-fitting.

The penalized parameter for using slack variables is `Cost`, which is the error penalty for the generalized stability. With higher `cost`, the penalty for slack variables is largers, which means more data points are involved as support vectors. This is the trade-off between misclassification and the simplicity of the model.

In this class, we have two classes to seperate.

```
# Build SVM
vis.svm=svm(CARVP~., data=visapptr)
summary(vis.svm)
```

```
##
## Call:
## svm(formula = CARVP ~ ., data = visapptr)
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.03333333
##
## Number of Support Vectors:  478
##
##  ( 237 241 )
##
##
## Number of Classes:  2
##
## Levels:
##  Cnon Coui
```

```
# Confusion matrix on training set
vis.pred=predict(vis.svm,data=visapptr)
table(vis.pred,visapptr$CARVP)
```

```
##
## vis.pred Cnon Coui
```

```
##     Cnon  542   44
##     Coui   25  252
```

```r
# Confusion matrix on test set
vis.pred=predict(vis.svm,newdata=vistestr)
table(vis.pred,vistestr$CARVP)
```

```
##
## vis.pred Cnon Coui
##     Cnon  124   22
##     Coui   15   39
```

We build the support vector machine model by `svm()`, which automatically choose tuning parameters. The output model is using `radial` as kernel type, with the `cost = 1` and `gamma = 0.0333`.

- The `cost` plays the role as penalized parameters, so only 482 out of 863 observations are used as support vectors, with 238 and 244 for each of two classes.
- The error rate in test set is unsurprisingly higher than the error rate in training set

6.2. Optimization

We try to optimize the SVM by grids of `gamma`, `cost`, and different kernels (polynomial, sigmoid, and radius). Among them, the svm with `radius` kernel, with the miss-classification rate of 19%

```r
# Try a relatively good model
vis.svm=svm(CARVP~., data=visapptr,
            gamma=0.015,cost=6)
# Confusion matrix on test set
vis.pred=predict(vis.svm,newdata=vistestr)
table(vis.pred,vistestr$CARVP)
```

```
##
## vis.pred Cnon Coui
##     Cnon  124   22
##     Coui   15   39
```

```r
# 1. Tuning with radial kernel
tune1 = tune(svm, CARVP~., data=visapptr,
         kernel="radial", ranges=list(cost=4:6, gamma=seq(0.015, 0.02, by = 0.001)))
tune1$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = CARVP ~ ., data = visapptr,
##     ranges = list(cost = 4:6, gamma = seq(0.015, 0.02, by = 0.001)),
##     kernel = "radial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  radial
##        cost:  4
##       gamma:  0.015
##
## Number of Support Vectors:  413
```

```r
vis.pred=predict(tune1$best.model,newdata=vistestr)
table(vis.pred,vistestr$CARVP)
```

```
##
## vis.pred Cnon Coui
##     Cnon  123   23
##     Coui   16   38
```

```r
# 2. Tuning with polynomial kernel
tune2 = tune(svm, CARVP~., data=visapptr,
     kernel="polynomial", ranges=list(cost=4:6, gamma=seq(0.015, 0.02, by = 0.001)))
tune2$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = CARVP ~ ., data = visapptr,
##     ranges = list(cost = 4:6, gamma = seq(0.015, 0.02, by = 0.001)),
##     kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  6
##      degree:  3
##       gamma:  0.018
##      coef.0:  0
##
## Number of Support Vectors:  520
```

```r
vis.pred=predict(tune2$best.model,newdata=vistestr)
table(vis.pred,vistestr$CARVP)
```

```
##
## vis.pred Cnon Coui
##     Cnon  130   42
##     Coui    9   19
```

```r
# 3. Tuning with sigmoid kernel
tune3 = tune(svm, CARVP~., data=visapptr,
     kernel="sigmoid", ranges=list(cost=4:6, gamma=seq(0.015, 0.02, by = 0.001)))
tune3$best.model
```

```
##
## Call:
## best.tune(method = svm, train.x = CARVP ~ ., data = visapptr,
##     ranges = list(cost = 4:6, gamma = seq(0.015, 0.02, by = 0.001)),
##     kernel = "sigmoid")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  sigmoid
##        cost:  4
##       gamma:  0.016
##      coef.0:  0
```

```
##
## Number of Support Vectors:  382
```

```r
vis.pred=predict(tune3$best.model,newdata=vistestr) # test set
table(vis.pred,vistestr$CARVP)
```

```
##
## vis.pred Cnon Coui
##     Cnon  114   26
##     Coui   25   35
```

6.3. Conclusion

The SVM is a strong method, because:

- It could take into account the sparsity to prevent the over-fitting.
- Using the kernel, it performs well on complex and non-linearly seperable data
- It automatically maximize margin

However, comparing to other methods in this paper:

- The tuning is more complex, as we need to identify appropriate penalized parameter `cost`, kernel function paramters `gamma` and type of `kernel`
- SVM algorithm is complex, computational expensive which take time to obtain the tunning results.

For the average (miss)-classification rate in `vistestr`:

- **Tree**: 10% (in `vistestq` 20.5%)
- **Random Forest**: 10.5%
- **SVM**: 19.5%

(**Logit** on `vistestq`: 21.5%)

In this study, tree-based classification outperforms others.