TP4

Pierre Romon, Siyuan Li, Thu Uyen Nguyen

The aim of this practical work is to solve 2 distinct problems: a classification problem and a regression problem. To achieve this goal, we will select carefully the best models for both problems.

General functions

Principal Component Analysis (PCA)

Plotting cumulated explained variance is a good way to visually determine if performing a PCA would be relevant. Indeed, the goal of PCA is to keep a minimum number of components while keeping most of information. This is possible when few principal components support great proportion of information. Graphically speaking, the curve quickly goes up and then flattens. Otherwise, the curve tends to be linear which show a progressive increase of the cumulated explained variances.

```r
plot.pca <- function(data.X) {
  pc <- prcomp(x=data.X)$sdev
  plot(cumsum(pc), ylab='Cumulated explained variance', xlab='Principal component', type='o', col="#69b3
}
```

Cross-Validation (CV)

Studying the accuracy of model for a problem with a specific dataset can be tricky through any metrics when it comes to bias. Indeed, it can be difficult and sometimes not intuitive to eliminate bias in the score computation of the model. One of the most popular method to achieve this is the K fold cross validation. Cross validation permits the elimination of the bias resulting from the partition in a train and test subsets. In fact, the set is divided in K subsets and the algorithm proceed with K iteration. Each iteration, 1 subset is selected as the validation set, the K-1 other are gathered in a training set. On next iteration, it will be the turn of another subset to be the validation set. For each iteration, an intermediary score is computed and the final score is the average of the intermediary scores, thus, bias from partition is removed. Let's apply a cross validation through the caret library.

```r
cv <- function(data, compute_metric, method, K=10, seed=12) {
  set.seed(seed)
  control <- trainControl(method='cv', number=K)
  model <- train(y~., data=data, trControl=control, method=method)
  metric <- compute_metric(model)
  return(metric)
}
```

Nested Cross-Validation (NCV)

However cross validation is not sufficient for complex model that need hyper parameters to be tuned. Indeed, in the same way that scores are calculated without bias, cross validation is able to compute without bias best hyper parameters for a data set. But once these hyper parameters determined, score also need to be computed without bias for the model tuned with this best parameters. And here comes the nested cross validation. A nested cross validation is composed of 2 cross validation: a K outer one and a K' inner one. The inner one is in charge of the determination of the hyper parameters whereas the second one computes scores without bias. Caret is again very helpful here since it makes simple the inner loop and the sampling for the outer loop. Here we use a generic function for metric computation which will be passed as a parameter in order to generalize the implemented ncv to different kind of problem. The computed metrics are then averaged and so the bias is removed.

```r
ncv <- function(data, compute_metric, method, outer.K=10, inner.K=10, seed=12) {
  set.seed(seed)
  outer.folds <- createFolds(data$y, k=outer.K)
  metrics <- rep(0, outer.K)
  for(fold in 1:outer.K) {
    outer.idx <- outer.folds[[fold]]
    data.inner <- data[-outer.idx,]
    data.validation <- data[outer.idx,]
    inner.control  <- trainControl(method='cv', number=inner.K)
    if(method=='multinom' || method=='nnet') {
      inner.model <- train(y~., data=data.inner, trControl=inner.control, method=method, trace=FALSE)
    }
    else if(method=='leapBackward' || method=='leapForward' || method=='leapSeq') {
      inner.model <- train(y~., data=data.inner, trControl=inner.control, method=method, tuneGrid=data.
    }
    else {
      inner.model <- train(y~., data=data.inner, trControl=inner.control, method=method)
    }
    inner.grid <- inner.model$finalModel$tuneValue
    if(method=='multinom' || method=='nnet') {
      outer.model <- train(y~., data=data.inner, method=method, tuneGrid=inner.grid, trace=FALSE)
    }
    else {
      outer.model <- train(y~., data=data.inner, method=method, tuneGrid=inner.grid)
    }
    metrics[fold] <- compute_metric(outer.model, data.validation)
  }
  return(metrics)
}
```

Model selection

Once the score computed without bias, it is simple to choose the best model by comparing the scores.This function has a single goal: gathering the scores from the different considered models in an further aim of comparison. While doing that, we are also building boxplot that will allow a human and graphically choice of the best model.

```r
select_model <- function(data, compute_metric, method_list, outer.K=10, inner.K=10, seed=12) {
  df <- data.frame()[1:10, ]
  metrics <- c()
  for(method in method_list) {
    if(modelLookup(method)$parameter[1]=='parameter') {
      metrics <- c(metrics, cv(data, compute_metric, method, outer.K, seed))
    }
    else {
      scores <- ncv(data, compute_metric, method, outer.K, inner.K, seed)
      df[method] <- scores
      methods = c(methods, method)
      metrics <- c(metrics, mean(scores))
    }
  }
  long <- melt(df, id.vars=NULL)
  plot(value~variable, data=long, ylab='Metric', xlab='Methods')
  return(metrics)
}
```

Final training

Once the best model selected, it has to be trained again by cross validation on the whole dataset this time in order to compute its best hyper parameter. Finally, the model is ready to be used on test set.

```
final_training <- function(data, select_model_function, K=10, outer.K=10, inner.K=10, seed=12) {
  method <- select_model_function(data, outer.K, inner.K, seed)
  control <- trainControl(method='cv', number=K)
  model <- train(y~., data=data, trControl=control, method=method)
  return(model)
}
```

Classification Problem

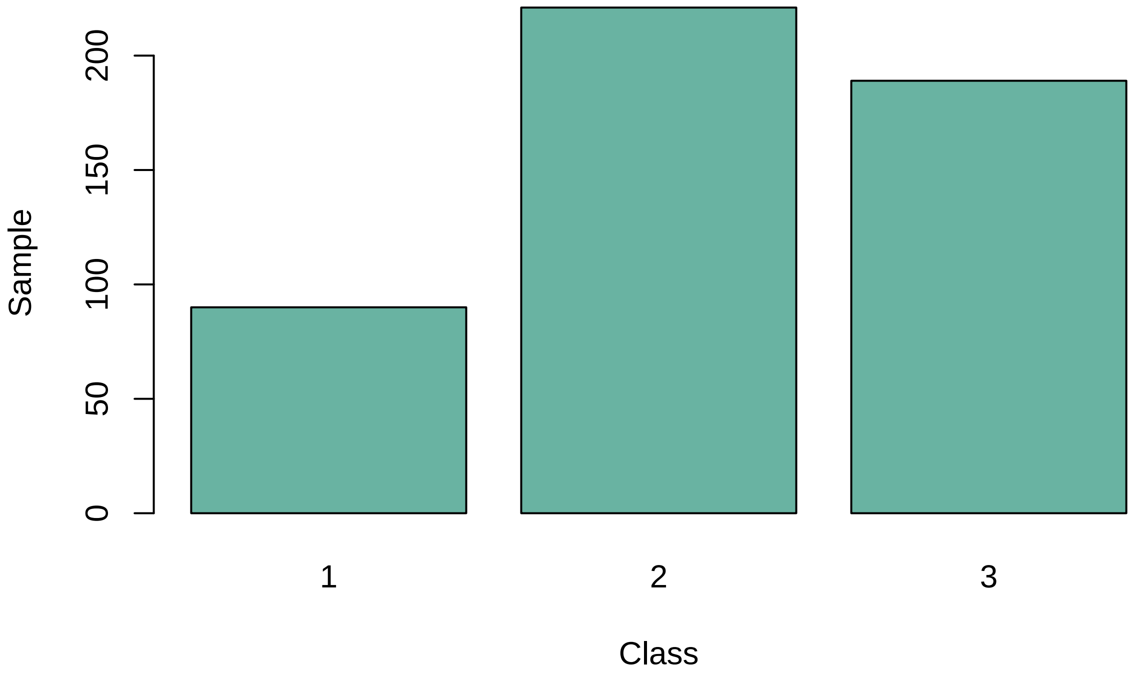
Dataset overview

First step is obviously to read the data and conduct an overview analysis about the shape of the data. Here, there are 500 samples with 50 features. This samples are split in 3 distinct classes. As samples are labeled, this classification problem belongs to the field of the supervised machine learning.

Metric

It is interesting to evaluate the distribution of the data between these 3 classes. Thanks to a *barplot*, the visualization of this distribution clearly shows an unbalanced distribution. 3 classes and an unbalanced distribution encourage the use of the weighted F-measure metric since it combines precision (positive predictive value) and recall (sensitivity) and take into account the class imbalance.

```
barplot.class()
```



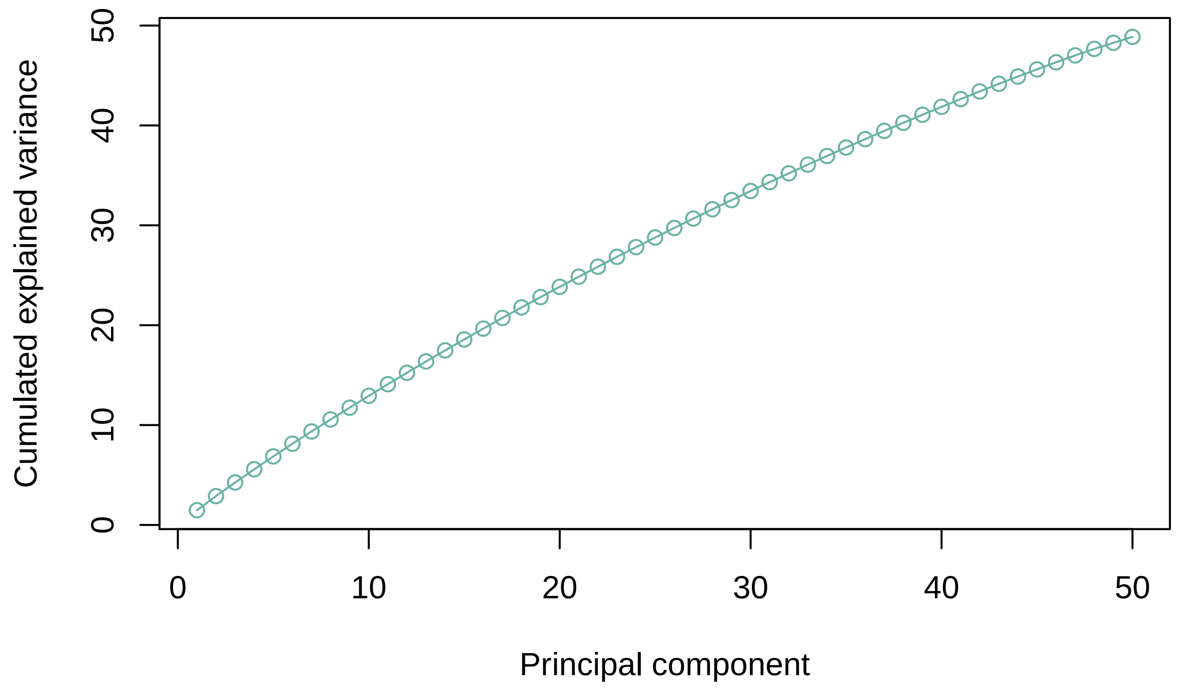
There exists implementation for F-score computation in other R module such as *caret* or *MLmetrics* but these implementations are not enough flexible and adaptive for our case. Indeed, we want here to be able to compute F-score after cross-validation and in the outer loop of nested cross-validation in a multi-class context. Thus, we choose to implement our own F-score computation function that uses confusion matrix.

We also implement a general function that compute f-score using previous function from a confusion matrix. This function is meant to be passed as a parameter in general cross validation and nested cross validation functions.

Preprocessing

The next step is to determine if we will use some preprocessing and, if yes, which preprocessing method we will use. The first to come in mind is PCA for feature selection and fortunately it is a preprocessing method whose efficiency is very easy to evaluate by plotting cumulated explained variance. Here, we notice that there is almost a linear increasing tendency. As said above, PCA is not relevant in this case.

```r
plot.pca(clf.data.X)
```



Cross Validation

Once the cross-validation performed, it is possible to exploit the confusion matrix to compute the F-score in this classification problem. Once again, we are using caret to easily extract the confusion matrix. We observe here good results from the discriminant analysis models (with quadratic better than linear discriminant analysis). That means that our data need a more flexible model like QDA but the high number of features makes us think that LDA might be also relevant. A trade-off between flexibility and robustness against high dimensions can be a solution. Thus, it may be interesting to explore further the world of discriminant analysis in the next steps.

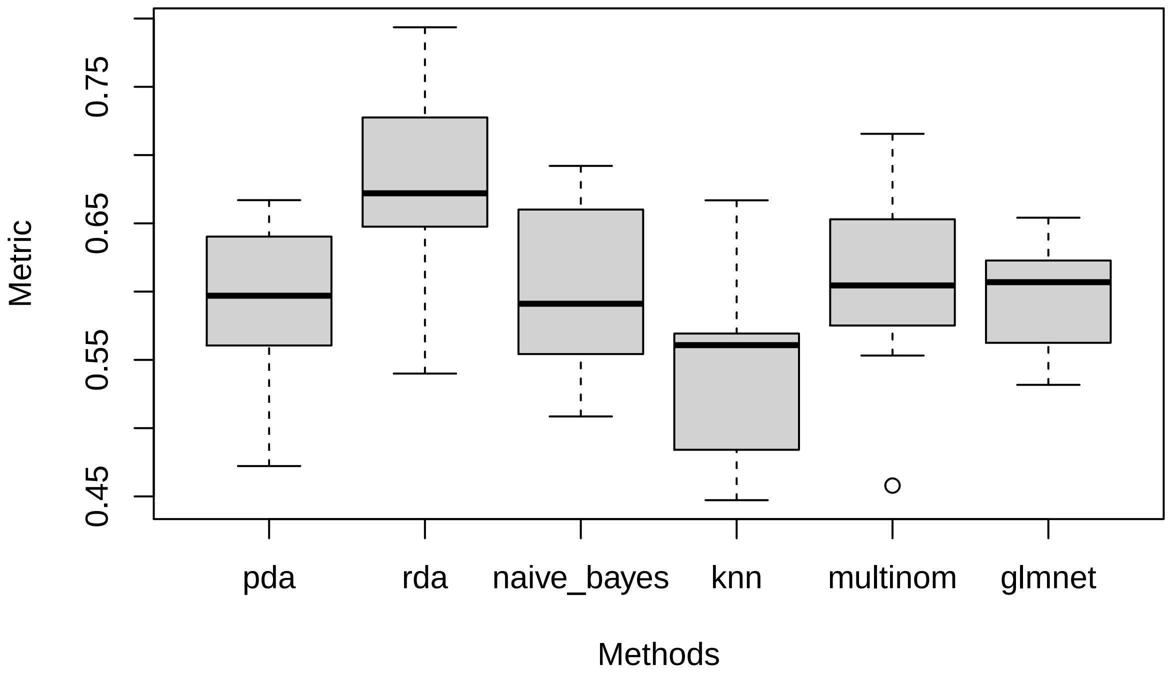
Nested Cross Validation

Here again, F-scores are computed from the confusion matrix resulting from the prediction of the tuned model. The template functions allow us to test a lot of different models. As stated above, we focus here on discriminant analysis models with the Penalized Discriminant Analysis that introduce shrinkage hyper parameter, the Regularized Discriminant Analysis and Naive Bayes. We also add other various models to widen our study.

Model selection

We are now able to compute decent F-score for every model we want to try. There is only model selection left. So that, we select model among the most popular classification models : Discriminant Analysis family, Naive Bayes, K Nearest Neighbors, Regularized Discriminant Analysis, Multinomial Regression, Generalized Linear Regression family. The best model is determined according to the maximized F-score. Thus, RDA is selected as best model. It was expected since RDA is said to be a good compromise between LDA and QDA. The result given by our model selection function is confirmed by the following box plot.

```r
clf.model <- clf.select_model()
```



Final model training

We then train RDA on the whole data set while computing the best parameters. The trained model is then stored in an environment file that will be used to load it for testing.

As expected from the relatively good results from linear and quadratic discriminant analysis, *rda*, which is a regularized version of the discriminant analysis, is very effective for classifying the samples of this problem.

Regression Problem

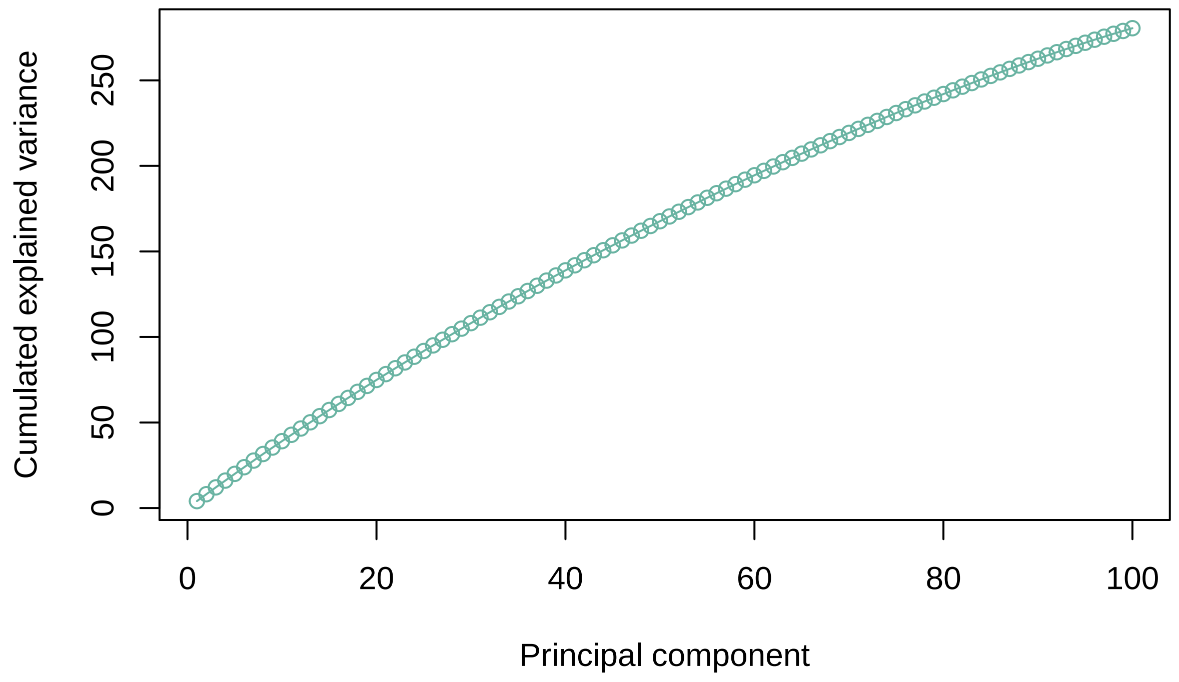
Dataset loading

Like the classification problem, we first want to overview the data. Here, there are 500 samples with 100 features. As solutions are given, this regression problem belongs to the field of the supervised machine learning.

Preprocessing

In the same way than the classification problem, we want to know if performing a PCA is relevant. Again, plotting the cumulated explained variances shows an almost linear curve which means that PCA is not relevant here.

```r
plot.pca(reg.data.X)
```



Metrics

The goal is to build a similar metric computation function in respect of the classification one. This metric computation function will be used for the cross-validation and the nested cross-validation. We chose here to use RMSE the classic metric for regression problems.

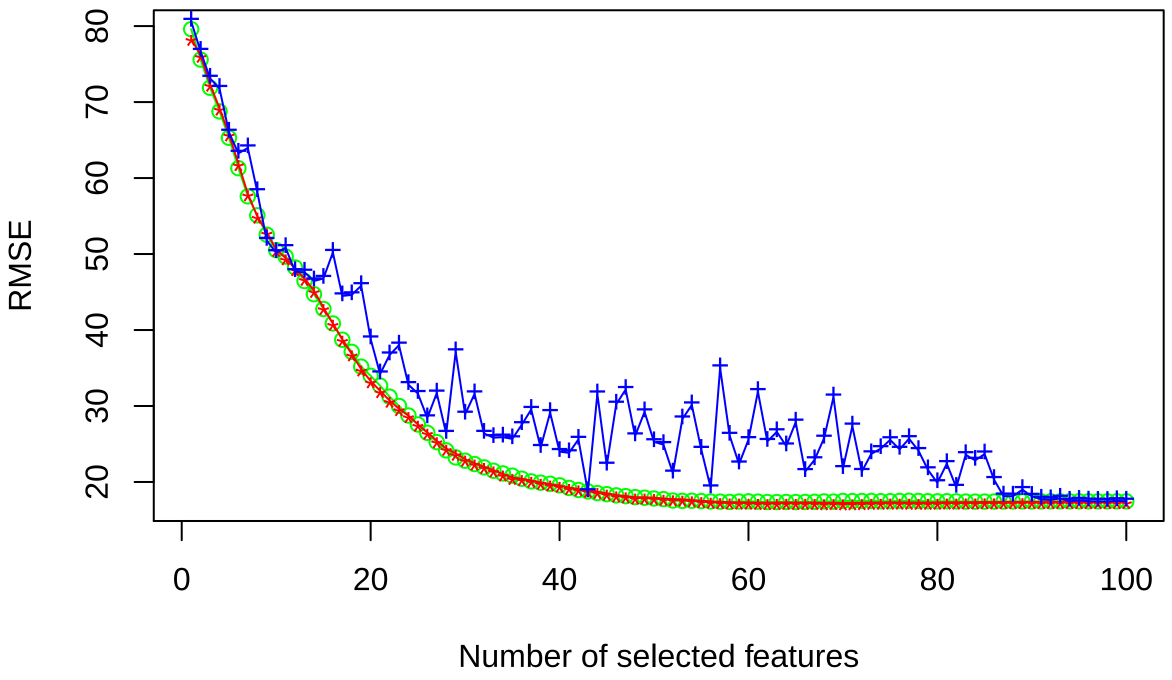
Cross-Validation

In the same way than classification problem, we are using cross validation to obtain non-biased score from our simple regression models. The first model that come in mind for a regression problem is the linear regression. From our results, we notice that linear regression is efficient for this problem with a RMSE. It can be then interesting to explore other linear regression based models.

Nested Cross-Validation

Thanks to the generalization work in advance, nested cross validation is here very simple since it is using the general template ncv function. As said before, we want to explore derivation of the linear regression. In fact, it is very easy since *glmnet* provides a generalization of the linear regression with models such as ridge, lasso or elastic net regression. Moreover, it is here important to considerate the case of selecting features models such as *leapBackward*, *leapFrontward* or *leapSeq* models. An hyper parameter, nvmax, describing the maximum number of selected features can be set. Let's see the evolution of the linear regression in function of the number of features selected through a graph (backward in green, forward in red and sequence in blue). We notice that linear regression is not interesting under 50 selected features. This information will be taken into account in the inner loop of the nested cross validation for the nvmax computation.

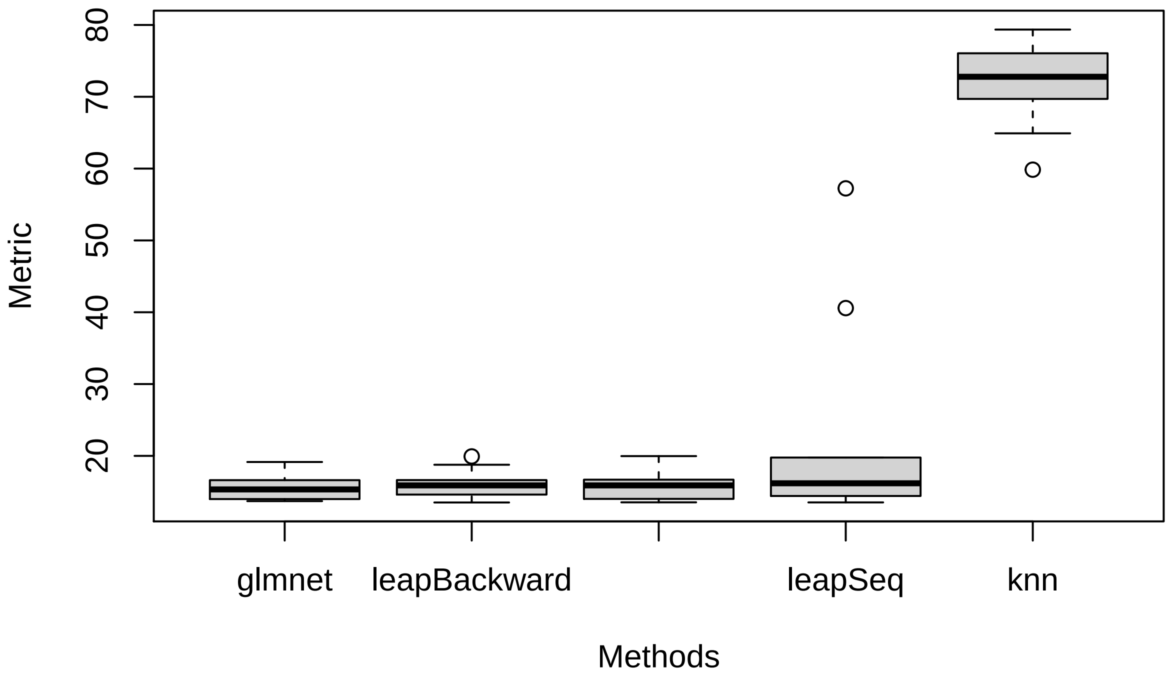
```r
plot.nvmax()
```



Model Selection

For the regression problem, we also selected models among the most popular one: linear regression family, linear regression with feature selection family and K Nearest Neighbors. K Nearest Neigbors' aim is to compare the efficiency of linear regression based model to other regression models. Through the template function, we obtain that the best model for this dataset is *glmnet*.

```
reg.model <- reg.select_model()
```



Final Training

Now that we choose the best model, we want to fit it to the whole data set and to compute best hyper parameter with cross-validation. We retrieve the best model already trained with best tune parameters on the whole data set.

Here, the *glmnet* seems to be the best model for this problem. However it is important to note that *glmnet* is a general linear regression model that gathers linear, ridge, lasso and elastic net regression. By inspecting alpha and lambda hyper parameter value, we know that we are in fact using an elastic net model (lambda and alpha hyper parameter are different from 0 and 1).

Model Storing

Best models selected for classification and regression problem are stored in an environment file so that they can be loaded in another R file which will be used for testing.

```r
save(reg.model, clf.model, file="env.RData")
```