TP4 - SY19

Rapport du TP4: Régression et classification - Sélection de modèles

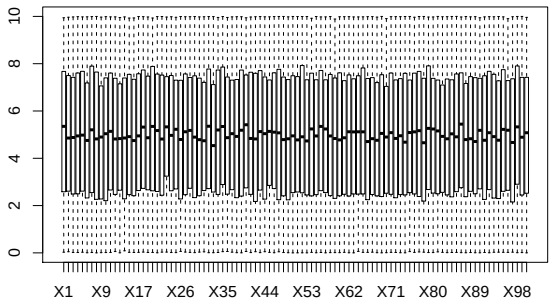
1 Regression data set

1.1 data import, exploration and partitioning

```
library(MASS)
library(pls)
library(leaps)
library(dplyr)
library(FNN)
library(glmnet)
```

```
reg.set <- read.table('data/TPN1_a22_reg_app.txt', header = TRUE)

# exploration
boxplot(as.data.frame(reg.set[1:100]))
```



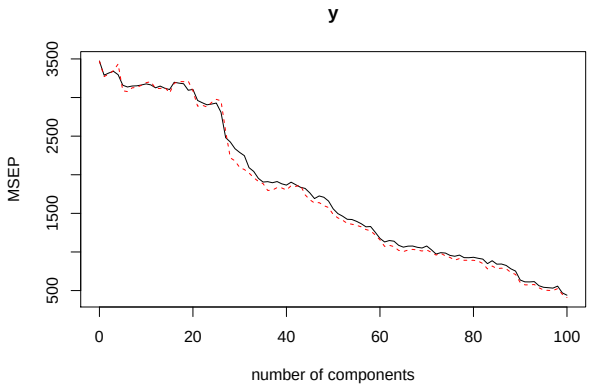
We can see that the range of all the variables are contained between 0 and 10.

```
train.percentage <- 2/3 #train partitioning
n_reg <- nrow(reg.set)
n_train <- as.integer(n_reg * train.percentage)
n_test <- n_reg - n_train
set.seed(69)
id_train <- sample(n_reg, n_train)
data.train <- reg.set[id_train,]
data.test <- reg.set[-id_train,]
y.test <- reg.set[-id_train, c(101)]
y.train <- reg.set[id_train, c(101)]
```

1.2 Model selection

1.2.1 PCA

```
library(pls)
pcr_model <- pcr(y~., data = data.test, validation = "CV")
validationplot(pcr_model, val.type="MSEP")
```



Looking at the graph, we can conclude that all the variables are needed to have an optimal performance, so this method is rejected.

1.2.2 Linear model

```
reg.set.lm <- lm(formula = y ~., data = data.train)
```

```
summary(reg.set.lm) # We have  small p-value and a relatively big R-Square
```

From the summary, the general linear model has already a very little p-value and an optimal R^2: 0.9626.

```
pre.lm <- predict(reg.set.lm, newdata = data.test)
mse.lm <- mean((pre.lm - y.test) ^ 2)
```

We got a mean squared error (MSE) of 200.176. Also, plotting the standard residuals we see that there is no apparent correlation.

We did the regression again with the significant variables:

```
reg.set.lm.revise <- lm(formula = y~X6+X11+X12+X15+X17+X22+X23+X25+X27+X32+X33+X35+X37+X39+X46+X47+X48+
pre.lm.revise <- predict(reg.set.lm.revise, newdata = data.test)
mse.lm1 <- mean((pre.lm.revise - y.test) ^ 2)
```

We got a R^2 of 0.9429 and a MSE of 246.1544, which is a bit worse, as predicted with the PCA analysis (it is better to keep all coefficient).

1.3 Subset selection

1.4 KNN using the significant variables

In this part, we try the KNN method with the previous variables.

1.4.1 Data with/without scaling

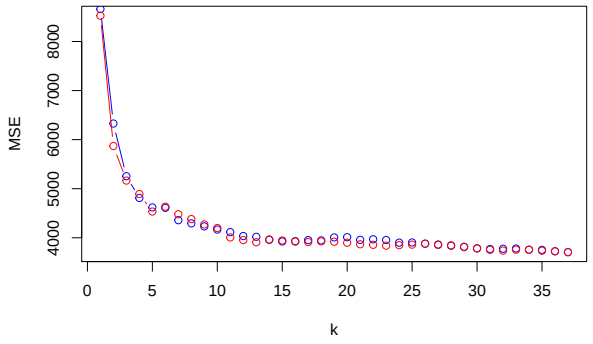
```
knn_k_max <- 37

tmp <- sapply(1:knn_k_max, function(local_k){
  reg.knn.noscale <- knn.reg(train = x.app_k, test = x.test_k, y = y.app_k, k = local_k)
  res <- mean((reg.knn.noscale$pred - y.test) ^ 2)
  return(res)
})

tmp.scale <- sapply(1:knn_k_max, function(local_k){
  reg.knn.scale <- knn.reg(train = x.app_k.scale, test = x.test_k.scale, y = y.app_k, k = local_k)
  res.scale <- mean((reg.knn.scale$pred - y.test) ^ 2)
  return(res.scale)
})

knn_MSE_noscale <- tmp
knn_MSE_scale <- tmp.scale
#erreur global
plot(1:knn_k_max, knn_MSE_noscale,
     type='b', col='blue',
     ylim=range(knn_MSE_scale),
     xlab='k', ylab='MSE', lty = 1, pch = 1)

lines(1:knn_k_max, knn_MSE_scale, type='b', col='red', lty = 1, pch = 1)
```



which.min.knn_MSE_scale.	min.knn_MSE_scale.	which.min.knn_MSE_noscale.	min.knn_MSE_noscale.
37	3707.279	37	3707.279

The minimum MSE is 3707.

1.5 Subset selection

Due to the fact that there is too many variables, we can't use the best subset selection.

1.5.1 Forward selection

To do this, we use regsubsets to find the significant variables with which we can reach a best adjr2.

```
reg.selection.forward <- regsubsets(y~., data = data.train, method = "forward", nbest = 1, nvmax = 100)
summary_forward <- summary(reg.selection.forward)
```

From summary_forward, we can get a list of potential models. To choose which is best suited, we compare the adjusted determination coefficient of each model. This value gives us the possibility to measure the quality of the linear regression.

```
rss<-data.frame(summary_forward$outmat, RSS=summary_forward$rss)
rsquare_max_forward <- summary_forward$outmat[which.max(summary_forward$adjr2),]
#La ligne avec la plus grande adjr2
rsquare_max_forward[rsquare_max_forward == '*'] <- as.numeric(1)
rsquare_max_forward[rsquare_max_forward == ' '] <- as.numeric(0)
rsquare_max_forward <- as.numeric(rsquare_max_forward)
#Le masque pour sélectionner les variables
reg.subset.forward <- reg.set[c(rsquare_max_forward==1)]
```

rsquare_max_forward is therefore a vector containing integers between 0 and 1 (1 for the columns to keep in the model and 0 for the columns to discard). We filter the data with it.

```
n.subset.forward <- nrow(reg.subset.forward)
set.seed(69)
n.subset.forward.train <- as.integer(train.percentage * n.subset.forward)
n.subset.forward.sample <- sample(n.subset.forward, n.subset.forward.train)
reg.subset.forward.train <- reg.subset.forward[n.subset.forward.sample,]
reg.subset.forward.test <- reg.subset.forward[-n.subset.forward.sample,]
```

```
reg.subset.forward.lm <- lm(formula = y~., data = reg.subset.forward.train)
reg.subset.forward.lm.predict <- predict(reg.subset.forward.lm, newdata = reg.subset.forward.test)
reg.subset.forward.mse <- mean((reg.subset.forward.lm.predict - reg.subset.forward.test$y) ^ 2)#188.44
```

The MSE is 188.44.

1.5.2 Backward selection

We use the same method previously mentioned to find the variables.

```
n.subset.backward <- nrow(reg.subset.backward)
set.seed(69)
n.subset.backward.train <- as.integer(train.percentage * n.subset.backward)
n.subset.backward.sample <- sample(n.subset.backward, n.subset.backward.train)
reg.subset.backward.train <- reg.subset.backward[n.subset.backward.sample,]
reg.subset.backward.test <- reg.subset.backward[-n.subset.backward.sample,]
reg.subset.backward.lm <- lm(formula = y~., data = reg.subset.backward.train)
reg.subset.backward.lm.predict <- predict(reg.subset.backward.lm, newdata = reg.subset.backward.test)
reg.subset.backward.mse <- mean((reg.subset.backward.lm.predict - reg.subset.backward.test$y) ^ 2)#186.
```

The MSE is 186.92.

1.5.3 K-Cross validation de Backward selection

To validate the model we have chosen, we have a K-Fold cross-validation. We will compare the different values obtained using the method with other potential models.

```
K <- 10
fold <- sample(K, n_train, replace = TRUE)
CV <- rep(0, 10)
for (i in (1:10)){
  for (k in (1:K)){
    reg.cross<-lm(Formula[[i]],data=reg.set[fold!=k,])
    pred.cross <- predict(reg.cross, newdata=reg.set[fold == k,])
    CV[i]<-CV[i]+ sum((reg.set$y[fold==k]-pred.cross)^2)
  }
  CV[i]<-CV[i] / n_reg
}
CV.min = min(CV)#181
```

We obtain the lowest value for the model we had chosen (181).

1.6 Ridge Regression

```
cv.out.ridge <- cv.glmnet(data.train.regu, y.train.regu, alpha = 0)
#plot(cv.out.ridge)
fit.ridge <- glmnet(data.train.regu, y.train.regu, lambda = cv.out.ridge$lambda.min, alpha = 0)
ridge.predict <- predict(fit.ridge, s = cv.out.ridge$lambda.min, newx = data.test.regu)
mse.ridge <- mean((ridge.predict - y.test.regu) ^ 2)#200.44
```

With this method, we obtain an MSE of 200.44.

1.7 Lasso regression

```
cv.out.lasso <- cv.glmnet(data.train.regu, y.train.regu, alpha = 1)
#plot(cv.out.lasso)
fit.lasso <- glmnet(data.train.regu, y.train.regu, lambda = cv.out.lasso$lambda.min, alpha = 1)
lasso.predict <- predict(fit.lasso, s = cv.out.lasso$lambda.min, newx = data.test.regu)
mse.lasso <- mean((lasso.predict - y.test.regu) ^ 2)#178
```

With this method, we obtain an MSE of 178.13.

2 Conclusion

```
cbind(mse.lm, mse.lm1, mse.knn.scale, mse.knn.noscale, reg.subset.forward.mse, reg.subset.backward.mse,
```

```
##        mse.lm  mse.lm1 mse.knn.scale mse.knn.noscale reg.subset.forward.mse
## [1,] 200.1762 246.1544      3707.279        3707.279               188.4402
##      reg.subset.backward.mse   CV.min mse.ridge mse.lasso
## [1,]                186.9206 181.4936  200.4437  178.1343
```

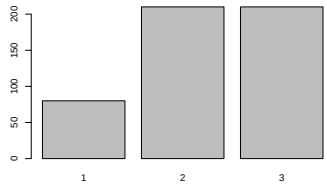
3 Classification data set

4 Classification

4.1 Preparation : Partitioning raw data to train & test

In order to separate the training and test data, we chose to randomly shuffle the data, and to take as many as four fifths of them for training. We also tested with two thirds of the training data, but the errors were slightly higher.

4.2 Data exploration




```
## [1] 0.58
```

We explore the data a with barplot can be seen: Y consists of three classes, the number of class1 is significantly smaller than the number of class2, 3. So if we do not do machine learning and choose the class with the largest proportion each time, our error rate will be 0.58, which will be the highest error rate we can accept

4.3 Nonparametric method: kNN

The first method we choose is the non-parametric one, the KNN method. Firstly we apply KNN with an arbitrary k = 10 to have a look at general result.

```
## [1] "Error total:"
```

```
## [1] 0.51
```

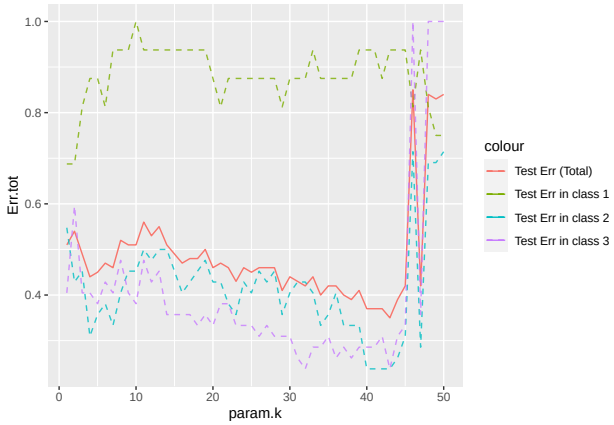
Error within each class:

```
##         1         2         3
## 1.0000000 0.4523810 0.3809524
```

The error rate reaches 0.51

4.3.1 KNN with an arbitrary k

Next we try to iterate over k to see if we can optimize the error rate



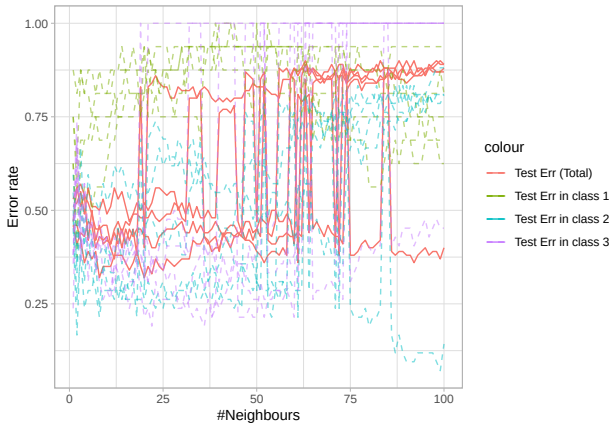

```
## [1] "Error minimal:"
```

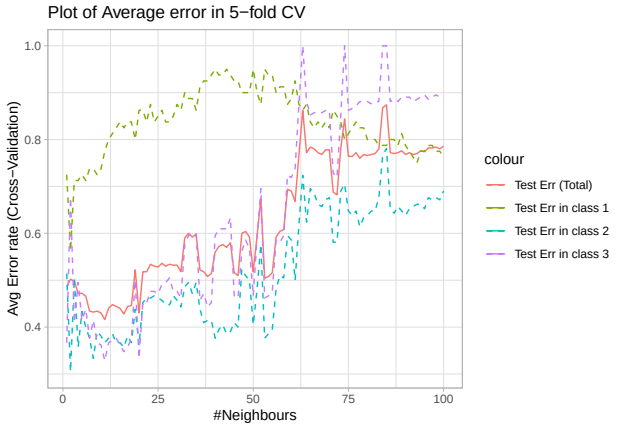
	Err.tot	Err.1	Err.2	Err.3	param.k
43	0.35	0.9375	0.2380952	0.2380952	43

Observing the plot, we see that class1 has an error rate of 1 when the value of k exceeds 10, most likely because class1 is a smaller class and is therefore divided into other classes when the value of k increases. But at k = 43 we observed a minimum error rate of 0.35, which is unlikily and next we applied cross comparisons to confirm the results. ### k_f-fold validation with k=5 In the k-fold validation, firtly we choose k = 5

```
## [1] "result"
## [1] "best parameter k : "
```

param.k	avg.Err.tot
11	0.416



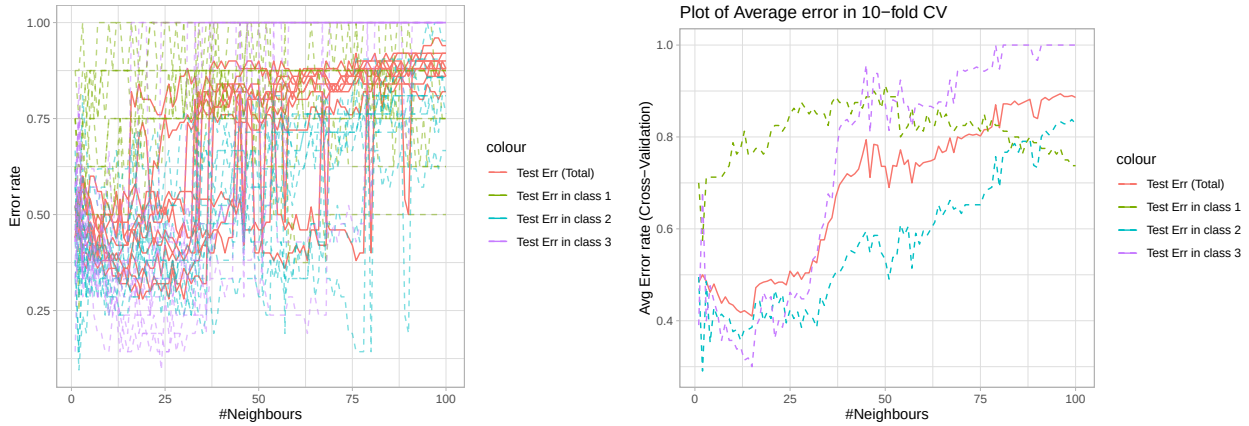


After k-fold =5, we obtained optimal results at KNN k=11 with an error rate of 0.41.

4.3.2 k_f-fold validation with k=10

```
## [1] "result"
## [1] "best parameter k : "
```

param.k	avg.Err.tot
15	0.41



After k-fold =10, we obtained optimal results at KNN k=15 with an error rate of 0.41, The results are similar

4.4 Model Selection for parametric methods

4.4.1 Forward and backward regression

In order to use QDA, LDA, naive bayes, and logistic regression we can try to see if model selection can be useful for us.

Using linear regression, backward selection and forward selection, along with lasso regression, we found those formulas:

```
formula <- c(y~.)
formula <- append(formula, y~X26+X44+X47+X40+X24+X19+X16+X5) # with selection of significant parameters
formula <- append(formula, y ~ X1 + X3 + X5 + X6 + X16 + X17 + X19 + X22 + X23 + X24 + X26 + X28 + X30 +
formula <- append(formula, y ~ X1 + X3 + X5 + X6 + X16 + X19 + X24 + X26 + X28 + X40 + X44 + X47) # wit
```

However:

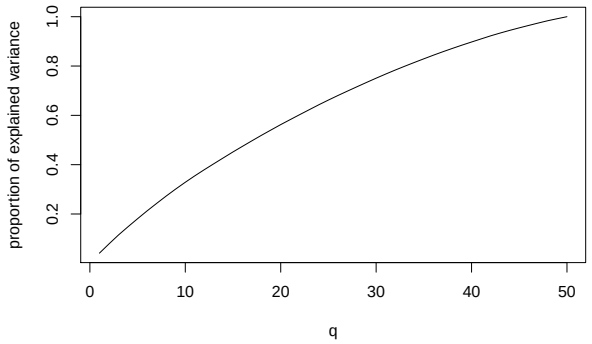
- The linear model had a adjusted R^2 of 0.04436, so it can't be reliable.
- The best predictor of the backward/forward selection had a adjusted R^2 of 0.099, that is to say that the model is unreliable.
- For the lasso regression, along for the two other formulas, we always found worst error rates with all the classifications methods used.

That is to say that, although we tried classification with all the four models, the three with coefficients selection couldn't help us.

4.4.2 Principal component analysis

We tried to use PCA to see if it is also relevant to reduce dimensionnality in our model.

```
X <- clas.set[1:50]
X<-scale(X) # Scale to improve PCA
pca<-princomp(X)
Z<-pca$scores
lambda<-pca$sdev^2
plot(cumsum(lambda)/sum(lambda),type="l",xlab="q",ylab="proportion of explained variance")
```



We generally find no relevant analysis on PCA, as the curve really plummet onto 0 explained variance, so it is difficult to find a trade-off to have a good explained variance and a small number of dimensions.

4.5 QDA

```
clas.set$y <- as.factor(clas.set$y)

fit.qda  <- qda(y ~ ., data=data.train)
pred.qda <- predict(fit.qda, newdata=data.test)
perf.qda <- table(data.test$y, pred.qda$class)
sum (diag(perf.qda)) / n_test
err.qda <- 1-sum (diag(perf.qda)) / n_test
```

After one QDA, we obtained an error rate of 0.38, which is already better than the optimal KNN result.

We used the K-Fold Cross Validation to have a more precise result on the robustness of the QDA model.

4.6 QDA with K-fold validation

```
K<-10
err = rep(0,4)
par(mfrow = c(2, 2))
for (i in (1:4)){
  CV <- rep(0,10)
  folds=sample(1:K,n_clas,replace=TRUE)
```

```
  for(k in (1:K)){
    class<-qda(formula[[i]],data=clas.set[folds!=k,])
    pred<-predict(class,newdata=clas.set[folds==k,])
    conf <- table(clas.set[folds==k,]$y, pred$class)
    CV[k]<-1-sum(diag(conf))/nrow(clas.set[folds==k,])
  }
  CV_mean<-mean(CV)
  plot(CV, type="l")
  err[i] <- CV_mean
}
err.qda = min(err) # We find the complete model (y~.) gives the best error rate
```

We found a global error rate of **0.3205** for the QDA, thanks to the 10-Folds cross validation.

4.7 LDA

We show directly how we use the LDA methods with K-Folds Cross validation to evaluate the error rate of the

4.8 LDA with K-fold validation

```
K<-10
err = rep(0,4)
par(mfrow = c(2, 2))

for (i in (1:4)){
  CV <- rep(0,10)
  folds=sample(1:K,n_clas,replace=TRUE)
  for(k in (1:K)){
    class<-lda(formula[[i]],data=clas.set[folds!=k,])
    pred<-predict(class,newdata=clas.set[folds==k,])
    conf <- table(clas.set[folds==k,]$y, pred$class)
    CV[k]<-1-sum(diag(conf))/nrow(clas.set[folds==k,])
  }
  CV_mean<-mean(CV)
  err[i] <- CV_mean
}
err.lda = min(err) # We find the complete model (y~.) gives the best error rate
```

We found an mean error rate across all 10 folds of **0.4148** using LDA, on one realization, that is bigger error rate than the one of QDA method.

4.9 Naive Bayes

We then tried to use the QDA naive bayes method to see if it predicts better than the others.

```
clas.set$y <- factor(clas.set$y)

K<-10
```

```
err = rep(0,4)
par(mfrow = c(2, 2))
for (i in (1:4)){
  CV <- rep(0,10)
  folds=sample(1:K,n_clas,replace=TRUE)
  for(k in (1:K)){
    class<-naive_bayes(formula[[i]],data=clas.set[folds!=k,])
    pred<-predict(class,newdata=clas.set[folds==k,][1:50])
    conf <- table(clas.set[folds==k,]$y, pred)
    CV[k]<-1-sum(diag(conf))/nrow(clas.set[folds==k,])
  }
  CV_mean<-mean(CV)
  err[i] <- CV_mean
}
err.naiveqda = min(err)
```

This time we found, for one realization of 10-Folds cross validation for Naive Bayes method on our data set, of 0.3509.

4.10 Multinomial logistic regression

Here, our data have the classes c > 2, so we used the "Multinomial logistic regression" method.

```
K<-10
err = rep(0,4)
par(mfrow = c(2, 2))

for (i in (1:4)){
  CV <- rep(0,10)
  folds=sample(1:K,n_clas,replace=TRUE)
  for(k in (1:K)){
    class<-multinom(formula[[i]],data=clas.set[folds!=k,])
    pred<-predict(class,newdata=clas.set[folds==k,])
    conf <- table(clas.set[folds==k,]$y, pred)
    CV[k]<-1-sum(diag(conf))/nrow(clas.set[folds==k,])
    #  class<-qda(y~.,data=clas.set[folds==k,])
    #  pred<-predict(class,newdata=clas.set[folds==k,])
    #  conf <- table(clas.set[folds==k,]$y, pred$class)
    #  CV2<-CV2+1-sum(diag(conf))/nrow(clas.set[folds==k,])
  }
  CV_mean<-mean(CV)
  err[i] <- CV_mean
}
err.naiveqda = min(err)
```

This time, error rates for one realization of the K-Folds CV with the multinomial logistic regression is of **0.4212**.

4.11 ROC Curves

The area under the curve (AUC) of QDA ROC Curve is really low (close to 0.5). It means that in this case, QDA cannot be reliable. The best that we find is Naive Bayes and that is also the case for the error rate,

so Naive Bayes method is best to choose (we couldn't plot logistic regression with pROC library, but we already now it has a high average error rate).

4.12 Summary for classification

As we have the lowest error rate (about 0.33) and and good ROC curve for the Naive Bayes method, we choose this method.

4.13