Task 1

Contents

1 Task 1

1.1 Introduction

In the present part of the report, we will investigate to what extent we will be able to classify respondents in their country, and then we will compare the performance of different classifiers.

1.2 Data

The data have been obtained from the 6^{th} Wave of the World Value Survey, which was carried out between 2010 and 2013. The data include the standardized scores of 3929 respondents of 3 countries on 32 variables, that have been summarized with 7 factors obtained using exploratory factor analysis with oblique rotation. The 7 factors related to the 32 variables are:

1. **Rights**, that it's related to homosexuality, prostitution, abortion, divorce, sex before marriage, suicide;
2. **Steal**, that it's related to claiming benefits, avoiding fare, stealing property, cheating taxes, accept a bribe;
3. **Crime**, that it's related to robberies, alcohol, police-military, racist behavior, drug sale;
4. **Religion**, that it's related to attend religious services, pray, the importance of God;
5. **Realize self**, that it's related to creative, rich, spoil oneself, be successful, exciting life;
6. **Do good**, that it's related to security, do good, behave properly, protect environment, tradition;
7. **Violence**, that it's related to beat wife, parents beating children, violenc.

1.3 Methodology

To investigate the possibility to classify the respondents in their country based on the 7 factors we have used the canonical discriminant analysis. We have applied the linear regression function with 7 predictors and 1 dependent variable, the Country. Then to the output, we have applied the Canonical Discriminant Analysis.

```
lm.out<-lm(cbind(F_rights, F_steal, F_crime,F_religion,F_realizeself,F_dogood,
                 F_violence)~as.factor(country), data=dwvs)
candisc.out<-candisc(lm.out)
print(candisc.out)
```

```
Canonical Discriminant Analysis for as.factor(country):

   CanRsq Eigenvalue Difference Percent Cumulative
1 0.80691    4.17882     3.5622  87.142     87.142
2 0.38142    0.61661     3.5622  12.858    100.000

Test of H0: The canonical correlations in the
current row and all that follow are zero

  LR test stat approx F numDF denDF   Pr(> F)
1      0.11944  1059.53    14  7834 < 2.2e-16 ***
2      0.61858   402.65     6  3918 < 2.2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

As we can see both the Square Canonical Correlation are significant, but the discriminating power to separate between the groups is higher for the first than for the second discriminant function: 0.81 and 0.38 respectively. The LR test indicates that the discriminant analysis is meaningful. The first test's null hypothesis is $H_0 : \lambda_1 = \lambda_2 = 0$ and this hypothesis as we can see from the *p-value* is rejected. The *null* hypothesis of the first test it's equivalent to the test for $H_0 : \mu Netherlands = \mu Nigeria = \mu Philippines$.

The second LR test indicates that $H_0 : \lambda_2 = 0$, and also this null hypothesis is rejected. So even if the second discriminant function has less discriminant power cannot be omitted and it's statistically meaningful.

On our analysis, we have also applied two different tests for centroids and to test the equal covariance.

To see if the three-country has different centroids and confirm the results of the canonical discriminant analysis we have applied on the linear regression the function *Manova*:

```
res_t1_2 <- summary(Manova(lm.out), test="Wilks")

summary.default(Manova(lm.out), test="Wilks")
```

```
          Length Class  Mode
SSP        1     -none- list
SSPE      49     -none- numeric
df         1     -none- numeric
error.df   1     -none- numeric
terms      1     -none- character
repeated   1     -none- logical
type       1     -none- character
test       1     -none- character
```

The *p-value* is small, and the test confirms that the analysis is meaningful and that at least there is a pair of centroids that differs significantly. The function *Manova* in r doing the *Wilks Lambda test* uses the Rao approximation. To test the assumption on equal population covariance we have applied to the linear regression the function *boxM*:

```
boxM(lm.out)
```

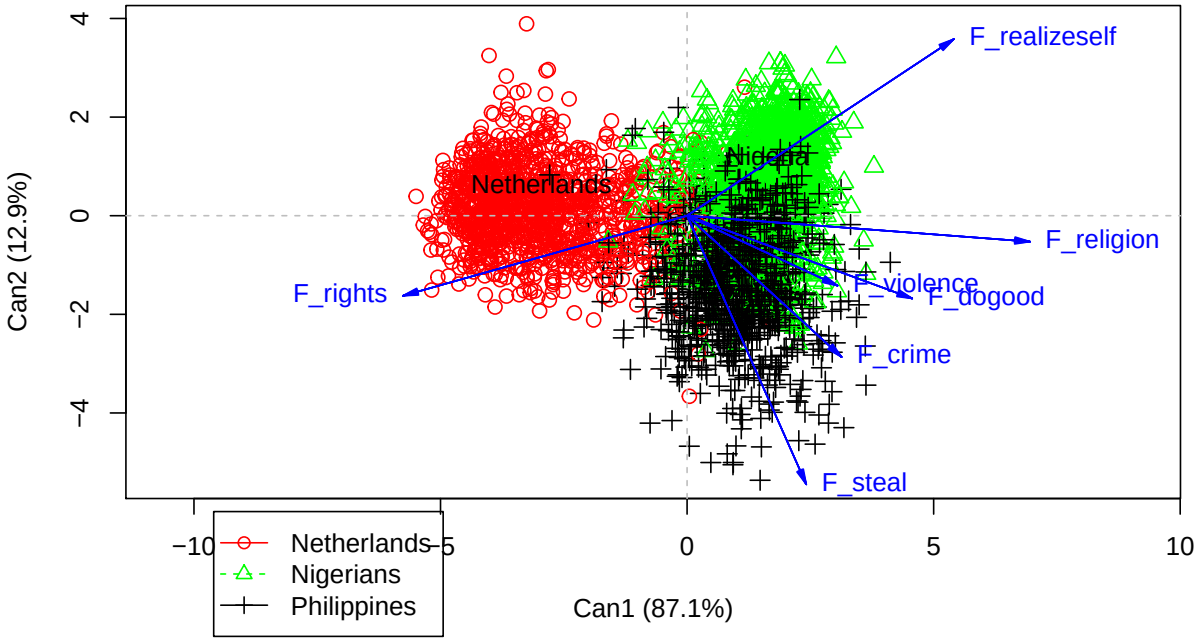
```
    Box's M-test for Homogeneity of Covariance Matrices

data:  Y
Chi-Sq (approx.) = 5479.2, df = 56, p-value < 2.2e-16
```

The test of Box indicates that H_0 of equal covariance matrices across groups is not supported by data.

1.3.1 Plot

To complete the Canonical Discriminant analysis, we have plotted the three countries and the 7 variables.

```
Vector scale factor set to 7.827
```



We can see that the group of individuals in *red* are Netherlands citizens, the group of individuals in *green* are Nigerians citizens and the group of individuals in *black* are Philippines citizens. In blue we can see the 7 explanatory variables. The plot shows a clear separation between Netherlands and the other two countries on the first discriminant function while the second discriminant function could help to separate Nigeria and the Philippines. The first discriminant function especially correlates with the factors: rights, religion, realize self, and do good; whereas the second discriminant function correlates with the factors: steal and realize self. The two-factor crime and violence have a lower correlation on the two factors and so has in this analysis lower importance on separate the 3 countries.

1.3.2 Compare the performance of different classifiers

We are going now to compare the performance of different classifiers to classify respondents in their country based on the 7 factors. To be able to do that we are going to compute the training error and the leave-one-out cross-validation error. The classification method that we are going to compare are: - Linear discriminant analysis; - Quadratic discriminant analysis; - K-nearest neighbors with k ranging from *1 to 100*; - High Dimensional Discriminant Analysis;

1.3.3 Linear discriminant analysis

This method aims to separate in the clearest possible way different groups using the linear combination of observed independent variables. The linear discriminant analysis method assumes that the covariance structure of the independent variable is the same across groups. In our analysis, we know from the Box test previously computed that this assumption is not supported by the data. It will be an interesting test if in this case the Quadratic discriminant analysis, where the assumption on the equality of covariance matrix is relaxed, will perform better. In the linear discriminant analysis, we have applied the method of Fisher correcting for the different prior probability.

```
lda.out1<-lda(country ~ F_rights + F_steal + F_crime + F_religion + F_realizeself +
                F_dogood + F_violence, data=dwvs)
#print(lda.out1)
pred.train1 <- predict(lda.out1,dwvs, prior=c(1,1,1)/3)
```

```
tab1 <- table(dwvs$country,pred.train1$class)
#print(tab1)
kbl(tab1)
```

	Netherlands	Nigeria	Philippines
Netherlands	1145	39	76
Nigeria	10	1327	241
Philippines	17	220	851

```
#training hit rate
kbl(sum(diag(tab1))/sum(tab1))
```

x
0.8464086

```
#classify test observations using LDA
pred.loocv2<-lda(country~F_rights+F_steal+F_crime+F_religion+F_realizeself+F_dogood+
                   F_violence,data=dwvs, prior=c(1,1,1)/3, CV=TRUE)
tab2<-table(dwvs$country,pred.loocv2$class)
print(tab2)
```

```
             Netherlands Nigeria Philippines
  Netherlands        1145      39          76
  Nigeria              11    1326         241
  Philippines          17     224         847
```

```
#LOOCV hit rate
kbl(sum(diag(tab2))/sum(tab2))
```

x
0.845135

We can see that in that case, the difference between the performance for training error and LOOCV error is really small, so there is no evidence for overfitting.

1.3.4 Quadratic discriminant analysis

The second method that we have applied is Quadratic discriminant analysis. It should perform better considering the difference in the covariance matrix for the different groups. QDA even if has a lower bias with a different covariance matrix, has a larger variance, and as in our case with a small dataset can be problematic.

Even in this case, we have applied the method of Fisher correcting for prior probabilities.

```
qda.out3<-qda(country ~ F_rights + F_steal + F_crime + F_religion + F_realizeself +
                F_dogood + F_violence, data = dwvs)

pred.train3<-predict(qda.out3,dwvs, prior=c(1,1,1)/3)

tab3<-table(dwvs$country,pred.train3$class)
#print(tab3)
kbl(tab3)
```

	Netherlands	Nigeria	Philippines
Netherlands	1210	21	29
Nigeria	40	1319	219
Philippines	41	219	828

```
#training hit rate
kbl(sum(diag(tab3))/sum(tab3))
```

x
0.8550688

```
#classify test observations using QDA
pred.test4 <- qda(country ~ F_rights + F_steal + F_crime + F_religion + F_realizeself +
                    F_dogood + F_violence, data=dwvs,prior=c(1,1,1)/3,CV=TRUE)
tab4<-table(dwvs$country,pred.test4$class)
#print(tab4)
kbl(tab4)
```

	Netherlands	Nigeria	Philippines
Netherlands	1210	21	29
Nigeria	40	1315	223
Philippines	43	223	822

```
#LOOCV hit rate
kbl(sum(diag(tab4))/sum(tab4))
```

x
0.8525217

In that case there is also no evidence of overfitting. We can see from the results that the QDA performs better than the LDA but not with a significant improvement.

1.3.5 K-nearest Neighbors

The third model that we had analyzed is the K-nearest Neighbors. We have computed the model using all the 3926 observations, and to choose which is the correct number k of parameters to use, we have compared the training error with the Leave one out cross-validation error.

```
#str(dwvs)
table(dwvs$country)
```

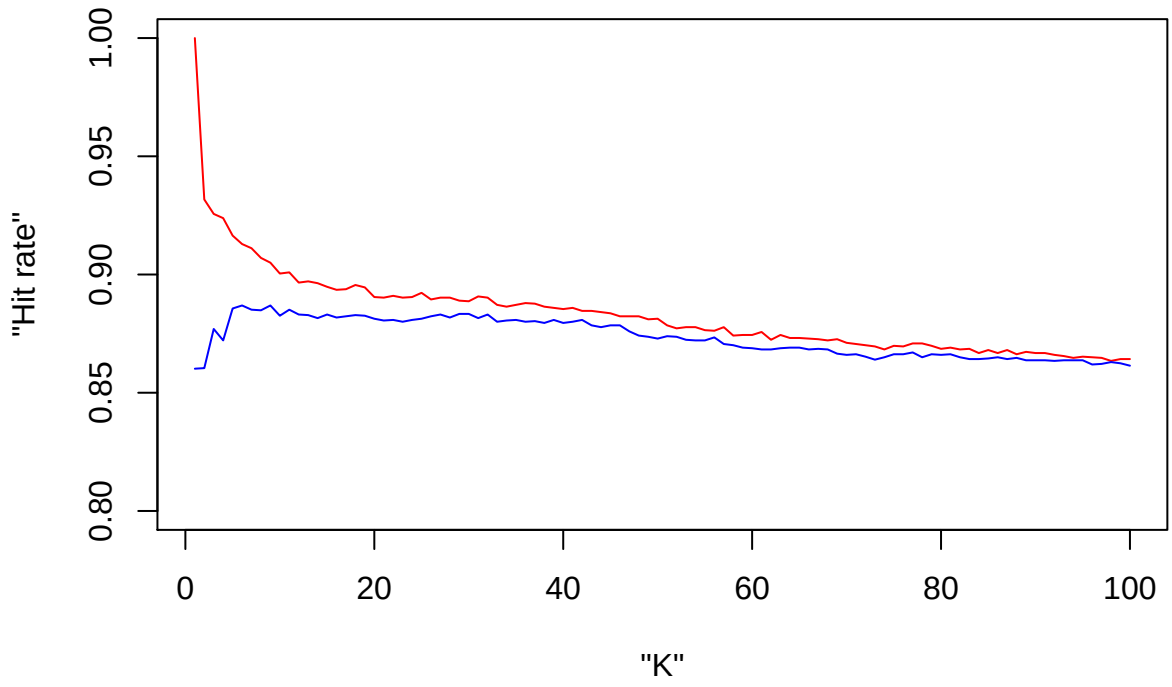
```
Netherlands     Nigeria Philippines
       1260        1578        1088
```

```
set.seed(9850)   # -> random number generator
gp<-runif(nrow(dwvs))
dwvs2<-dwvs[order(gp),]
#str(dwvs)
#str(dwvs2)
#head(dwvs)
#head(dwvs2)

hitratknn<-function(observed,predicted){
  tab<-table(observed,predicted)
  hitratknn<-sum(diag(tab))/sum(tab)
  return(hitratknn)
}

knnmax<-100
err<-matrix(rep(0,knnmax*2), nrow=knnmax)

for(j in 1:knnmax) {
  predknn.train<-knn(dwvs2[,2:8], dwvs2[,2:8], dwvs2$country, k=j)
  err[j,1]<-hitratknn(dwvs2$country,predknn.train)
}

for(j in 1:knnmax) {
  predknn.train<-knn.cv(dwvs2[,2:8], dwvs2$country, k=j)
```

```
  err [j,2]<-hitratknn(dwvs2$country,predknn.train)
}

plot('K', 'Hit rate',xlim=c(1,knnmax),ylim=c(0.8,1))
```

```
Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduced by coercion

Warning in xy.coords(x, y, xlabel, ylabel, log): NAs introduced by coercion
```

```
lines(c(1:knnmax),err[,1],col="red") # -> training error
lines(c(1:knnmax),err[,2],col="blue")
```



We can see that with K=1 the model is flexible and by definition, we have training hit rate (red line) of 0, but the LOOCV hit rate (blue line) is higher in this case, while with model less flexible, as with k=98, the two errors are similar. Since both the errors increase if we increase the parameter K, probably the model that describes the dataset better is the model with K=30 or K=66.

1.3.6 High Dimensional Discriminant Analysis

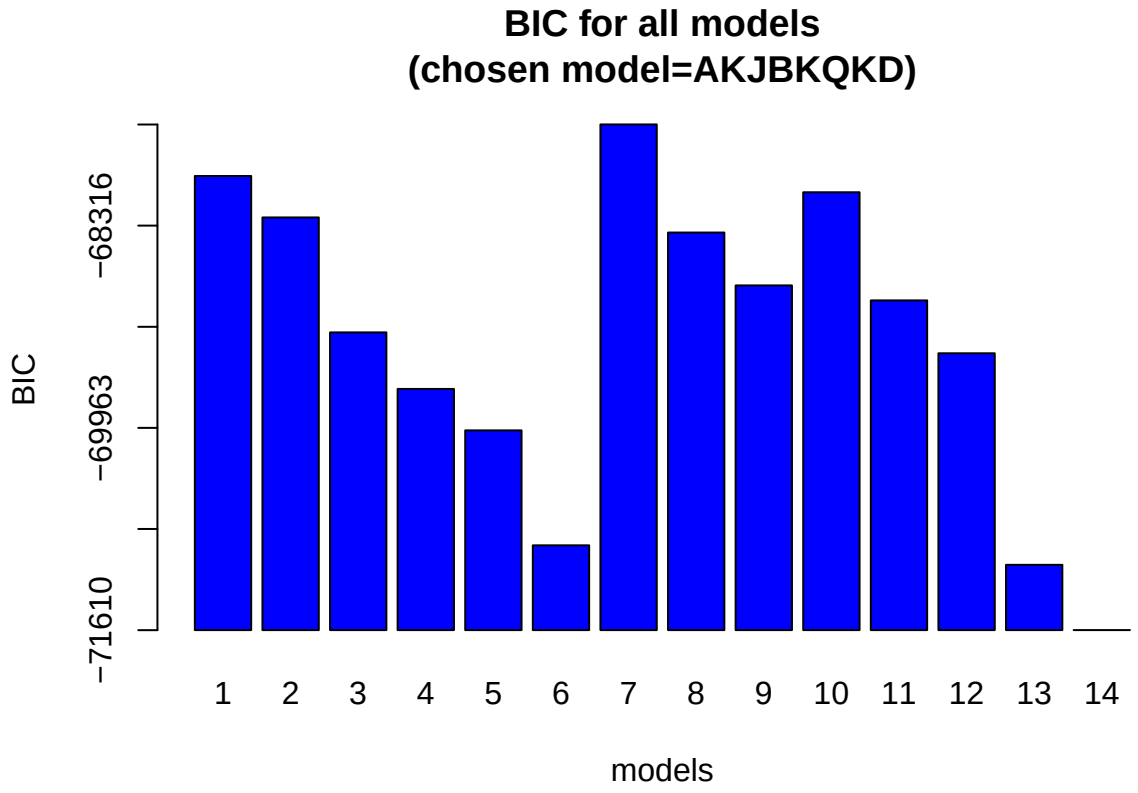
The fourth method that we have used to discriminate between different groups is the HDDA method. This method could be useful while the number of parameters is high compared to the number of data.

Table 1

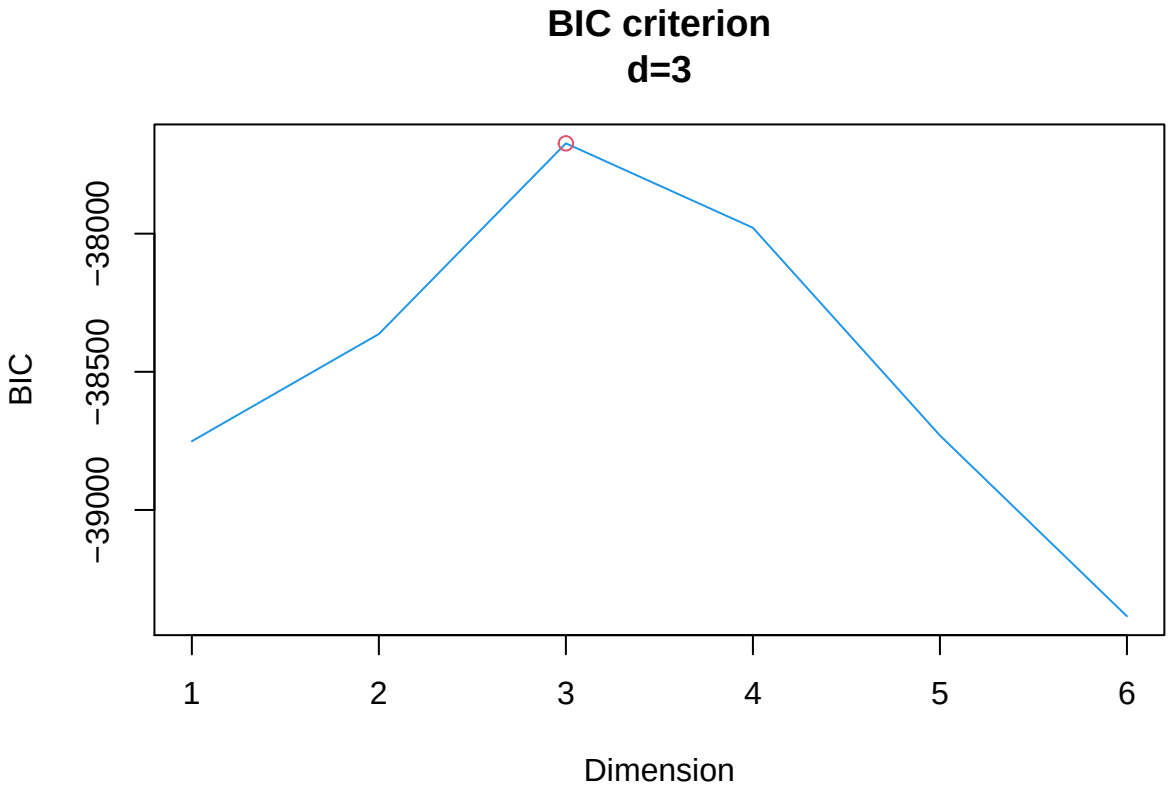
K	TRAINING HIT RATE	LOOCV HIT RATE
1	1	0.8601630
30	0.8886908	0.8833418
66	0.8728986	0.8683138
100	0.8642384	0.8614366

```
w <- dwvs[,-1]
cls <- dwvs[,1]
#HDDA on the learning dataset:
hdda.out7 <- hdda(w, cls, scaling=TRUE, model="all", d="BIC",graph=TRUE,show=TRUE)
```

```
 # :     Model     BIC
 1 :     AKJBKQKDK   -67912.32
 2 :     AKBKQKDK    -68249.76
 3 :     ABKQKDK     -69185.72
 4 :     AKJBQKDK    -69646.01
 5 :     AKBQKDK     -69983.46
 6 :     ABQKDK      -70919.41
 7 :     AKJBKQKD    -67492.38
 8 :     AKBKQKD     -68372.73
 9 :     ABKQKD      -68803.13
10 :     AKJBQKD     -68044.89
11 :     AKBQKD      -68925.24
12 :     ABQKD       -69355.63
13 :     AJBQD       -71078.02
14 :     ABQD        -71609.34

SELECTED: Model AKJBKQKD, BIC=-67492.38.
```

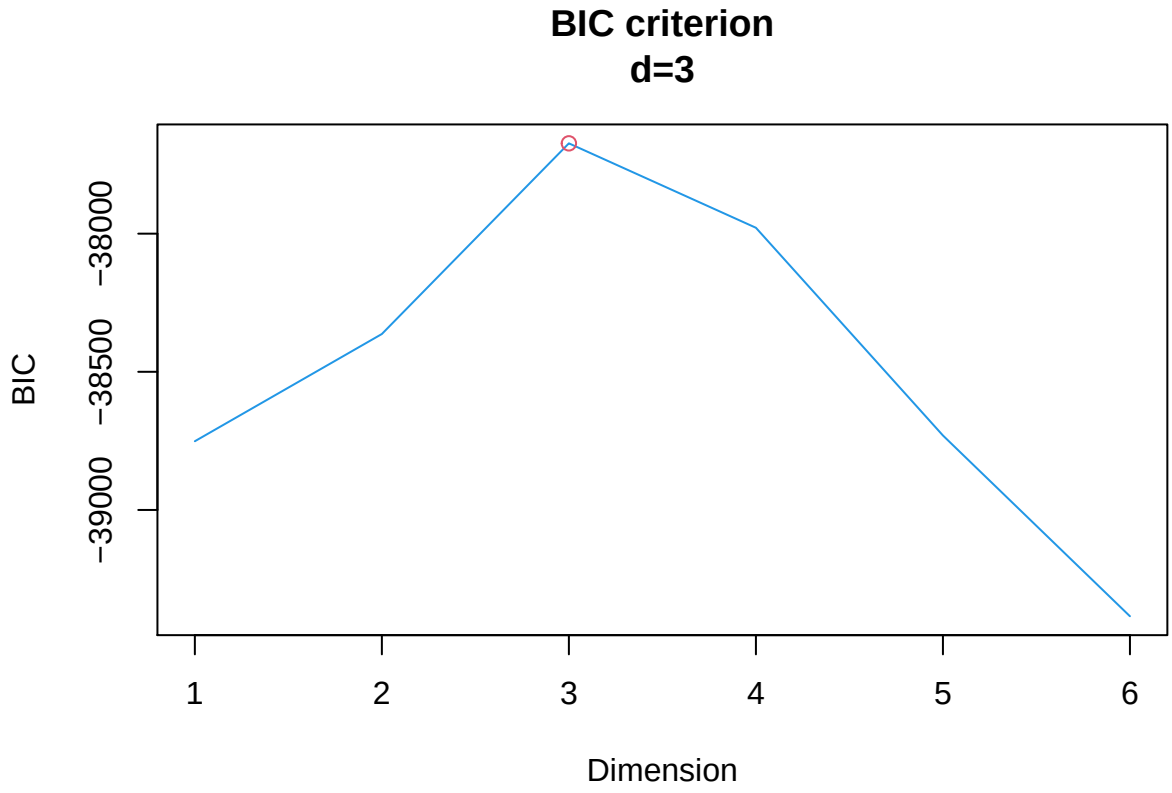



```
plot(hdda.out7)
```



The model used from the HDDA analysis applying the BIC criterion is model 7($A_{kj}K_jB_kQ_kd$). The decision following the *BIC* criteria is to choose the model with the lowest value, in that case, the reference system is negative, so the lowest value is for model 7.

```
plot(hdda.out7,method="BIC")
```



The dimension choose for the model is 3. The model chosen by the *BIC* criteria has the same number of principal components for all the three different classes.

```
pred.train7<-predict(hdda.out7,w,cls)
```

```
Correct classification rate: 0.8530311.
               Initial class
Predicted class Netherlands Nigeria Philippines
    Netherlands        1196      20          40
    Nigeria              26    1356         251
    Philippines          38     202         797
```

```
#print(tab7)
tab7<-table(dwvs$country,pred.train7$class)
#training hit rate
kbl(sum(diag(tab7))/sum(tab7))
```

x
0.8530311

```
pred.loocv8 <- hdda(w, cls,scaling=TRUE, d="BIC", LOO=TRUE)
tab8<-table(cls,pred.loocv8$class)
#print(tab8)
kbl(tab8)
```

	Netherlands	Nigeria	Philippines
Netherlands	1197	26	37
Nigeria	22	1378	178
Philippines	44	285	759

```
#LOOCV hit rate
kbl(sum(diag(tab8))/sum(tab8))
```

x
0.8492104

Also with the *HDDA* model, there is rather small evidence of overfitting, but the HDDA model does not perform better than the other models.

1.3.7 Error comparison for the different models

We have computed for all 4 models the hit rate, for the comparison in the table we will present the training and LOOCV errors computing $1 - *hit$ rate:

Table 2

MODEL	TRAINING ERROR	LOOCV ERROR
LDA	0.1535914	0.154865
QDA	0.1449312	0.1474783
KNN (K=30)	0.1113092	0.1166582
HDDA	0.1469689	0.1507896

In all the present models there is little evidence of overfitting, the two errors computed are in all the cases similar. The K-nearest Neighbors is a good model to compare the others and we can see that even if it performs betters it has not a huge difference. The model that performs better between the other 3 is the Quadratic discriminant analysis, it is the most complex one with the highest number of parameters used.

Confronting the results, we can say that even if there is a difference between the models, no one of the computed ones has outstanding results.

Table of *QDA LOOCV* rate:

As we were expecting in the analysis of the canonical discriminant analysis, the model has a high ability to differentiate between Netherlands and the two other countries, while it has a high error rate discriminating between Nigeria and the Philippines.

Table 3

-	Netherlands	Nigeria	Philippines
Netherlands	1210	21	29
Nigeria	40	1315	223
Philippines	43	223	822

1.3.8 Multinomial logistic regression model

```
m1<- multinom(country~F_rights+F_steal+F_crime+F_religion+F_realizeself+F_dogood +
                F_violence, family=multinomial, data=dwvs, maxit=3926, hess=TRUE)
```

```
# weights:  27 (16 variable)
initial  value 4313.151845
iter  10 value 1376.724330
iter  20 value 1352.584896
final  value 1346.617148
converged
```

```
t1_15_result <- summary (m1)
t1_15_result
```

```
Call:
multinom(formula = country ~ F_rights + F_steal + F_crime + F_religion +
    F_realizeself + F_dogood + F_violence, data = dwvs, family = multinomial,
    maxit = 3926, hess = TRUE)

Coefficients:
            (Intercept)  F_rights   F_steal   F_crime F_religion F_realizeself
Nigeria       0.6472240 -2.122889 0.5537755 0.5976742   2.887712     2.8922393
Philippines   0.9959826 -1.466848 1.5556448 1.0662700   1.932117     0.9680894
               F_dogood F_violence
Nigeria     -0.01110756  1.2844737
Philippines  1.16765772  0.7560728

Std. Errors:
            (Intercept)  F_rights   F_steal   F_crime F_religion F_realizeself
Nigeria       0.1612100 0.1642741 0.1735127 0.1334749  0.2143403     0.1695936
Philippines   0.1510987 0.1475679 0.1685630 0.1296852  0.1866007     0.1556415
             F_dogood F_violence
Nigeria     0.1218099  0.1534901
Philippines 0.1197516  0.1470007

Residual Deviance: 2693.234
AIC: 2725.234
```

We can see that there are 2 different regression model estimates: the first one compares the probability of Nigeria to the probability of the Netherlands, the second model compares the probability of the Philippines to the probability of the Netherlands. All the parameters are significant except *F_dogood* for Nigeria, which's not significantly different from 0. The sign of the parameters is the same for all the parameters in both the regressions, except for *F_dogood* where the Nigeria coefficient is not significant. This could be explained because have we analyzed previously Netherlands strongly differs from the other two countries, while Nigeria and the Philippines have not a clear separation. This analysis shows that sample data is more likely to belong to Nigeria and the Philippines than to the Netherlands when it has a higher positive value in the parameters of steal, crime, violence, religion, and a lower negative value in rights These coefficients could be probably well explained from the fact that the Netherlands is one of the most developed countries in all the world, the statistics of the Human Development Index published by the United Nations Development Programme places it in the 8th place in the world, while Nigeria and the Philippines are both considered developing states (161 and 107 respectively in the HDI ranking).

```
#### compute hitrate training data####
train.pred<-predict(m1,newdata=dwvs)
tab<-table(dwvs$country, train.pred)
kbl(sum(diag(tab))/sum(tab))
```

x
0.8571065

Error rate: 0.1428935

```
########compute LOOCV
nobs<- 3926
hit<-rep(0,nobs)
for (i in 1:nobs){
  train<-c(1:nobs)
  mod<- multinom(country ~ F_rights + F_steal + F_crime + F_religion +
                   F_realizeself + F_dogood + F_violence, data=dwvs,
                 subset=train[-i], print= FALSE,maxit=3926)
  pred<- predict(mod, newdata=dwvs[i,])
  hit[i]<-ifelse(pred==dwvs$country[i],1,0)
}
```

```
#hitrate
mean(hit)  #### LOOCV
```

Error rate: 0.1444218

The Multinomial logistic regression model has slightly better error values than the other models, except for KNN.

2 Task 2

2.1 Introduction

For task 2, we are going to deal with a dataset containing information about 4601 webmails. We have 48 variables describing the frequency of some specific words like *"remove"* in each observation, 6 variables describing the frequency of some specific chars like *"$"* in one observation, and three variables, *capital_run_length_longest*, *capital_run_length_average* and *capital_run_length_total*, describing the length of the longest uninterrupted sequence of capital letters, the average length of uninterrupted sequences of capital letters, and the total number of capital letters in each observation respectively. We also have a variable called *spam*, which indicates whether this webmail is a spam with 0 and 1, where 1 for spam, and 0 for not spam. Here all our variables are numeric type. Our task is to use these 57 attribute variables to classify whether a webmail is spam.

2.2 Methodology

In order to validate the accuracy of our methods, we firstly divide our dataset into a train set, which contains 2500 observations, and a test set, which contains 2101 observations. We use the train set to train our models, and then apply it to the test set to validate its accuracy.

2.3 Results

2.3.1 1. Classification Trees

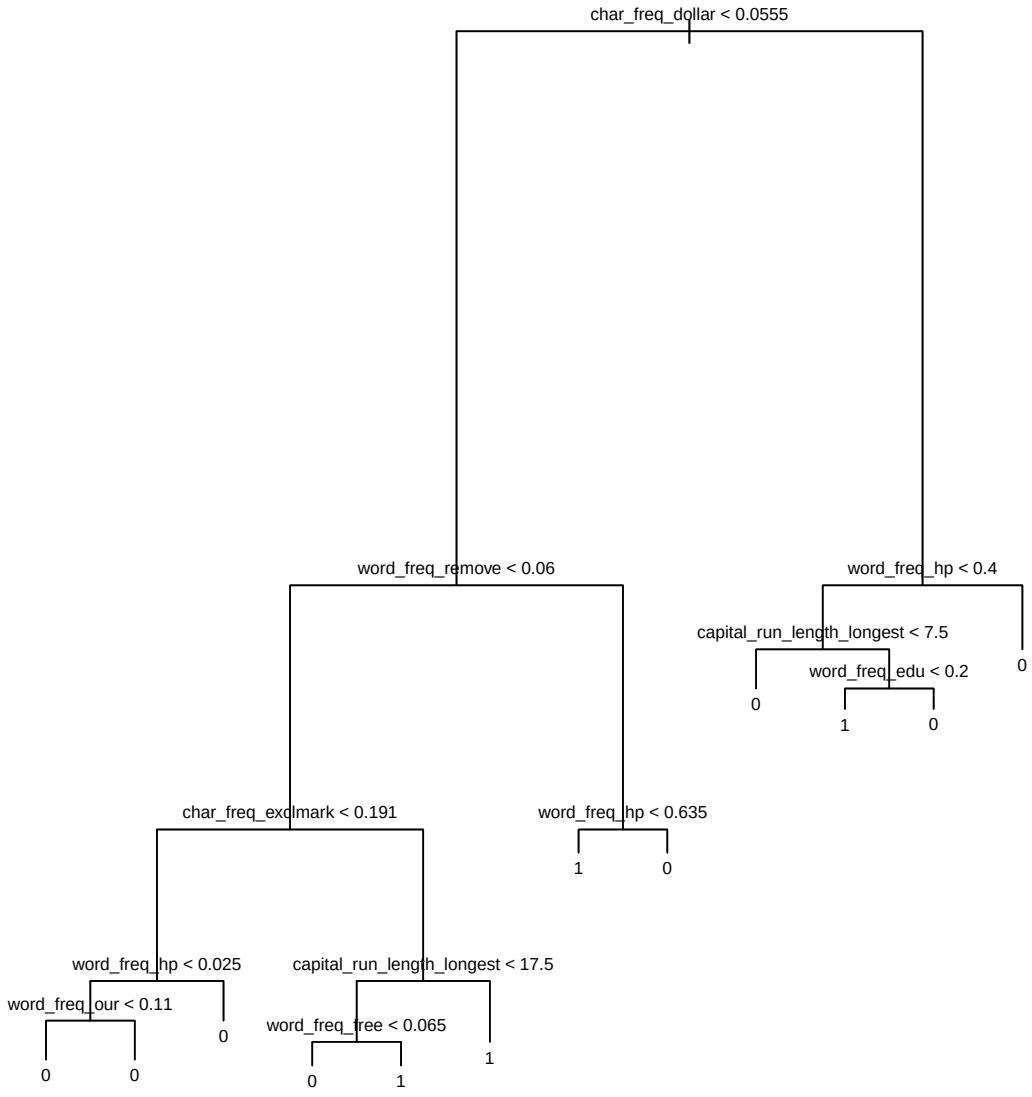
In this part, we are going to discuss the results obtained by complex tree model and pruned tree model. We begin with construct a complex tree model by dividing our observation into small non-overlapping regions according to some numerical criteria. Here we split our dataset until each leaf of our classification tree contains only less then 2 observations. The method used here is recursive binary splitting.

```
#grow complex tree using deviance as criterion
tree.mod = tree(factor(spam) ~ . ,data.train,
                control = tree.control(nobs = 2500, minsize = 2, mincut = 1),
                split = "deviance")

summary(tree.mod)
```

```
Classification tree:
tree(formula = factor(spam) ~ ., data = data.train, control = tree.control(nobs = 2500,
    minsize = 2, mincut = 1), split = "deviance")
Variables actually used in tree construction:
[1] "char_freq_dollar"          "word_freq_remove"
[3] "char_freq_exclmark"        "word_freq_hp"
[5] "word_freq_our"             "capital_run_length_longest"
[7] "word_freq_free"            "word_freq_edu"
Number of terminal nodes:  12
```

```
Residual mean deviance:  0.4883 = 1215 / 2488
Misclassification error rate: 0.084 = 210 / 2500
```

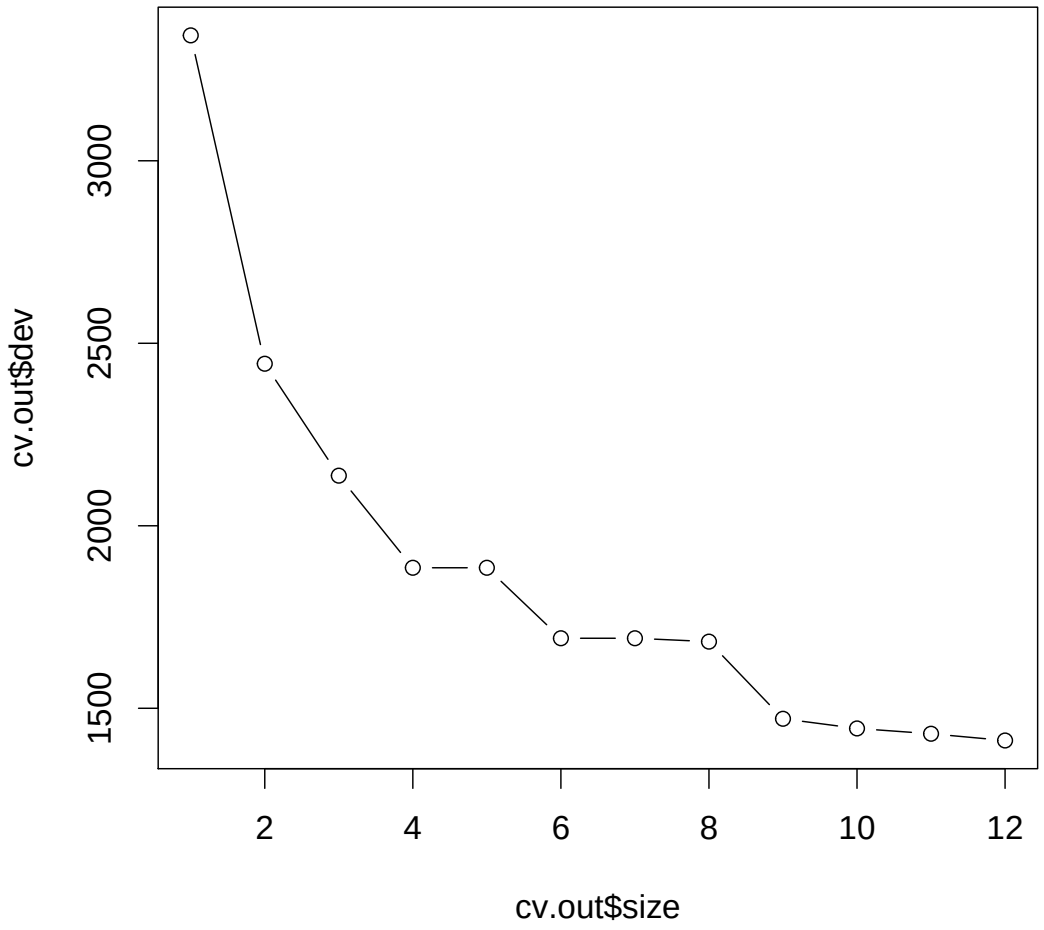
```r
#plot tree
plot(tree.mod)
text(4, 7, expression(bar(x) == sum(frac(x[i], n), i==1, n)))
#rpart.plot(tree.mod)
text(tree.mod,pretty=10,cex=0.65)
```



We can see clearly here that criteria concerning the frequency of "$", "remove", "!", "hp", "our", "free", "edu" as well as the length of the longest uninterrupted sequence of capital letters are used for splitting. It's actually quite reasonable, because from our own experience, spam webmails are always advertisements on money related topics or education related topics, and are always filled with words in capital letters, together with exclamation symbols, to draw attention.

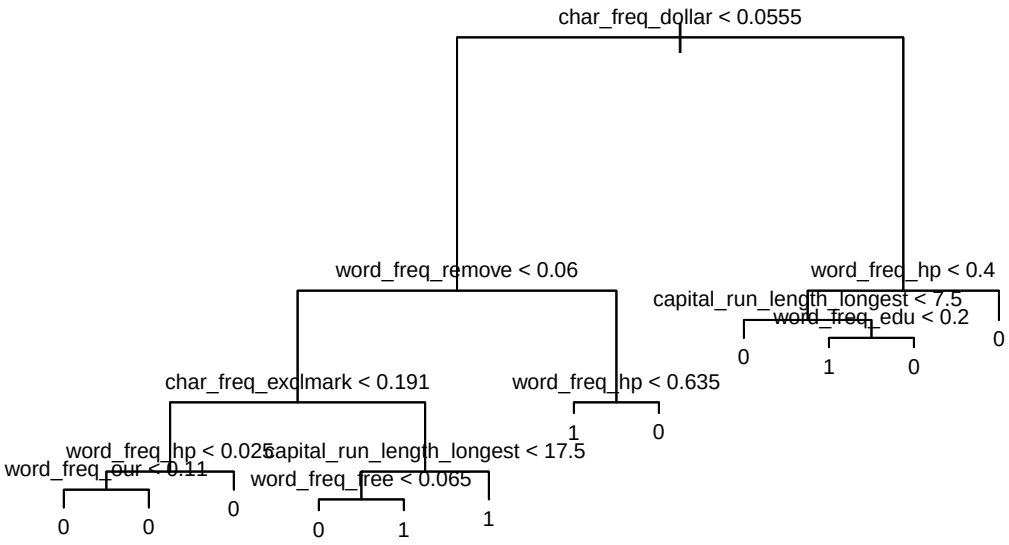
Since the process of recursive binary splitting may lead to overfitting, where we obtain a complex tree with a good fit on the training data, but with a poor performance on test data, we introduce the process of tree pruning. Actually, a smaller tree with fewer splits may have a lower variance at the cost of acceptable little bias. In order to decide the optimal tuning parameter which leads to both much lower variance and acceptable bias, we use cross-validation to make a selection. In fact, the least cross-validation error implies the least probability of overfitting, as it's also a train-test process.

```
#use cross-validation to select tuning parameter for pruning the tree
set.seed(0829539)
cv.out=cv.tree(tree.mod,K=5)
par(cex=1.4)
plot(cv.out$size,cv.out$dev,type='b')
```



However, in this specific task, we can see that the cross-validation error is monotonously decreasing, so the optimal fold number is the original fold number. We choose best size=12 here such that the pruned tree model here is exactly the same as our complex tree model.

```
#prune the tree
prune.mod=prune.tree(tree.mod,best=12)
plot(prune.mod)
text(prune.mod,pretty=10, cex=0.65)
```



Now we validate the accuracy of our classification tree model with the test data.

```
#make predictions on training and test set using the unpruned tree
pred.train<- predict(tree.mod,newdata=data.train)
classif.train<-ifelse(pred.train[,2]>=pred.train[,1],1,0)
t26 <- err(data.train$spam,classif.train)
kbl(t26$tab)
```

	0	1
0	1460	67
1	143	830

```
#make predictions on training and test set using the unpruned tree
pred.test<-predict(tree.mod,newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=pred.test[,1],1,0)
t27 <- err(data.test$spam,classif.test)
kbl(t27$tab)
```

	0	1
0	1209	52
1	153	687

```
#make predictions on training and test set using the pruned tree
pred.train<-predict(prune.mod,newdata=data.train)
classif.train<-ifelse(pred.train[,2]>=pred.train[,1],1,0)
t28 <- err(data.train$spam,classif.train)
kbl(t28$tab)
```

	0	1
0	1460	67
1	143	830

```
#make predictions on training and test set using the pruned tree
pred.test<-predict(prune.mod,newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=pred.test[,1],1,0)
t29 <- err(data.test$spam,classif.test)
kbl(t29$tab)
```

	0	1
0	1209	52
1	153	687

We can conclude that our classification model performs very well. since the complex tree is the same as pruned tree here, we can see that the test error is just very slightly higher than the train error. There is not much overfitting here.

2.3.2 2. Different Classification Models

Now for this part, we are going to compare different classification models using two different classification scenarios.

Firstly, we consider the scenario where we have different prior probabilities and equal classification costs. We use the entire dataset to figure out the prior probability.

```
#(a)Account for different prior probabilities and equal classification costs
# (1) linear discriminant analysis
set.seed(0829539)
ldaS.out<-lda(spam ~ .,data = spamdata)
print(ldaS.out$prior)
```

```
        0         1
0.6059552 0.3940448
```

So, we take P (is spam)=0.394, P (not spam) =0.606 as our prior probability, and we can verify that their sum equals 1.

The four classification models we are going to compare is 1) Linear Discriminant Analysis, 2) Bagging, 3) Random Forests and 4) Gradient Boosting. Since the dimension here is very high

and the mathematical assumption of our dataset is very weak, we do not suppose that Linear Discriminant Analysis without Principal Component analysis here will have very good performance. We just take it as a baseline. The three other methods are all methods used to improve the classification tree model, as tree models are always with high variance, thus high probability to cause overfitting.

The results are listed as below:

```
lda.out<-lda(spam~.,prior=c(0.606,0.394),data=data.train)

scaling <- lda.out$scaling %>% as.data.frame() %>% mutate(name = rownames(lda.out$scaling))
scaling %>% filter(name %in% c("char_freq_dollar", "word_freq_remove", "word_freq_table",
                               "word_freq_conference", "char_freq_dotcomma",
                               "char_freq_parenthesis", "char_freq_bracket", "char_freq_exclma
```

2.3.2.1 1) Linear Discriminant Analysis

```
         LD1                  name
1  0.8727936      word_freq_remove
2 -0.7164016       word_freq_table
3 -0.2623494  word_freq_conference
4 -0.6216711    char_freq_dotcomma
5 -0.3637729 char_freq_parenthesis
6 -0.2686609     char_freq_bracket
7  0.3970286    char_freq_exclmark
8  0.8798683      char_freq_dollar
```

Here we can see that the three attributes highly correlated with spam webmails the frequency of "$", "!" and "remove", are exactly the same as what we have got from our classification tree model. What is interesting here is we can also see that the frequency of "table", ";" and "parenthesis" shows strong negative correlation with spam webmails.

```
pred.train<-predict(lda.out,data.train)
t212 <- err(data.train$spam,pred.train$class)
kbl(t212$tab)
```

	0	1
0	1460	67
1	202	771

```
pred.test<-predict(lda.out,data.test)
t213 <- err(data.test$spam,pred.test$class)
kbl(t213$tab)
```

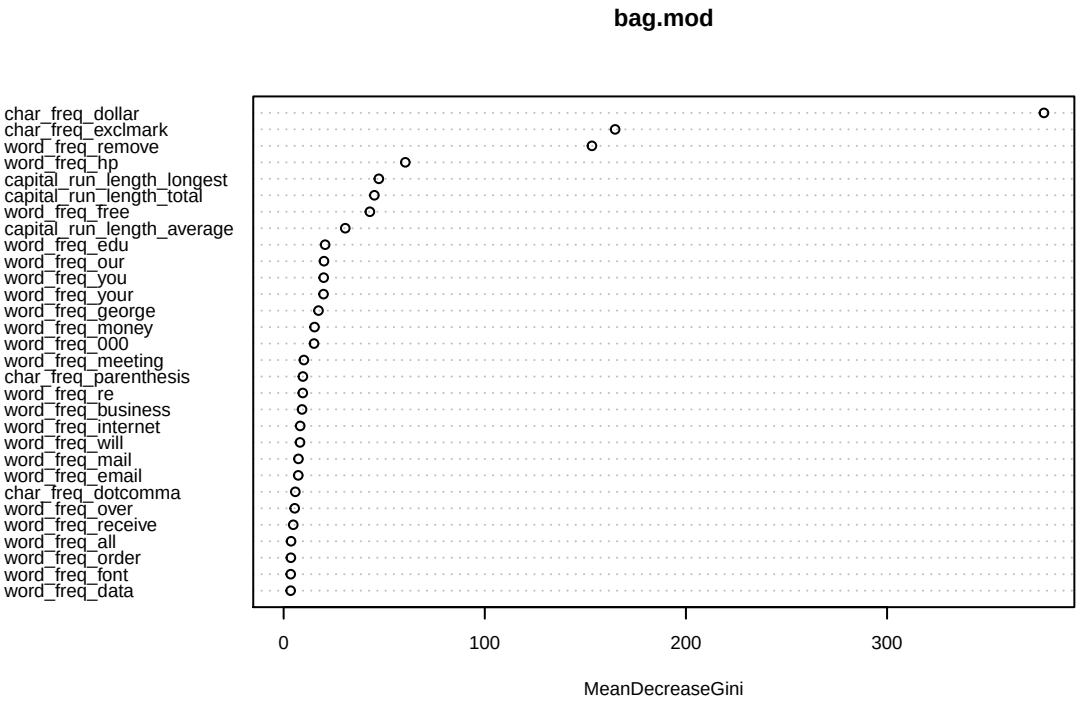
	0	1
0	1205	56
1	195	645

2.3.2.2 2) Bagging just some text

```
#(2) Bagging
set.seed(0829539)
bag.mod=randomForest(spam ~ ., data = data.train,
                     classwt=c(0.606,0.394),
                     mtry=57, ntree=nrotree, importance=TRUE)
bag.mod
```

```
Call:
 randomForest(formula = spam ~ ., data = data.train, classwt = c(0.606,      0.394), mtry = 57
               Type of random forest: classification
                     Number of trees: 5000
No. of variables tried at each split: 57

        OOB estimate of  error rate: 6.76%
Confusion matrix:
      0_no 1_yes class.error
0_no  1463    64  0.04191225
1_yes  105   868  0.10791367
```

```
#importance(bag.mod,plot=TRUE)
varImpPlot(bag.mod, type = 2, cex = 0.6)
```




```
pred.train<-predict(bag.mod,newdata=data.train)
t218 <- err(data.train$spam,pred.train)
kbl(t218$tab)
```

	0_no	1_yes
0	1527	0
1	0	973

```
pred.test<-predict(bag.mod,newdata=data.test)
t219 <- err(data.test$spam,pred.test)
#kbl(t219$tab)
```

Actually, with the MeanDecreaseGini graph we can see that the frequency of "$", "!" and "remove" made evidently the most contribution. This is the same conclusion we have drawn from our classification tree model and Linear Discriminant Analysis. Here Out Of Bag error rate is 6.76%, which is acceptable.

2.3.2.3 3) Random Forests just some text.

```r
# (3) random forests
set.seed(0829539)
rf.mod=randomForest(spam ~ ., data = data.train,
                    classwt=c(0.606,0.394), mtry=5,
                    ntree=5000, importance=TRUE)
rf.mod
```

```
Call:
 randomForest(formula = spam ~ ., data = data.train, classwt = c(0.606,      0.394), mtry = 5,
               Type of random forest: classification
                     Number of trees: 5000
No. of variables tried at each split: 5

        OOB estimate of  error rate: 5.12%
Confusion matrix:
      0_no 1_yes class.error
0_no  1484    43  0.02815979
1_yes   85   888  0.08735868
```

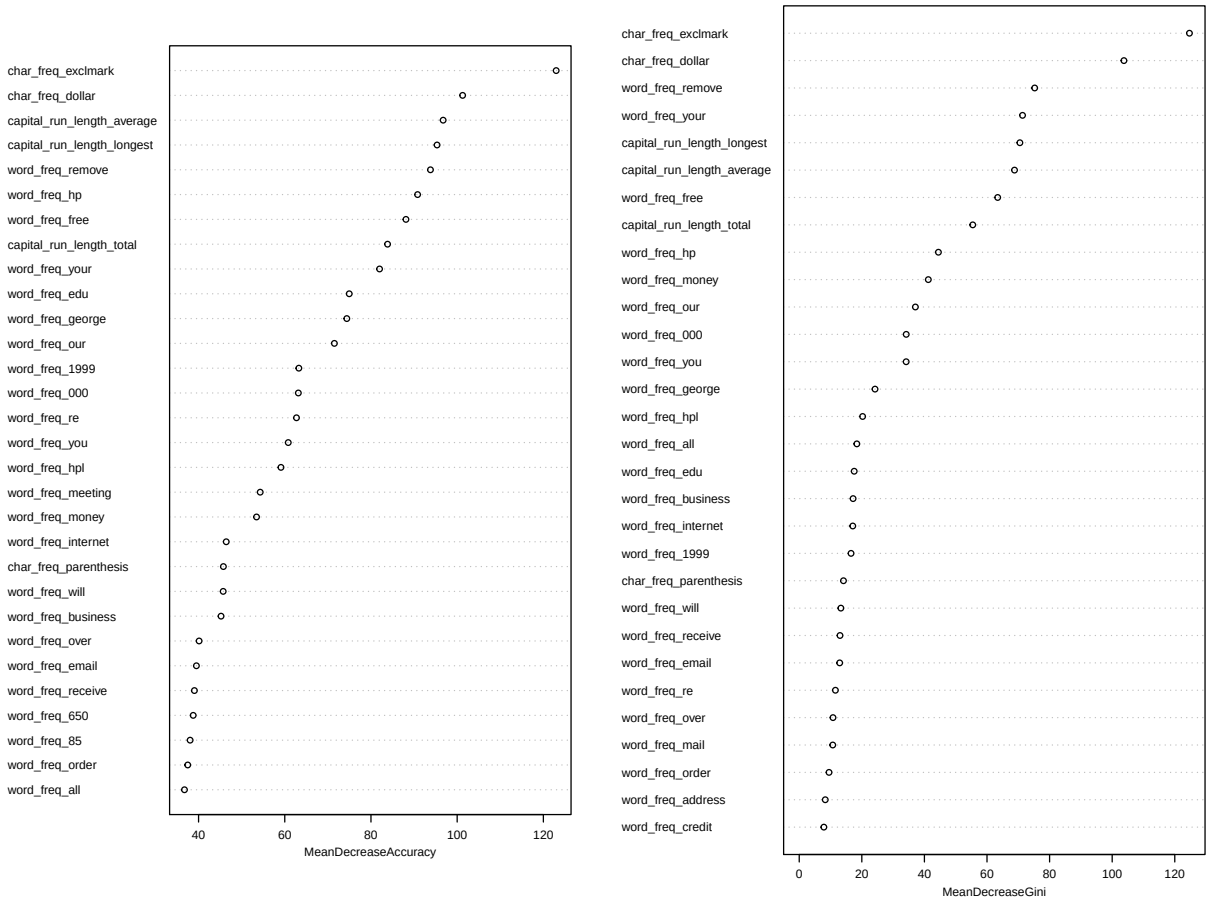
```r
pred.train<-predict(rf.mod,newdata=data.train)
t2_21Y1 <- err(data.train$spam,pred.train)
t2_21Y1$tab
```

```
        predicted
observed 0_no 1_yes
       0 1527     0
       1    9   964
```

```r
pred.test<-predict(rf.mod,newdata=data.test)
t2_21Y2 <-err(data.test$spam,pred.test)
t2_21Y2$tab
```

```
        predicted
observed 0_no 1_yes
       0 1227    34
       1   72   768
```

```r
varImpPlot(rf.mod, cex = 0.6)
```




```
#varImpPlot(bag.mod, type = 2, cex = 0.6)
```

Random Forests Model is also based on the idea of decorrelating trees. Here we choose m=5 for the size of our predictor set. can see that although there is slightly difference, "$", "!" and "remove" still stand for spam webmails, as well as the length of the longest uninterrupted sequence of capital letters.

2.3.2.4 4) Gradient Boosting Different from Bagging and Random Forests, the number of trees for Gradient Boosting is not expected to be as large as possible, since using too many trees may cause overfitting for Gradient Boosting. We start with deciding the optimal number of trees by cross-validation, for the same reason explained before.

Note that since we are going to suppose Bernoulli distribution in our gbm function, we should firstly make sure that our variable for classification is numeric type.

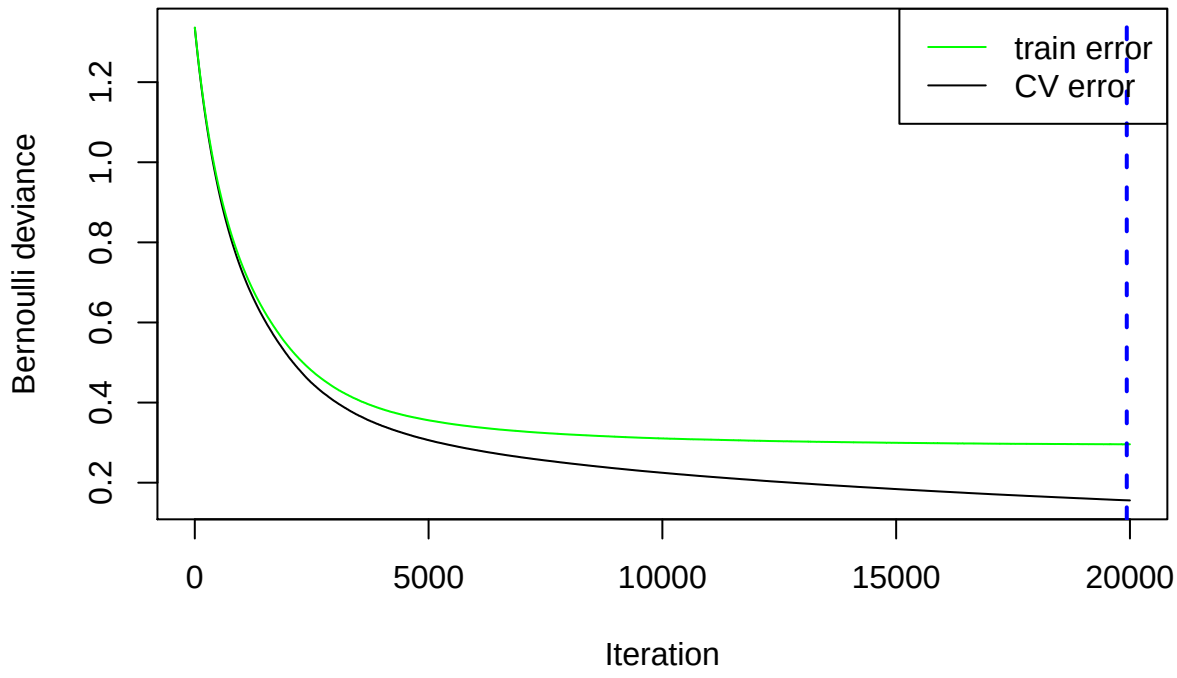
```
# (4) gradient boosting
set.seed(0829539)
data.train$spam<-ifelse(data.train$spam=="1_yes",1,0)
```

```
data.test$spam<-ifelse(data.test$spam=="1_yes",1,0)
spamdata$spam<-ifelse(spamdata$spam=="1_yes",1,0)
```

```
load(paste0(base_url,"T2/T2_gbm_3.Rdata"))
boost.mod<-t2_gbm_3
```

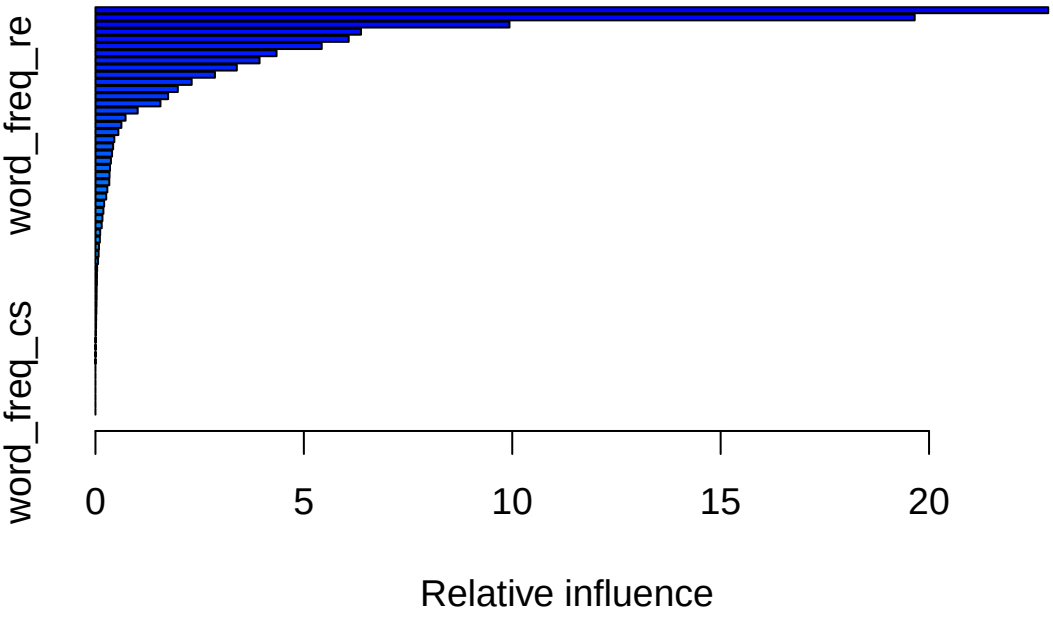
```
#use distribution="bernoulli" for binary target
#interaction.depth=4, means that we fit a tree that uses 4 splits
#(and that includes at most a 4-th order interaction)
boost.mod=gbm(spam ~ . ,data = data.train, distribution = "bernoulli",
              n.trees = 20000, interaction.depth = 4, shrinkage = 0.001, cv.folds = 5)
gbm.perf(boost.mod,method="cv")
legend("topright",c("train error","CV error"),col=c("green","black"),lty=c(1,1))
```

```
[1] 19932
```



So here we get that the optimal number of trees is 19932. We use this parameter to continue our analysis.

```
#relative influence plot
par(cex=1.2)
summary(boost.mod,n.trees=19932)
```



	var	rel.inf
char_freq_dollar	char_freq_dollar	2.285917e+01
char_freq_exclmark	char_freq_exclmark	1.965558e+01
word_freq_remove	word_freq_remove	9.933912e+00
word_freq_hp	word_freq_hp	6.371358e+00
word_freq_your	word_freq_your	6.075841e+00
word_freq_free	word_freq_free	5.428856e+00
capital_run_length_longest	capital_run_length_longest	4.344714e+00
capital_run_length_average	capital_run_length_average	3.937332e+00
capital_run_length_total	capital_run_length_total	3.391273e+00
word_freq_george	word_freq_george	2.866851e+00
word_freq_edu	word_freq_edu	2.308666e+00
word_freq_our	word_freq_our	1.976375e+00
word_freq_money	word_freq_money	1.743456e+00
word_freq_000	word_freq_000	1.559673e+00
word_freq_you	word_freq_you	1.013728e+00
word_freq_meeting	word_freq_meeting	7.223353e-01
word_freq_re	word_freq_re	6.227953e-01
word_freq_internet	word_freq_internet	5.493848e-01
word_freq_1999	word_freq_1999	4.530588e-01
word_freq_business	word_freq_business	4.244831e-01
word_freq_over	word_freq_over	3.985102e-01
word_freq_email	word_freq_email	3.710928e-01

```
word_freq_receive                 word_freq_receive 3.484809e-01
word_freq_will                       word_freq_will 3.377431e-01
char_freq_dotcomma                 char_freq_dotcomma 3.321253e-01
char_freq_parenthesis           char_freq_parenthesis 2.853891e-01
word_freq_report                   word_freq_report 2.587867e-01
word_freq_technology             word_freq_technology 2.084485e-01
word_freq_mail                       word_freq_mail 1.903978e-01
word_freq_hpl                         word_freq_hpl 1.677002e-01
word_freq_650                         word_freq_650 1.504262e-01
word_freq_3d                           word_freq_3d 1.131915e-01
word_freq_all                         word_freq_all 1.063984e-01
word_freq_font                       word_freq_font 8.440753e-02
word_freq_project                 word_freq_project 7.595445e-02
word_freq_make                       word_freq_make 6.275894e-02
word_freq_pm                           word_freq_pm 4.042293e-02
word_freq_address                 word_freq_address 3.613297e-02
word_freq_order                     word_freq_order 3.496306e-02
word_freq_parts                     word_freq_parts 2.876464e-02
word_freq_conference           word_freq_conference 2.842824e-02
word_freq_credit                   word_freq_credit 2.467152e-02
word_freq_people                   word_freq_people 2.197801e-02
word_freq_85                           word_freq_85 1.685307e-02
char_freq_pound                   char_freq_pound 1.468910e-02
word_freq_data                     word_freq_data 1.128843e-02
word_freq_labs                     word_freq_labs 4.584238e-03
char_freq_bracket                 char_freq_bracket 3.254895e-03
word_freq_direct                   word_freq_direct 2.550972e-03
word_freq_lab                         word_freq_lab 4.378566e-04
word_freq_original               word_freq_original 1.606371e-04
word_freq_addresses             word_freq_addresses 9.088811e-05
word_freq_table                     word_freq_table 7.843543e-05
word_freq_telnet                   word_freq_telnet 0.000000e+00
word_freq_857                         word_freq_857 0.000000e+00
word_freq_415                         word_freq_415 0.000000e+00
word_freq_cs                           word_freq_cs 0.000000e+00
```

We can actually get nearly the same result as we obtained from all the former models concerning the variable most correlated to spam webmails here, that the frequency of "$", "!" and "remove" come top.

```
pred.train<-predict(boost.mod,n.trees=19932,newdata=data.train,type="response")
classif.train<-ifelse(pred.train>=1-pred.train,1,0)
t225 <- err(data.train$spam,classif.train)
t225$tab
```

```
        predicted
observed    0    1
       0 1547  953
```

```
pred.test<-predict(boost.mod,n.trees=19932,newdata=data.test,type="response")
classif.test<-ifelse(pred.test>=1-pred.test,1,0)
t226 <- err(data.test$spam,classif.test)
t226$tab
```

```
        predicted
observed    0    1
       0 1287  814
```

Then, we consider the scenario where we have different prior probabilities and unequal classification costs: C(spam|email)=10*C(email|spam). Actually, the models here remain the same, and we just need to do some reclassification based on the posterior probabilities obtained from our built models and a criterion taking the classification costs as weights for posterior probabilities.

2.3.2.4.1 1) Linear Discriminant Analysis some text here

```
pred.train<-predict(lda.out,data.train)
classif.train<-ifelse(1*pred.train$posterior[,2]>=10*pred.train$posterior[,1],1,0)
t227Y2 <- err(data.train$spam, classif.train)
t227Y2$tab
```

```
        predicted
observed    0    1
    0_no 1878  622
```

```
pred.test<-predict(lda.out,data.test)
classif.test<-ifelse(1*pred.test$posterior[,2]>=10*pred.test$posterior[,1],1,0)
t227Y3 <- err(data.test$spam,classif.test)
t227Y3$tab
```

```
        predicted
observed    0    1
    0_no 1601  500
```

2.3.2.4.2 2) bagging some text here

```
pred.train<-predict(bag.mod,type="prob",newdata=data.train)
classif.train<-ifelse(1*pred.train[,2]>=10*pred.train[,1],1,0)
t228Y1 <- err(data.train$spam,classif.train)
t228Y1$tab
```

```
        predicted
observed    0    1
    0_no 1702  798
```

```
pred.test<-predict(bag.mod,type="prob",newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=10*pred.test[,1],1,0)
t228Y2 <- err(data.test$spam,classif.test)
t228Y2$tab
```

```
        predicted
observed    0    1
    0_no 1512  589
```

2.3.2.4.3 3) Random Forests some text here

```
pred.train<-predict(rf.mod,type="prob",newdata=data.train)
classif.train<-ifelse(1*pred.train[,2]>=10*pred.train[,1],1,0)
t229Y1 <- err(data.train$spam,classif.train)
t229Y1$tab
```

```
        predicted
observed    0    1
    0_no 1745  755
```

```
pred.test<-predict(rf.mod,type="prob",newdata=data.test)
classif.test<-ifelse(1*pred.test[,2]>=10*pred.test[,1],1,0)
t229Y2 <- err(data.test$spam,classif.test)
t229Y2$tab
```

```
        predicted
observed    0    1
    0_no 1645  456
```

4) Gradient Boosting **some text here**

```
pred.train<-predict(boost.mod,n.trees=19932,newdata=data.train,type="response")
classif.train<-ifelse(1*pred.train>=10*(1-pred.train),1,0)
t230Y1 <- err(data.train$spam,classif.train)
t230Y1$tab
```

```
        predicted
observed    0    1
    0_no 1693  807
```

```
pred.test<-predict(boost.mod,n.trees=19932,newdata=data.test,type="response")
classif.test<-ifelse(1*pred.test>=10*(1-pred.test),1,0)
t230Y2 <- err(data.test$spam,classif.test)
t230Y2$tab
```

```
        predicted
observed    0    1
    0_no 1467  634
```

We arrange all the training errors and test errors into a whole table below, and calculate the sensitivities as well as the false positive rates for the test set, using the probability tables we printed out:

Table 4

-	training error	test error	sensitivity	false positive rate
a)LDC	0.1076	0.1195	0.7679	0.0444
a)Bagging	0	0.0643	0.8869	0.0317
a)Random Forests	0.0036	0.0505	0.9143	0.0270
a)Gradient Boosting	0.0216	0.0571	0.9131	0.0373
b)LDC	0.2128	0.2285	0.4452	0.0111
b)Bagging	0.07	0.1290	0.6893	0.0008
b)Random Forests	0.0872	0.1856	0.5393	0.0002
b)Gradient Boosting	0.068	0.1066	0.7440	0.0007

We can see that as we claimed at the beginning, Linear Discriminant Analysis performs the worst in both scenarios, Random Forests performs the best in scenario a), where we have different prior probabilities but equal classification costs, and Gradient Boosting performs the best in scenario b), where we have different prior probabilities and unequal classification costs. Generally, for scenario a), the performance of Random Forests and Gradient Boosting is equally good, and Bagging also performs good, just with slightly higher test error than Random Forests and Gradient Boosting. However, for scenario b), the accuracy of Random Forests decreases much faster than Bagging.

What's more, as is expected, in scenario b), since misclassifying a not spam webmail into spam ones costs much more than misclassifying a spam webmail into not spam ones, our models all tend to classify more cases as not spam ones to minimize the cost. As a result, the false positive rate is very low for all models, but the accuracy and sensitivity decrease significantly for all models, since a number of real positive cases are classified as not spam webmails. Gradient Boosting and Bagging are relatively more stable here compared with the other two models.

3 Task 3

3.1 1. Introduction

For task 3, we are going to deal with a dataset containing the shopping behavior of 487 different customers. We have 11 statements here and the corresponding variables measuring to what extent they agree with these statements, namely: 1) organising_trip: It is very important for me to organize the shopping well. 2) knowing_buy: When I leave to go shopping, I know exactly what I am going to buy. 3) duty_responsability: By doing the shopping, I fulfill my duty and take my responsibility. 4) shopping_fun: I enjoy doing the shopping. 5) take_at_ease: I do the shopping at a leisurely pace. 6) enjoy: I enjoy the atmosphere while shopping. 7) shopping_drag: Doing the shopping is a drag. 8) minimise_shoppingtime: I try to keep the time that I spend doing

the shopping to a minimum. 9) shopping_list: I usually take a list with me when I go to do the shopping. 10) shopping_with_family: I like shopping with the whole family. 11) have_stock: I like having a stock of products on hand at home.

The variables vary from 1 to 7, where 1 means totally disagree and 7 means totally agree. Our task is to cluster the customers basing on the 11 items of shopping mentioned above and try to interpret the results we will get. What's more, we are also going to test the stability of the cluster solutions deduced by different methods.

3.2 2. Methodology

We will try to do the clustering with (1) hierarchical clustering with Ward's method on squared Euclidean distances followed by k-means with the centroid of the hierarchical clustering as starting point, (2) model-based clustering with hddc(), and (3) model-based clustering using Mclust(). In order to test the stability of our cluster solutions, we are going to split our dataset randomly into a train set and a test set. The cluster methods will be firstly applied on the train set, and then on test set. After that, we will also form a cluster solution on the test set by assigning the points to the cluster centroid of the training sample that is closest. Finally, by comparing our two cluster solutions on the test set, we can evaluate the stability of our methods.

3.3 3. Result

Firstly, in order to make our cluster solution more reasonable, we are going to standardize our data first.

```
## standardize variables
shopping<-scale(shopping,center=TRUE,scale=TRUE)
```

Then, we form our train set and test set randomly for the validation work.

```
#create train and test set
set.seed(0829539)
sel<-sample(1:487,size=250,replace=FALSE)
train<-shopping[sel,]
valid<-shopping[-sel,]
```

Before applying our cluster methods on our dataset, we first try to do an exploratory factor analysis to summarize our 11 attributes, as we are going to make use of it for the interpretation of our cluster solutions. We start with a principal component analysis.

```
#compute variance accounted for by each component
kbl(round(prcomp.out$sd^2/sum(prcomp.out$sd^2),3))
```

x
0.341
0.207
0.090
0.079
0.070
0.056
0.045
0.036
0.034
0.026
0.016

As is shown here, only the first two components accounts for more than 10% of the variances, and they account for 54.8% of the variances in total. We then continue to conduct the exploratory factor analysis with 2 factors.

According to the result shown above, we can define two factors here. Enjoy: The extent to which someone enjoys shopping, summarizing the attributes "shopping_fun", "take_at_ease", "enjoy", "shopping_drag", "minimise_shoppingtime" and "shopping_with_family". Here we can see that the loadings for "shopping_drag" and "minimise_shoppingtime" is negative, and the others are positive, which is quite reasonable. Organized: The extent to which someone is organized when doing shopping, summarizing the attributes "organising_trip", "knowing_buy", "duty_responsability", "shopping_list" and "have_stock". So, we can use these two factors to classify our customers as "enjoy shopping and organized", "enjoy shopping and not organized", "dislike shopping and organized", or "dislike shopping and not organized". We denote the principal components as comp for future use, as we are going to visualize our cluster solution in the space of the first two principal components.

```
comp<-as.matrix(shopping)%*%prcomp.out$rotation
```

We also define the "clusters" function here for classifying new points to the nearest cluster centroid. We are going to use this function for the validation of our cluster methods.

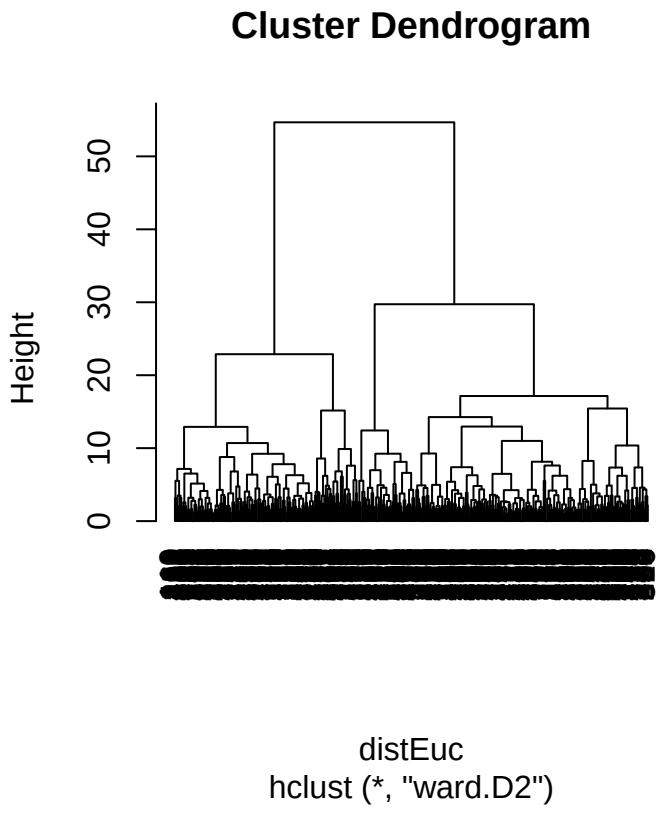
```
# function classify new points to nearest cluster centroid
clusters <- function(x, centers) {
  # compute squared euclidean distance from each sample to each cluster center
  tmp <- sapply(seq_len(nrow(x)),
                function(i) apply(centers, 1,
                                  function(v) sum((x[i, ]-v)^2)))
  max.col(-t(tmp))  # find index of min distance
```

Now we are going to apply our three different cluster methods on our dataset separately, and then interpret the results. For each method, we are going to compute the solution with 1 up to 6 solutions and choose the most reasonable solution.

3.3.1 I) Hierarchical clustering with Ward's method on squared Euclidean distances followed by k-means with the centroid of the hierarchical clustering as starting point

Firstly, we apply Hierarchical clustering with Ward's method on squared Euclidean distances and draw the dendrogram to observe the structure.

```
# compute Euclidean distances
distEuc<-dist(shopping, method = "euclidean", diag = FALSE, upper = FALSE)

##  hierarchical clustering method of Ward on squared Euclidean distance
hiclust_ward<- hclust(distEuc, "ward.D2")

## plot dendrogram
par(pty="s")
plot(hiclust_ward,hang=-1)
```



We can see from the dendrogram that cutting the tree at the height around 20, which means selecting 4 clusters, is quite reasonable. Then we continue with k-means method using the centroid of the hierarchical clustering as starting point, and compute solutions with 1 up to 6 clusters.

```
# classification of students for solutions with 1-6 clusters
clustvar<-cutree(hiclust_ward, k=1:6)
```

```
# k-means with 1 clusters using centroid of Ward as starting point
nclust<-1
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean1<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean1$centers,2)
```

3.3.1.1 a) 1 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1               0           0                   0            0            0
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1     0             0                    0             0                    0
  have_stock
1          0
```

```
# k-means with 2 clusters using centroid of Ward as starting point
nclust<-2
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean2<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean2$centers,2)
```

3.3.1.2 b) 2 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1           -0.07       -0.05                0.06         0.65         0.56
2            0.11        0.08               -0.10        -1.04        -0.90
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1  0.61         -0.66                -0.54         -0.04                 0.28
2 -0.97          1.06                 0.87          0.07                -0.44
  have_stock
1       0.04
2      -0.06
```

```
# k-means with 3 clusters using centroid of Ward as starting point
nclust<-3
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean3<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean3$centers,2)
```

3.3.1.3 c) 3 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1            0.36        0.40                0.28         0.69         0.56
2           -1.29       -1.26               -0.53         0.45         0.55
3            0.19        0.12               -0.08        -1.05        -0.94
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1  0.68         -0.65                -0.43          0.33                 0.41
2  0.30         -0.60                -0.73         -1.08                -0.09
3 -0.97          1.07                 0.87          0.11                -0.45
  have_stock
1       0.17
2      -0.33
3      -0.05
```

```
# k-means with 4 clusters using centroid of Ward as starting point
nclust<-4
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean4<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean4$centers,2)
```

3.3.1.4 d) 4 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1            0.37        0.42                0.29         0.69         0.57
2           -1.16       -1.20               -0.51         0.55         0.52
3            0.45        0.36                0.04        -1.03        -0.96
4           -1.00       -0.84               -0.56        -1.07        -0.67
```

```
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1  0.69         -0.65                -0.41          0.34                 0.44
2  0.39         -0.69                -0.87         -1.00                -0.14
3 -0.97          1.07                 0.87          0.44                -0.40
4 -0.98          1.02                 0.86         -1.15                -0.57
  have_stock
1       0.18
2      -0.31
3       0.24
4      -1.05
```

```
# k-means with 5 clusters using centroid of Ward as starting point
nclust<-5
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean5<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean5$centers,2)
```

3.3.1.5 e) 5 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1            0.52        0.45                0.80         0.86         0.66
2            0.20        0.26               -0.30         0.51         0.44
3           -1.33       -1.25               -0.46         0.55         0.59
4            0.45        0.36                0.04        -1.03        -0.96
5           -1.00       -0.84               -0.56        -1.07        -0.67
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1  0.87         -0.65                -0.45          0.28                 0.99
2  0.44         -0.63                -0.47          0.41                -0.15
3  0.48         -0.73                -0.79         -1.20                -0.13
4 -0.97          1.07                 0.87          0.44                -0.40
5 -0.98          1.02                 0.86         -1.15                -0.57
  have_stock
1       0.47
2      -0.18
3      -0.25
4       0.24
5      -1.05
```

```
# k-means with 6 clusters using centroid of Ward as starting point
nclust<-6
stat<-describeBy(shopping,clustvar[,nclust],mat=TRUE)
hcenter<-matrix(stat[,5],nrow=nclust)
rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
colnames(hcenter)<-c(colnames(shopping))
kmean6<-kmeans(shopping,centers=hcenter,iter.max=200)

round(kmean6$centers,2)
```

3.3.1.6 f) 6 cluster

```
  organising_trip knowing_buy duty_responsability shopping_fun take_at_ease
1            0.54        0.41                0.83         0.83         0.64
2            0.27        0.33                0.68         0.82         0.83
3            0.08        0.17               -0.49         0.49         0.33
4           -1.49       -1.40               -0.73         0.47         0.52
5            0.46        0.36                0.03        -1.04        -0.97
6           -1.00       -0.84               -0.56        -1.07        -0.67
  enjoy shopping_drag minimise_shoppingtime shopping_list shopping_with_family
1  0.88         -0.59                -0.42         0.82                 0.87
2  0.81         -0.81                -0.44        -1.17                 0.44
3  0.37         -0.61                -0.53         0.61                -0.02
4  0.37         -0.67                -0.81        -1.15                -0.22
5 -0.97          1.08                 0.87         0.45                -0.42
6 -0.98          1.02                 0.86        -1.15                -0.57
  have_stock
1       0.42
2       0.12
3      -0.06
4      -0.40
5       0.23
6      -1.05
```

We also list the sizes and the proportions of explained variance for all the solutions here.

```
##number of observations per cluster
kmean1$size
```

```
[1] 487
```

```
kmean2$size
```

```
[1] 300 187
```

```
kmean3$size
```

```
[1] 219  87 181
```

```
kmean4$size
```

```
[1] 214  86 143  44
```

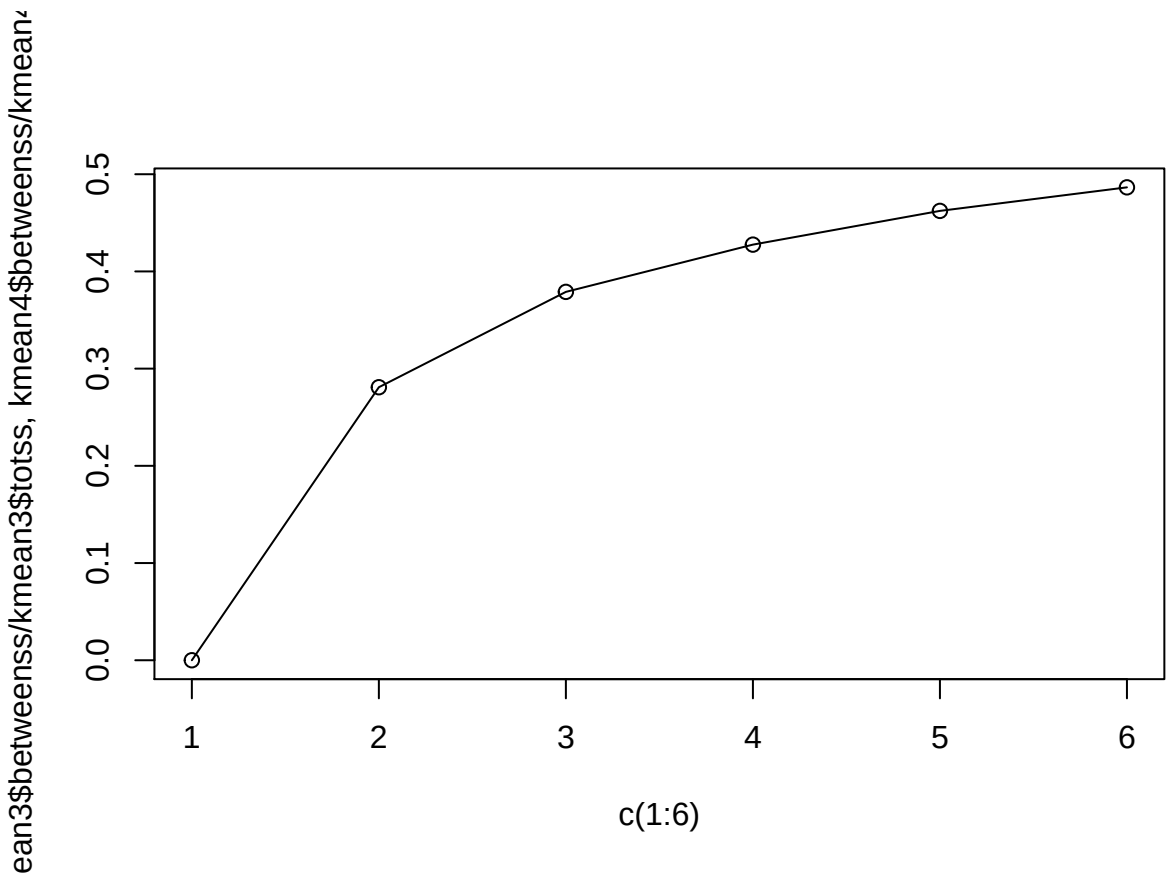
```
kmean5$size
```

```
[1] 110 115  75 143  44
```

```
kmean6$size
```

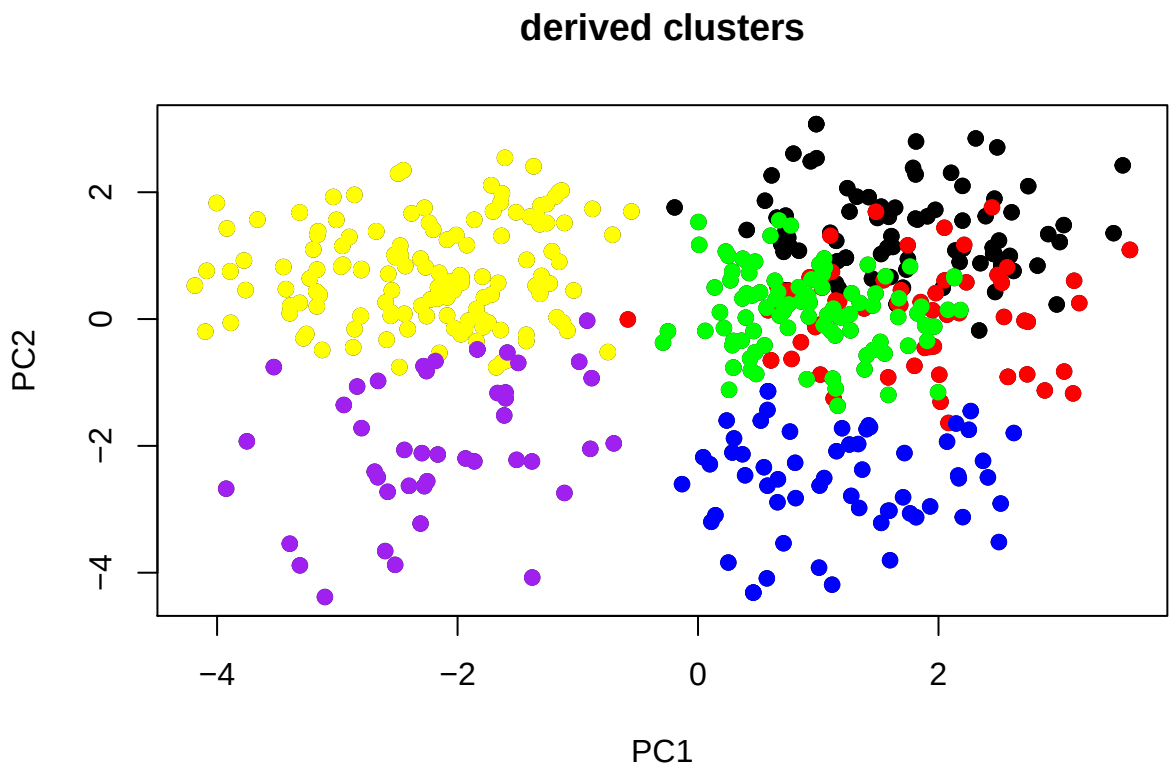
```
[1]  84  61  95  61 142  44
```

```
## proportion of explained variance
plot(c(1:6),c(kmean1$betweenss/kmean1$totss,kmean2$betweenss/kmean2$totss,
              kmean3$betweenss/kmean3$totss,kmean4$betweenss/kmean4$totss,
              kmean5$betweenss/kmean5$totss,kmean6$betweenss/kmean6$totss))
lines(c(1:6),c(kmean1$betweenss/kmean1$totss,kmean2$betweenss/kmean2$totss,
               kmean3$betweenss/kmean3$totss,kmean4$betweenss/kmean4$totss,
               kmean5$betweenss/kmean5$totss,kmean6$betweenss/kmean6$totss))
```



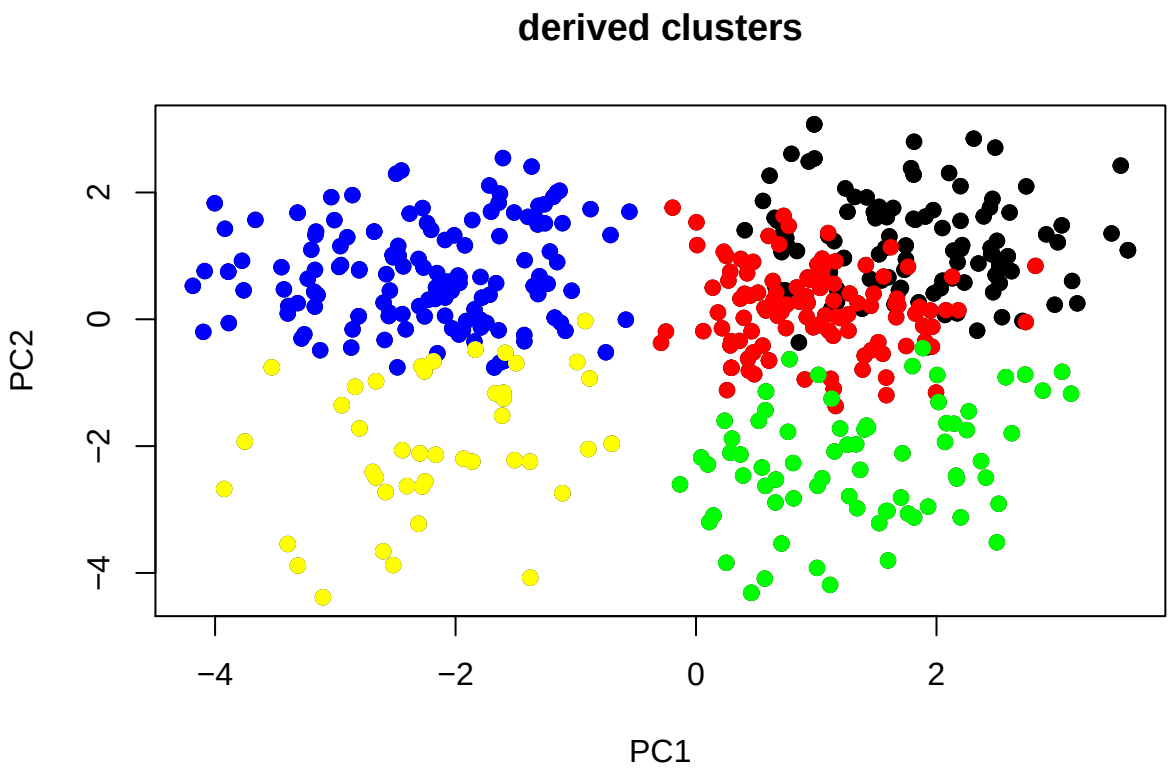
We can see here that the more clusters we select, the larger the proportion of explained variance is. However, the proportion grows lower and lower while we increase the number of clusters. When we take 6 clusters, the size of each cluster seems relatively small, but not too small, so we still decide to take 6 clusters now, in order to get maximum explained variance. Then we visualize our cluster solution in the space of the first two principal components.

```
plot(comp,main="derived clusters")
points(comp[kmean6$cluster==1,],col="black",pch=19)
points(comp[kmean6$cluster==2,],col="red",pch=19)
points(comp[kmean6$cluster==3,],col="green",pch=19)
points(comp[kmean6$cluster==4,],col="blue",pch=19)
points(comp[kmean6$cluster==5,],col="yellow",pch=19)
points(comp[kmean6$cluster==6,],col="purple",pch=19)
```



We can see that the green part and the red part seem to be totally mixed up with each other, while our principal component 1, the "enjoy" factor, still separates the clusters well into positive ones and negative ones, namely, those who enjoy shopping and those who dislike shopping. As the result corresponding to our principal component 2, the "organized" factor, seems really too hard to interpret, we decide to try to use 5 clusters, as this won't make a great decrease in the explained variance. We visualize our solution with 5 clusters as below:

```r
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[kmean5$cluster==1,],col="black",pch=19)
points(comp[kmean5$cluster==2,],col="red",pch=19)
points(comp[kmean5$cluster==3,],col="green",pch=19)
points(comp[kmean5$cluster==4,],col="blue",pch=19)
points(comp[kmean5$cluster==5,],col="yellow",pch=19)
```



Here the borders become much clearer. We can interpret the cluster solutions as: The blue part: Positive for the "organized" factor but negative for the "enjoy" factor. The customers who dislike shopping but are organized. The yellow part: Negative for the "organized" factor and negative for the "enjoy" factor. The customers who dislike shopping and are not organized. The green part: Negative for the "organized" factor but positive for the "enjoy" factor. The customers who enjoy shopping but are not organized. The red part: Around 0 for the "organized" factor and positive for the "enjoy" factor. The customers who enjoy shopping and are moderately organized. The black part: Positive for the "organized" factor and positive for the "enjoy" factor. The customers who enjoy shopping and are organized. This interpretation seems to make sense, so we continue with the validation work. We first conduct the cluster method on our train dataset to get several clusters, and the cluster our validation dataset with the same method. After that, we cluster our validation data set again according to the clusters we've got from our train dataset. By comparing the difference between the two results we get from our validation dataset, we can measure the stability of our cluster method. The measurement of difference here is always ARI index, which has zero expected value in the case of a random partition and is bounded above by 1 in the case of perfect agreement

between two partitions.

```
  # cluster train data Ward + kmeans
  disttrain<-dist(train, method = "euclidean", diag = FALSE, upper = FALSE)
  wardtrain<- hclust(disttrain, "ward.D2")
  nclust<-5
  clustvar<-cutree(wardtrain, k=nclust)
  stat<-describeBy(train,clustvar,mat=TRUE)
  hcenter<-matrix(stat[,5],nrow=nclust)
  rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
  colnames(hcenter)<-c(colnames(train))
  kmeantrain<-kmeans(train,centers=hcenter,iter.max=200)

  # cluster validation data Ward + kmeans
  distvalid<-dist(valid, method = "euclidean", diag = FALSE, upper = FALSE)
  wardvalid<- hclust(distvalid, "ward.D2")
  nclust<-5
  clustvar<-cutree(wardvalid, k=nclust)
  stat<-describeBy(valid,clustvar,mat=TRUE)
  hcenter<-matrix(stat[,5],nrow=nclust)
  rownames(hcenter)<-paste("c_",rep(1:nclust),sep="")
  colnames(hcenter)<-c(colnames(valid))
  kmeanvalid<-kmeans(valid,centers=hcenter,iter.max=200)


  classif1<-clusters(valid, kmeantrain[["centers"]])
  classif2<-kmeanvalid$cluster
  table(classif1,classif2)
```

```
        classif2
classif1  1  2  3  4  5
       1 12  0  1 43  0
       2  2 30  0  0  1
       3  0  0  9  0 20
       4  0  0 64  0  0
       5 40  1  0 14  0
```

```
  cl1<-as.factor(classif1)
  cl2<-factor(classif2,levels=c("4","2","5","3","1"))
  mat<-table(cl1,cl2)
  print(mat)
```

```
   cl2
cl1  4  2  5  3  1
  1 43  0  0  1 12
  2  0 30  1  0  2
  3  0  0 20  9  0
```

```
  4  0  0  0 64  0
  5 14  1  0  0 40
```

```
percent5=sum(diag(mat))/sum(mat)
round(percent5,2)
```

```
[1] 0.83
```

```
rand5<-adjustedRandIndex(cl1,cl2)
round(rand5,2)
```

```
[1] 0.65
```

We get an 83% here for the match rate, and 0.65 here for ARI index, which is acceptable. Further, we test the stability for this method with 1 up to 6 clusters, and obtain the curve for match rate as well as ARI index below:

```
#match rate ARI Index
```

We can see that both the match rate and the ARI index, which also reveal the stability of the method, decrease monotonously as the number of clusters increases, and a rapid decrease occurs at 5. So, in order to increase the stability of our method, we turn to select 4 clusters as our final decision, and this still won't cause great loss in explained variance. The visualized result for 4 clusters case is as below:

```
#4 clusters
```

We have 4 very clear clusters here: The green part: Positive for the "organized" factor but negative for the "enjoy" factor. The customers who dislike shopping but are organized. The blue part: Negative for the "organized" factor and negative for the "enjoy" factor. The customers who dislike shopping and are not organized. The red part: Negative for the "organized" factor but positive for the "enjoy" factor. The customers who enjoy shopping but are not organized. The black part: Positive for the "organized" factor and positive for the "enjoy" factor. The customers who enjoy shopping and are organized.

This is also consistent with our initial guess from the dendrogram.

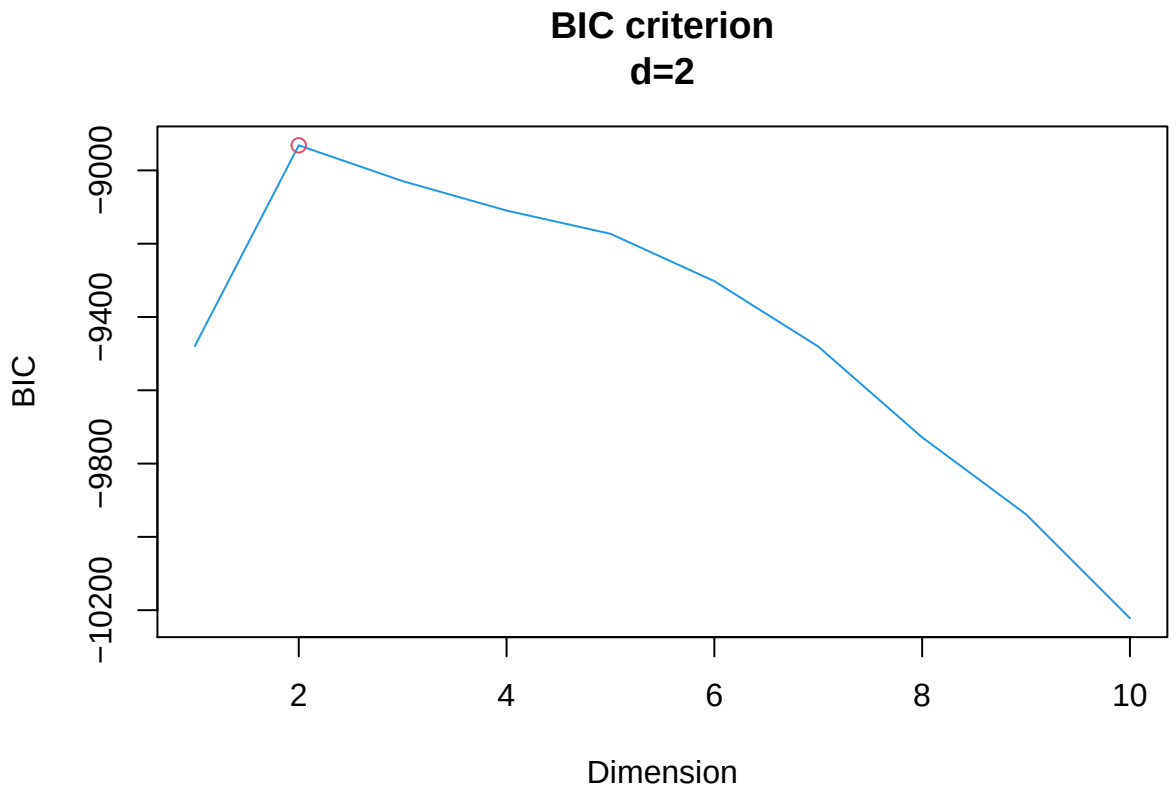
3.3.2 II) Model-based clustering with hddc()

Firstly, we let the hddc() function itself to select the optimal number of clusters.

```
#number of clusters chosen by hddc
#use BIC to select the number of dimensions
set.seed(0829539)
hddc1.out<-hddc(shopping,K=1:6,model="all")
hddc1.out
```

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: ABQD
  Posterior probabilities of groups
       1      2     3     4     5
   0.155 0.0637 0.194 0.315 0.272
      Intrinsic dimensions of the classes:
       1 2 3 4 5
  dim: 2 2 2 2 2

A: 1.09

B: 0.434
BIC:  -13426.7
```

```
plot(hddc1.out)
```



We can see that the hddc() function has chosen 5 clusters, and the optimal dimension to get the highest BIC value here is 2, which is consistent with our previous results from principal component analysis. Then we continue to get the solutions with 1 up to 6 clusters.

Then we continue to get the solutions with 1 up to 6 clusters.

```r
#fit all models with K=1
set.seed(0829539)
hddc1.out<-hddc(shopping,K=1,model="all")
hddc1.out
```

3.3.2.1 a) 1 cluster

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: AKJBKQKDK
  Posterior probabilities of groups
   1
   1
      Intrinsic dimensions of the classes:
       1
  dim: 2

Class   a1   a2
    1 3.75 2.27
        1
Bk: 0.551
BIC:  -13842.64
```

```r
#fit all models with K=2
set.seed(0829539)
hddc2.out<-hddc(shopping,K=2,model="all")
hddc2.out
```

3.3.2.2 b) 2 clusters

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: ABKQKD
  Posterior probabilities of groups
       1     2
   0.297 0.703
      Intrinsic dimensions of the classes:
       1 2
  dim: 2 2

A: 2.69
        1     2
Bk: 0.626 0.398
BIC:  -13630.49
```

```r
#fit all models with K=3
set.seed(0829539)
hddc3.out<-hddc(shopping,K=3,model="all")
hddc3.out
```

3.3.2.3 c) 3 clusters

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: AKBKQKD
  Posterior probabilities of groups
       1     2     3
   0.159 0.449 0.392
      Intrinsic dimensions of the classes:
       1 2 3
  dim: 2 2 2
       1    2    3
Ak: 1.71 1.17 1.74
        1     2     3
Bk: 0.456 0.375 0.518
BIC:  -13521.93
```

```r
#fit all models with K=4
set.seed(0829539)
hddc4.out<-hddc(shopping,K=4,model="all")
hddc4.out
```

3.3.2.4 d) 4 clusters

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: AJBQD
  Posterior probabilities of groups
       1      2     3     4
   0.463 0.0641 0.156 0.317
      Intrinsic dimensions of the classes:
       1 2 3 4
  dim: 2 2 2 2
      a1   a2
Aj: 1.26 0.98

B: 0.458
BIC:  -13429.53
```

```r
#fit all models with K=5
set.seed(0829539)
hddc5.out<-hddc(shopping,K=5,model="all")
hddc5.out
```

3.3.2.5 e) 5 clusters

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: ABQD
  Posterior probabilities of groups
       1      2     3     4     5
   0.155 0.0639 0.244 0.222 0.315
      Intrinsic dimensions of the classes:
       1 2 3 4 5
  dim: 2 2 2 2 2

A: 1.09

B: 0.435
BIC:  -13426.96
```

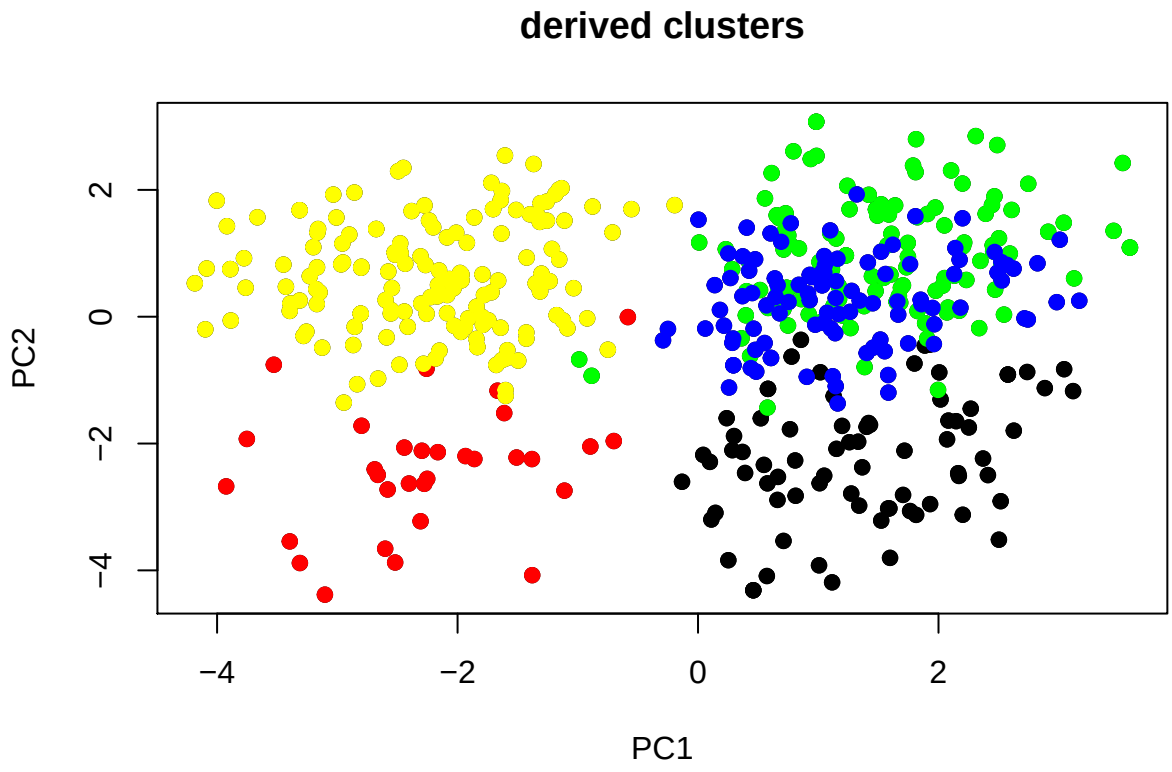
```r
set.seed(0829539)
hddc6.out<-hddc(shopping,K=6,model="all")
hddc6.out
```

3.3.2.6 f) 6 clusters

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: AJBQD
  Posterior probabilities of groups
        1     2     3    4     5     6
   0.0516 0.146 0.058 0.14 0.167 0.437
      Intrinsic dimensions of the classes:
       1 2 3 4 5 6
  dim: 2 2 2 2 2 2
      a1    a2
Aj: 1.22 0.875

B: 0.425
BIC:  -13395.17
```

Reasonably, we believe in the hddc() function and choose 5 clusters here as our initial solution. We visualize it in the space of the first two principal components as below:

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[hddc5.out$class==1,],col="black",pch=19)
points(comp[hddc5.out$class==2,],col="red",pch=19)
points(comp[hddc5.out$class==3,],col="green",pch=19)
points(comp[hddc5.out$class==4,],col="blue",pch=19)
points(comp[hddc5.out$class==5,],col="yellow",pch=19)
```



Hopefully, we can interpret in the same way as we have done for the kmeans method with 5 clusters, but here the green part and the blue part are almost overlapped, which is not very good for the interpretation. We hold on and test the stability of hddc() method with 5 clusters.

```
# cluster train data hddc
set.seed(0829539)
hddcT.out<-hddc(train,K=5,model="all")
hddcT.out
```

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: ABQD
  Posterior probabilities of groups
       1     2      3     4     5
  0.0723 0.454 0.0836 0.287 0.104
```

```
      Intrinsic dimensions of the classes:
       1 2 3 4 5
  dim: 2 2 2 2 2

A: 1.06

B: 0.447
BIC:  -7088.109
```

```
# cluster validation data hddc
set.seed(0829539)
hddcV.out<-hddc(valid,K=5,model="all")
hddcV.out
```

```
HIGH DIMENSIONAL DATA CLUSTERING
MODEL: ABQD
  Posterior probabilities of groups
       1     2     3     4     5
   0.309 0.092 0.134 0.277 0.188
      Intrinsic dimensions of the classes:
       1 2 3 4 5
  dim: 2 2 2 2 2

A: 0.989

B: 0.447
BIC:  -6742.602
```

```
classif1<-clusters(valid, hddcT.out[["mu"]])
classif2<-hddcV.out$class
table(classif1,classif2)
```

```
        classif2
classif1  1  2  3  4  5
       1  7 18  0  0  0
       2  0  0  1 66 41
       3  0  2 15  0  2
       4 66  1  0  0  0
       5  0  1 16  1  0
```

```
cl1<-as.factor(classif1)
cl2<-factor(classif2,levels=c("2","4","5","1","3"))
mat<-table(cl1,cl2)
#print(mat)
kbl(mat)
```

2	4	5	1	3
18	0	0	7	0
0	66	41	0	1
2	0	2	0	15
1	0	0	66	0
1	1	0	0	16

```
percent5=sum(diag(mat))/sum(mat)
round(percent5,2)
```

```
[1] 0.71
```

```
rand5<-adjustedRandIndex(cl1,cl2)
round(rand5,2)
```

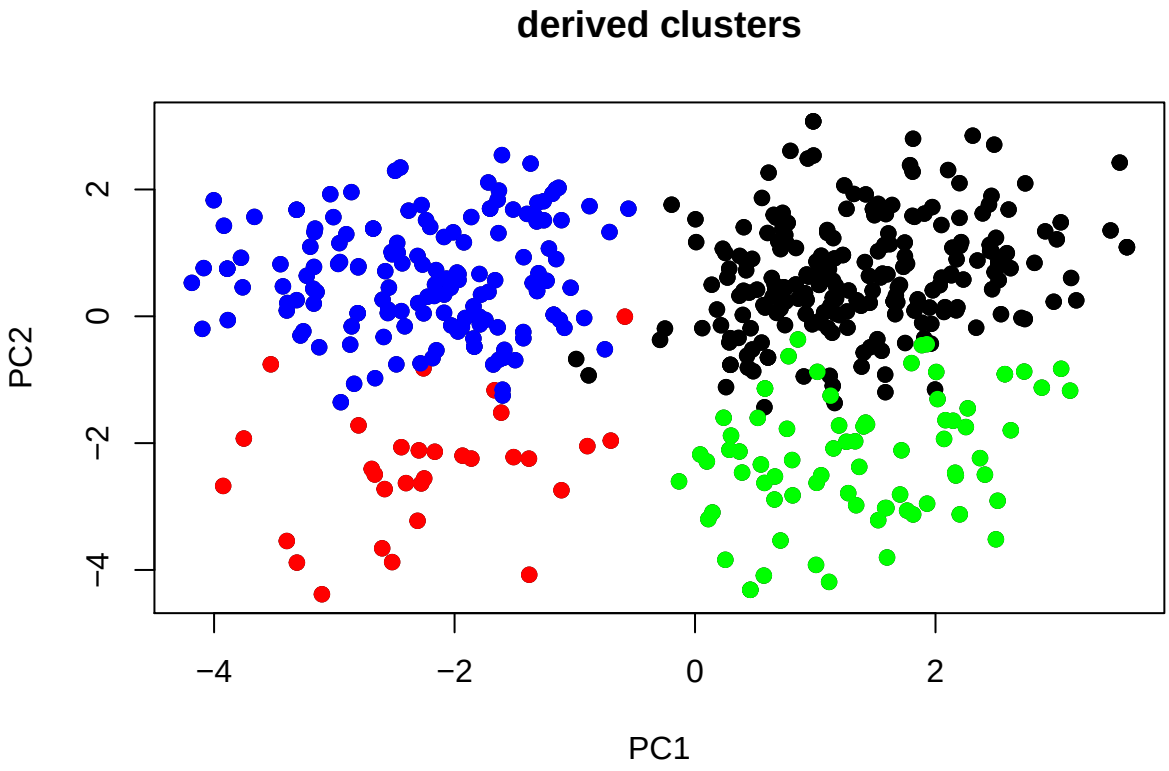
```
[1] 0.63
```

The match rate here is 71% and the ARI index here is 0.63, which are not quite good but still acceptable. However, if we continue to test the stability for this method with 1 up to 6 clusters, and obtain the curve for match rate as well as ARI index, we can see that:

```
#ari match
```

A sharp decrease occurs at 5, for both match rate and ARI index, and at 4 we get the maximal value for both match rate and ARI index, thus the most stable. We take a look at the solution with 4 clusters to check if we can get a result easy to interpret.

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[hddc4.out$class==1,],col="black",pch=19)
points(comp[hddc4.out$class==2,],col="red",pch=19)
points(comp[hddc4.out$class==3,],col="green",pch=19)
points(comp[hddc4.out$class==4,],col="blue",pch=19)
```



The borders are clear now and we can give an explicit interpretation for this figure. The customers are classified into four clusters according to our two principal components as below: The blue part: Positive for the "organized" factor but negative for the "enjoy" factor. The customers who dislike shopping but are organized. The red part: Negative for the "organized" factor and negative for the "enjoy" factor. The customers who dislike shopping and are not organized. The green part: Negative for the "organized" factor but positive for the "enjoy" factor. The customers who enjoy shopping but are not organized. The black part: Positive for the "organized" factor and positive for the "enjoy" factor. The customers who enjoy shopping and are organized. So finally, we take 4 clusters as our final decision.

3.3.3 III) Model-based clustering using Mclust()

Like hddc(), we firstly let the function Mclust() itself to choose an optimal number of clusters.

```
#number of clusters chosen by Mclust
set.seed(0829539)
mclust1.out<-Mclust(shopping,G=1:6)
summary(mclust1.out)
```

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
```

```
Mclust VVE (ellipsoidal, equal orientation) model with 4 components:

 log-likelihood   n  df       BIC       ICL
      -5996.532 487 146 -12896.55 -12942.22

Clustering table:
  1   2   3   4
 79 115 160 133
```

Here Mclust() has chosen 4 as the optimal number of clusters. Now we continue to obtain the solutions with 1 up to 6 clusters.

```
#fit all models with G=1
set.seed(0829539)
mclust1.out<-Mclust(shopping,G=1)
summary(mclust1.out)
```

3.3.3.1 a) 1 cluster

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust XXX (ellipsoidal multivariate normal) model with 1 component:

 log-likelihood   n df       BIC       ICL
       -6532.38 487 77 -13541.26 -13541.26

Clustering table:
  1
487
```

```
#fit all models with G=2
set.seed(0829539)
mclust2.out<-Mclust(shopping,G=2)
summary(mclust2.out)
```

3.3.3.2 b) 2 clusters

```
----------------------------------------------------
```

```
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust EVE (ellipsoidal, equal volume and orientation) model with 2 components:

 log-likelihood   n df       BIC       ICL
      -6376.903 487 99 -13366.44 -13392.97

Clustering table:
  1   2
321 166
```

```
#fit all models with G=3
set.seed(0829539)
mclust3.out<-Mclust(shopping,G=3)
summary(mclust3.out)
```

3.3.3.3 c) 3 clusters

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VVE (ellipsoidal, equal orientation) model with 3 components:

 log-likelihood   n  df       BIC       ICL
      -6152.089 487 123 -13065.33 -13093.78

Clustering table:
  1   2   3
190 160 137
```

```
#fit all models with G=4
set.seed(0829539)
mclust4.out<-Mclust(shopping,G=4)
summary(mclust4.out)
```

3.3.3.4 d) 4 clusters

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------
```

```
Mclust VVE (ellipsoidal, equal orientation) model with 4 components:

 log-likelihood   n  df       BIC       ICL
      -5996.532 487 146 -12896.55 -12942.22

Clustering table:
  1   2   3   4
 79 115 160 133
```

```
#fit all models with G=5
set.seed(0829539)
mclust5.out<-Mclust(shopping,G=5)
summary(mclust5.out)
```

3.3.3.5 e) 5 clusters

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VVE (ellipsoidal, equal orientation) model with 5 components:

 log-likelihood   n  df       BIC       ICL
      -5936.425 487 169 -12918.67 -12968.95

Clustering table:
  1   2   3   4   5
 72  55 155  73 132
```

```
#fit all models with G=6
set.seed(0829539)
mclust6.out<-Mclust(shopping,G=6)
summary(mclust6.out)
```

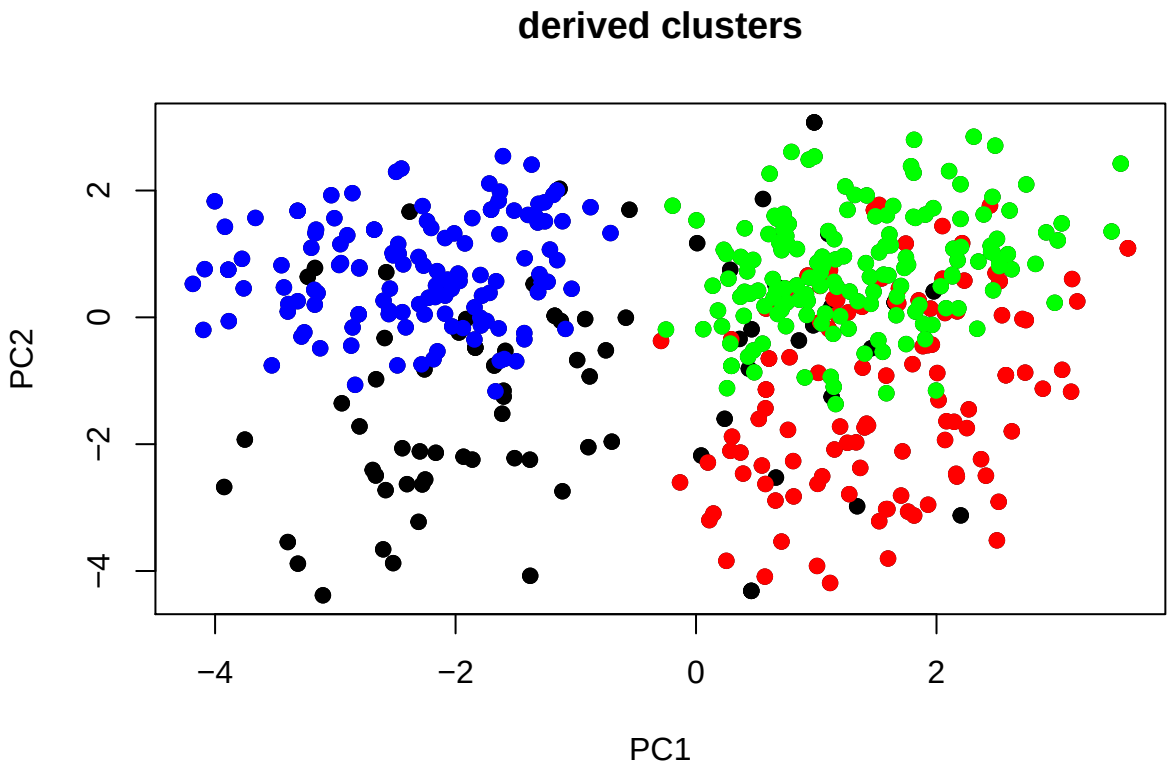
3.3.3.6 f) 6 clusters

```
----------------------------------------------------
Gaussian finite mixture model fitted by EM algorithm
----------------------------------------------------

Mclust VVE (ellipsoidal, equal orientation) model with 6 components:
```

```
 log-likelihood   n  df       BIC       ICL
      -5895.384 487 192 -12978.92 -13041.74

Clustering table:
  1   2   3   4   5   6
 71  53 122  68 132  41
```

We start with believing that Mclust() function will give us the optimal number of clusters, and visualize our result for Mclust() method with 4 clusters in the space of the first two principal components as below:

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[mclust4.out$class==1,],col="black",pch=19)
points(comp[mclust4.out$class==2,],col="red",pch=19)
points(comp[mclust4.out$class==3,],col="green",pch=19)
points(comp[mclust4.out$class==4,],col="blue",pch=19)
```

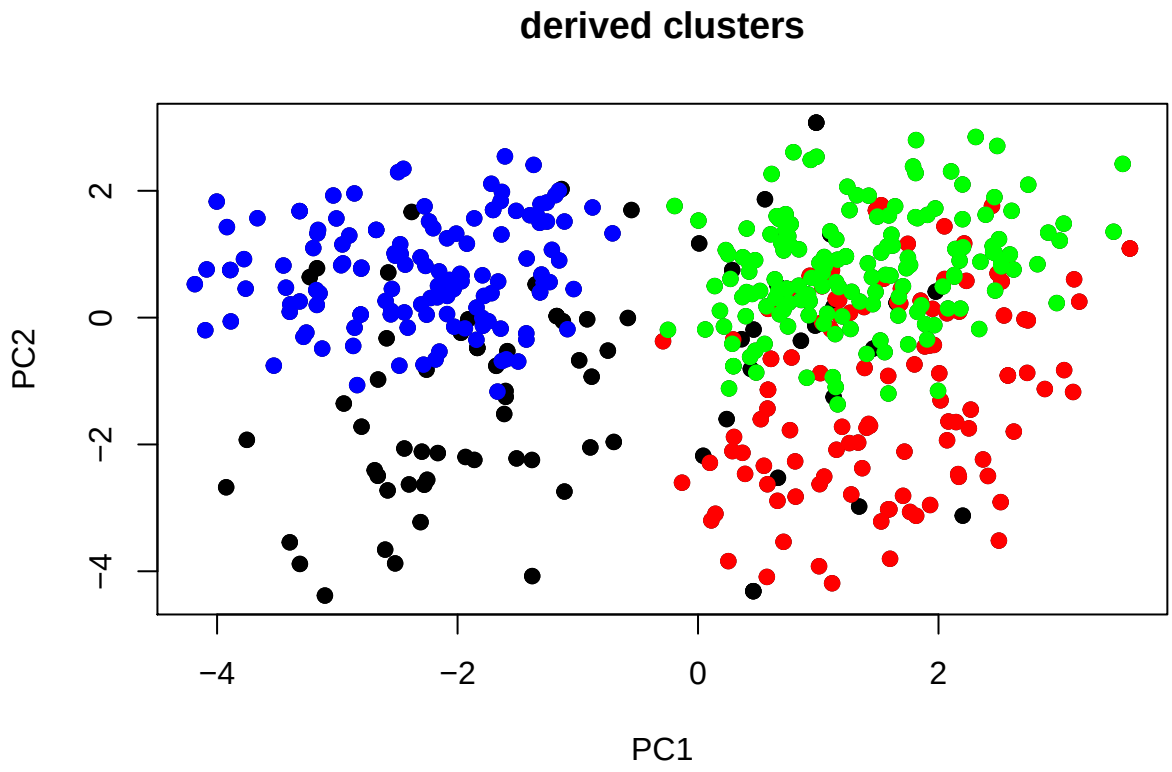


Actually, the black points here are very divergent, but its centroid is still located in the left-bottom corner. Actually, if we also take the visualized result with other numbers of clusters, we can see that:

6 clusters

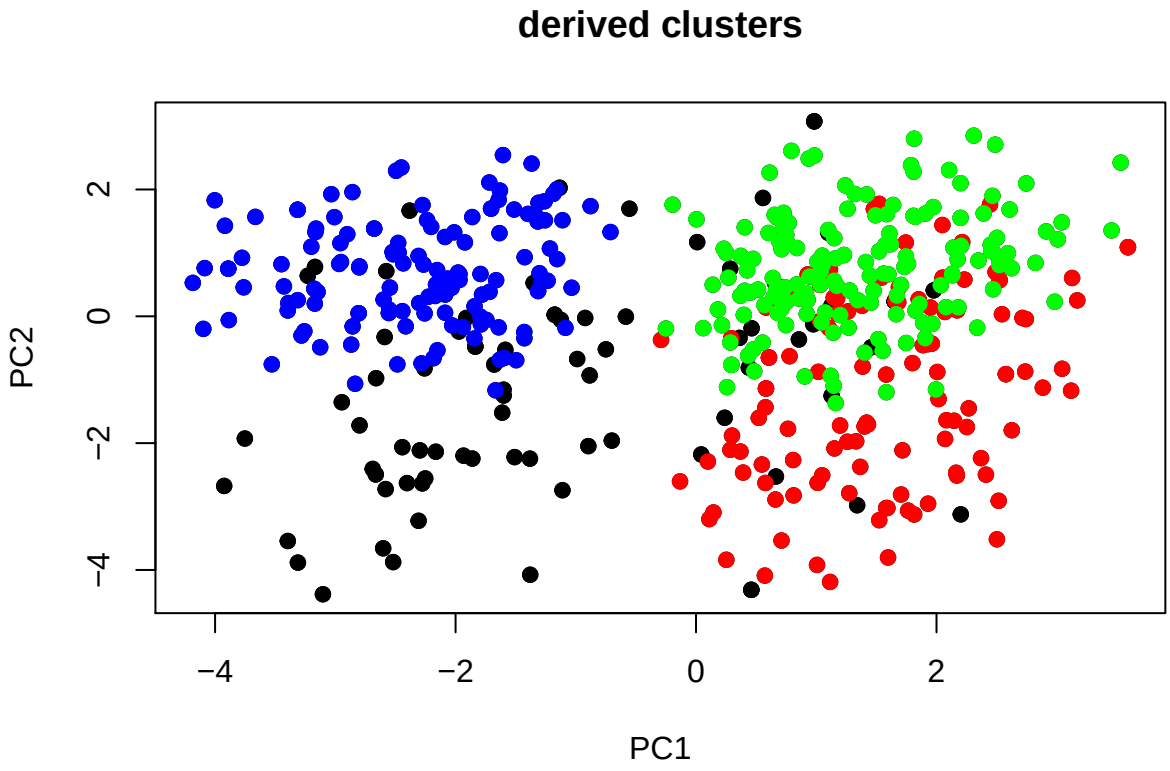
5 clusters

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[mclust4.out$class==1,],col="black",pch=19)
points(comp[mclust4.out$class==2,],col="red",pch=19)
points(comp[mclust4.out$class==3,],col="green",pch=19)
points(comp[mclust4.out$class==4,],col="blue",pch=19)
```



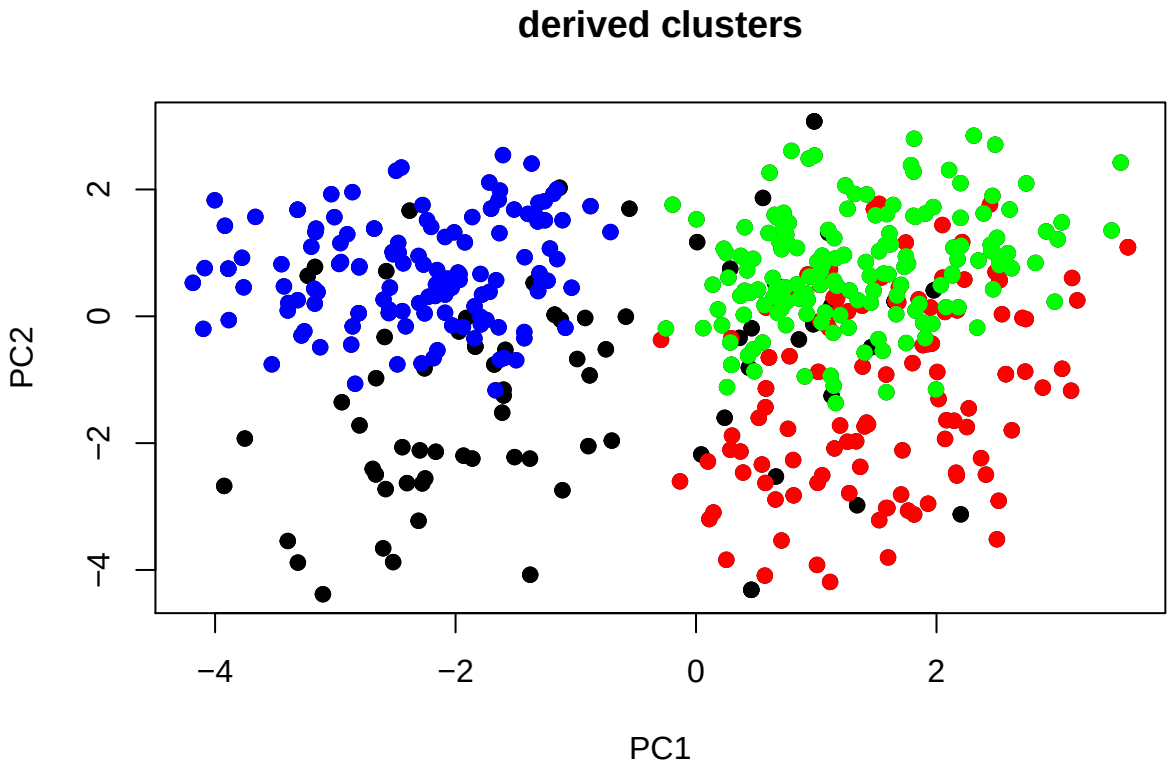
3 clusters

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[mclust4.out$class==1,],col="black",pch=19)
points(comp[mclust4.out$class==2,],col="red",pch=19)
points(comp[mclust4.out$class==3,],col="green",pch=19)
points(comp[mclust4.out$class==4,],col="blue",pch=19)
```



2 clusters

```
#plot clusters extracted with HDDC in space of first two principal components
plot(comp,main="derived clusters")
points(comp[mclust4.out$class==1,],col="black",pch=19)
points(comp[mclust4.out$class==2,],col="red",pch=19)
points(comp[mclust4.out$class==3,],col="green",pch=19)
points(comp[mclust4.out$class==4,],col="blue",pch=19)
```



For the cases with 6 clusters and 5 clusters, different parts are totally mixed up with each other, which is too hard to interpret, and for the cases with 3 and even 2 clusters, also we can't avoid the black dots to be divergent. So, 4 clusters seem to be the best choice here, and we can roughly interpret the result as: The blue part: Positive for the "organized" factor but negative for the "enjoy" factor. The customers who dislike shopping but are organized.

The red part: Negative for the "organized" factor and negative for the "enjoy" factor. The customers who dislike shopping and are not organized. The green part: Negative for the "organized" factor but positive for the "enjoy" factor. The customers who enjoy shopping but are not organized. The black part: Positive for the "organized" factor and positive for the "enjoy" factor. The customers who enjoy shopping and are organized.

Then we take a look at the stability.

```
# cluster validation data Mclust
set.seed(0829539)
#mclustV.out<-Mclust(valid,G=4)
#summary(mclustV.out)


#classif1<-clusters(valid, t(as.matrix(mclustT.out$parameters$mean)))
#classif2<-mclustV.out$class
#table(classif1,classif2)
```

```
#cl1<-as.factor(classif1)
#cl2<-factor(classif2,levels=c("3","4","2","1"))
#mat<-table(cl1,cl2)
#print(mat)

#percent4=sum(diag(mat))/sum(mat)
#round(percent4,2)

#rand4<-adjustedRandIndex(cl1,cl2)
#round(rand4,2)
```

We can see that here the match rate is 73% and the ARI index is 0.66, which are both acceptable. If we again continue to test the stability for this method with 1 up to 6 clusters, and obtain the curve for match rate as well as ARI index, we can see that:

```
#plot
#plot(c(2:6), c(percent2,percent3,percent4,percent5,percent6))
#lines(c(2:6), c(percent2,percent3,percent4,percent5,percent6))

#plot
#plot(c(2:6), c(rand2,rand3,rand4,rand5,rand6))
#lines(c(2:6), c(rand2,rand3,rand4,rand5,rand6))
```

4 clusters are already the most stable case, and we take it as our final decision.

3.4 4. Conclusion

As conclusion, we end up choosing 4 clusters for all these three methods and obtained the same interpretation with our two principal components “enjoy” and “organized”. With analysis for all these three methods, we can conclude that, Mclust() method here performs better than hddc() method in stability, but the boundaries of the clusters obtained by hddc() method are clearer. The hierarchical clustering with Ward’s method on squared Euclidean distances followed by k-means with the centroid of the hierarchical clustering as starting point is the most stable method here for this task.