Simone Travaglini

SEVENTH ASSIGNMENT

```
setwd("C:/Users/trava/OneDrive/Desktop/università/Daps&co/10-Big data analytics/")

getwd()

library(quanteda)

library(readtext)

library(glmnet)

library(xgboost)

library(Ckmeans.1d.dp)

library(dplyr)

library(ggplot2)

library(cowplot)

library(PerformanceAnalytics)

library(cvTools)

library(caret)

library(reshape2)

# Training-set

x <- read.csv("train_review23.csv", stringsAsFactors=FALSE)

str(x)

x$Sentiment <- factor(x$Sentiment,  levels=c("neg", "pos"), labels=c("Negative", "Positive"))

myCorpusTwitterTrain <- corpus(x)

tok2 <- tokens(myCorpusTwitterTrain , remove_punct = TRUE, remove_numbers=TRUE, remove_symbols = TRUE, split_hyphens = TRUE,
remove_separators = TRUE, remove_url = TRUE)

tok2 <- tokens_remove(tok2, stopwords("en"))

#remove symbols

tok2 <- tokens_remove(tok2, c("0*", "#*"))

tok2 <- tokens_wordstem (tok2)

Dfm_train <- dfm(tok2)

# trim the dfm

Dfm_train <- dfm_trim(Dfm_train , min_docfreq = 2, verbose=TRUE)

Dfm_train <- dfm_trim(Dfm_train, min_termfreq = 0.80, termfreq_type = "quantile",

                      max_docfreq = 0.20, docfreq_type = "prop")

Dfm_train  <- dfm_remove(Dfm_train , min_nchar = 2)

topfeatures(Dfm_train , 20)

train <- as(Dfm_train, "dgCMatrix")
```

I create a text corpus, myCorpusTwitterTrain, tokenize it, and perform essential text cleaning – removing punctuation, numbers, symbols, and applying stemming. Stopwords are also eliminated.

Turning the tokenized text into a Document-Feature Matrix (DFM), I trim it to remove low-frequency terms and further refine based on term and document frequency. Short terms are excluded.

Displaying the top 20 features provides insights, and the final step involves converting the DFM into a sparse matrix named train for machine learning model training.

#number of documents for each category

table(Dfm_train@docvars$Sentiment)

#proportion

prop.table(table(Dfm_train@docvars$Sentiment))

```
> table(Dfm_train@docvars$Sentiment)

Negative Positive
     233      262
> #proportion
> prop.table(table(Dfm_train@docvars$Sentiment))

 Negative  Positive
0.4707071 0.5292929
>
```

47% of the documents are categorized as negative, while 52% are positive.

then, I did the same for the validation dataset:

```
# Validation-set

x_val <- read.csv("validation_review23.csv", stringsAsFactors=FALSE)

str(x_val)

x_val$Sentiment <- factor(x_val$Sentiment,  levels=c("neg", "pos"), labels=c("Negative", "Positive"))

myCorpusVal <- corpus(x_val)

tok2 <- tokens(myCorpusVal , remove_punct = TRUE, remove_numbers=TRUE, remove_symbols = TRUE, split_hyphens = TRUE, remove_separators = TRUE)

tok2 <- tokens_remove(tok2, stopwords("en"))

tok2 <- tokens_remove(tok2, c("0*", "#*"))

tok2 <- tokens_wordstem (tok2)

Dfm_trainVal <- dfm(tok2)

Dfm_trainVal <- dfm_trim(Dfm_trainVal , min_docfreq = 2, verbose=TRUE)

Dfm_trainVal <- dfm_trim(Dfm_trainVal, min_termfreq = 0.80, termfreq_type = "quantile",

                  max_docfreq = 0.20, docfreq_type = "prop")

Dfm_trainVal <- dfm_remove(Dfm_trainVal , min_nchar = 2)

setequal(featnames(Dfm_train), featnames(Dfm_trainVal )) #FALSE

val_dfm  <- dfm_match(Dfm_trainVal , features = featnames(Dfm_train))

setequal(featnames(Dfm_train), featnames(val_dfm )) #TRUE

ValM <- as(val_dfm , "dgCMatrix")
```

… And for the test set:

```
x10 = read.csv("test_review23.csv", stringsAsFactors=FALSE)

str(x10)

nrow(x10)

myCorpusTwitterTest = corpus(x10)

tok <- tokens(myCorpusTwitterTest , remove_punct = TRUE, remove_numbers=TRUE, remove_symbols = TRUE, split_hyphens = TRUE,
remove_separators = TRUE, remove_URL = TRUE)

tok <- tokens_remove(tok, stopwords("en"))

tok <- tokens_remove(tok, c("0*"))

tok <- tokens_wordstem (tok)

Dfm_test <-dfm(tok)

Dfm_test <- dfm_trim(Dfm_test, min_docfreq = 2, verbose=TRUE)

Dfm_test <- dfm_remove(Dfm_test, min_nchar = 2)

Dfm_test <- dfm_trim(Dfm_test, min_termfreq = 0.80, termfreq_type = "quantile",

            max_docfreq = 0.2, docfreq_type = "prop")
```

Last I reduced the dfm in order to make them identical to the training set:

train and test-set:

```
setequal(featnames(Dfm_train), featnames(Dfm_test)) #FALSE

test_dfm  <- dfm_match(Dfm_test, features = featnames(Dfm_train))

setequal(featnames(Dfm_train), featnames(test_dfm )) #TRUE
```

train and validation set:

```
setequal(featnames(Dfm_train), featnames(Dfm_trainVal )) #FALSE

val_dfm  <- dfm_match(Dfm_trainVal , features = featnames(Dfm_train))

setequal(featnames(Dfm_train), featnames(val_dfm )) #TRUE
```

PART TWO: EXTERNAL VALIDITY

CROSS-VALIDATION:

first of all i started my analysis from the external validity, in order to asses which model is more accurate for my data:

In this script, I'm using a custom cross-validation function (Function_CV2XG) for training an XGBoost model.

I Converted Dfm_train to a sparse matrix (ttt) for modeling, I set the number of folds (k) for cross-validation to 5 and ensured reproducibility by setting a seed with set.seed(123).

GRADIENT BOOSTING MODEL:

```
# let's import the cross validation function for xgboost

source("Function_CV2XG.R")

Function_CV2XG

ttt <- as(Dfm_train, "dgCMatrix")#train matrix
```

```
data2 <- data.frame()# empty dataframe
k <- 5 #number of folds
set.seed(123) #for replicability
folds <- cvFolds(NROW(ttt), K=k)
str(folds)
class(Dfm_train@docvars$Sentiment)#it's a factor
y <- Dfm_train@docvars$Sentiment
y
y2 <- as.numeric(y)
y2
y2[ y2 ==1 ] <-0
y2[ y2 ==2 ] <-1
table(y2)
table(y)
Function_CV2XG(input=ttt, dt=data2, k=5, DV=y2, ML=xgboost)
XGBoost_res <- Function_CV2XG(ttt, data2, 5, y2, xgboost)
```

REGULARIZED REGRESSION:

Then, I did the same with the regularized regression:

in the script here below I'm examining the cross-validation folds using str(folds) and checking the number of observations per fold with table(folds$which) (each fold has 99 observations).

Next, I'm training regularized regression models:

Ridge regression (alpha=0) using cv.glmnet, and results are in Ridge_res.

Lasso regression (alpha=1) with complete regularization, results stored in Lasso_res.

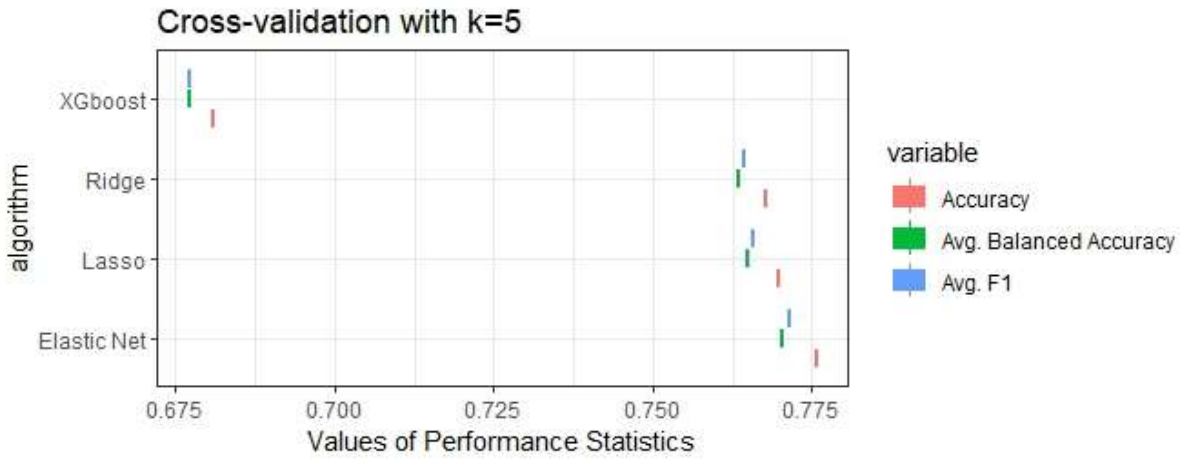
Elastic Net regression (alpha=0.5), with results in Elastic_res.

```
# let's import the CV function for the regularized regression with glmnet
source("Function_CV2Glmnet.R")
Function_CV2Glmnet
str(folds)
table(folds$which)#number of observation for each folds (99)
Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet, alpha=0, foldid=folds$which)
#ridge type alpha=0
Ridge_res <- Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet, alpha=0, foldid=folds$which)
Lasso_res <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=1)
Elastic_res <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=0.5)
```

I summarized the results of different models in my script. After calculating the column means for each model's performance metrics (XGBoost, Ridge, Lasso, and Elastic Net), I organized the information into a

dataframe called result. From the plot, it's evident that the Elastic Net regularized regression perform a little better than the other models, making it the best choice for this particular analysis.

```
#let's summarize the results to see which is the best model

result <- as.data.frame(colMeans(XGBoost_res[,c(1, 2, 3)]))

str(result)

result <- cbind(result, as.data.frame(colMeans(Ridge_res [,c(1, 2, 3)])))

result <- cbind(result, as.data.frame(colMeans(Lasso_res [,c(1, 2, 3)])))

result <- cbind(result, as.data.frame(colMeans(Elastic_res [,c(1, 2, 3)])))

str(result)

resultT <- as.data.frame(t(result))

str(resultT)

row.names(resultT)[1] = "XGboost"

row.names(resultT)[2] = "Ridge"

row.names(resultT)[3] = "Lasso"

row.names(resultT)[4] = "Elastic Net"

str(resultT)

resultT$algorithm <- row.names(resultT)

str(resultT)

df.long <- melt(resultT)

str(df.long)

ggplot(df.long,aes(algorithm,value,fill=variable, color = variable))+ geom_boxplot() + coord_flip() +

theme_bw()  + labs(title = "Cross-validation with k=5") + ylab(label="Values of Performance Statistics")

#the best model is the elastic net regularized regression
```



GRID SEARCH: LET'S TUNE THE HYPERPARAMETERS

In this script, I'm creating a grid of hyperparameters for XGBoost and then iterating through the combinations to conduct cross-validation. The results are stored in the dataframe dt, which includes mean cross-validated metrics and the corresponding hyperparameter values. I decided to set the range for eta

from 0.3 to 2, the max depth from 5 to 10 and the nrounds from 500 to 2000. This process allows for an analysis of how different hyperparameter configurations impact model performance.

```
#hyper parameter values: eta, max depth and nrounds
hyper_gridXG <- expand.grid(
  eta = c(0.3, 0.5, 1, 2),
  max_depth = c(5:10),
  nrounds  = c(500, 1000, 1500, 2000)
)
nrow(hyper_gridXG ) # 96 possibilities
dt <- data.frame()#empty dataframe

for(j in 1:nrow(hyper_gridXG )){
x <- Function_CV2XG(input=ttt, dt=dt, k=5, DV=y2, ML=xgboost,  eta = hyper_gridXG $eta [j], nrounds = hyper_gridXG $nrounds [j],
  max_depth = hyper_gridXG $ max_depth[j])
dt[j,1] <- mean(x[ , 1])
dt[j,2] <- mean(x[ , 2])
dt[j,3] <- mean(x[ , 3])
dt[j,4] <- hyper_gridXG $eta [j]
dt[j,5] <- hyper_gridXG $nrounds [j]
dt[j,6] <- hyper_gridXG $ max_depth[j]
colnames(dt)[1] <- "CV_Accuracy"
colnames(dt)[2] <- "CV_Avg. Balanced Accuracy"
colnames(dt)[3] <- "CV_Avg. F1"
colnames(dt)[4] <- "Eta"
colnames(dt)[5] <- "Nrounds"
colnames(dt)[6] <- "MaxDepth"
}
Dt
```

And those are the results:

```
head(arrange(dt, -CV_Accuracy ))
colMeans(XGBoost_res [ , c(1, 2, 3)])
```

	CV_Accuracy <dbl>	CV_Avg. Balanced Accuracy <dbl>	CV_Avg. F1 <dbl>	Eta <dbl>	Nrounds <dbl>	MaxDepth <int>
1	0.7050505	0.7045050	0.7029639	0.3	500	5
2	0.6929293	0.6947210	0.6918959	0.5	500	5
3	0.6828283	0.6800499	0.6799982	1.0	500	5
4	0.6808081	0.6770259	0.6772835	0.3	500	6
5	0.6804714	0.6808014	0.6779933	1.0	500	6
6	0.6686869	0.6656359	0.6655875	0.5	500	6

6 rows

Accuracy	Avg. Balanced Accuracy	Avg. F1
0.6808081	0.6770259	0.6772835

After, I set the best configuration for the model based on those results:

```
# the best configuration via grid search is eta=0.3        nrounds=500            and maxdepth=5

XGBoost_res2 <-Function_CV2XG(ttt, data2, 5, y2, xgboost, nrounds=500, max_depth=5, eta=0.3)

colMeans(XGBoost_res2[ , c(1, 2, 3)])
```

Accuracy	Avg. Balanced Accuracy	Avg. F1
0.7050505	0.7045050	0.7029639

REGULARIZED REGRESSION:

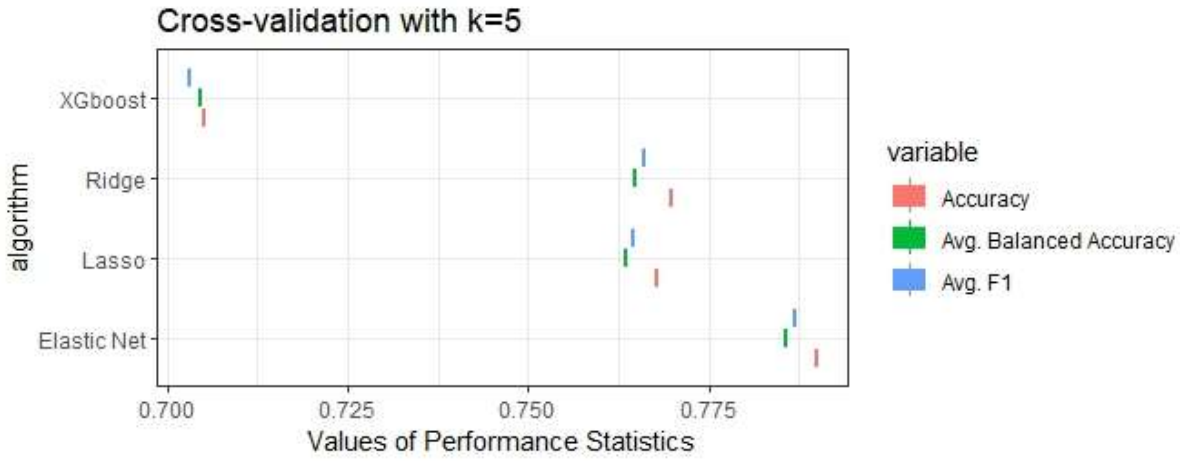
```
hyper_gridGlmnet <- expand.grid(

  alpha = seq(0, 1, by = 0.1)

)


hyper_gridGlmnet

dt <- data.frame()#empty dataframe

for(j in 1:nrow(hyper_gridGlmnet )){

x <- Function_CV2Glmnet(input=ttt, dt=data2, DV=y, ML=cv.glmnet, alpha=hyper_gridGlmnet $alpha[j], foldid=folds$which)

dt[j,1] <- mean(x[ , 1])

dt[j,2] <- mean(x[ , 2])

dt[j,3] <- mean(x[ , 3])

dt[j,4] <- hyper_gridGlmnet $alpha[j]

colnames(dt)[1] <- "CV_Accuracy"

colnames(dt)[2] <- "CV_Avg. Balanced Accuracy"

colnames(dt)[3] <- "CV_Avg. F1"

colnames(dt)[4] <- "Alpha"

}

dt

head(arrange(dt, -CV_Accuracy ))# the best model is alpha=0.2 (elastic)

Elastic_res2 <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=0.2)

Lasso_res2 <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=0)

Ridge_res2 <- Function_CV2Glmnet(ttt, data2, y, cv.glmnet, alpha=1)
```

	CV_Accuracy <dbl>	CV_Avg. Balanced Accuracy <dbl>	CV_Avg. F1 <dbl>	Alpha <dbl>
1	0.7898990	0.7856125	0.7868711	0.2
2	0.7878788	0.7808538	0.7818935	0.1
3	0.7858586	0.7796580	0.7808271	0.3
4	0.7858586	0.7803705	0.7816152	0.4
5	0.7757576	0.7703535	0.7714444	0.5
6	0.7696970	0.7643908	0.7653989	0.7

the optimal alpha value is 0.2

And again, I summarized the results in order to plot the accuracy level of all our model:

```
result <- as.data.frame(colMeans(XGBoost_res2[ , c(1, 2, 3)]))

str(result)

result <- cbind(result, as.data.frame(colMeans(Ridge_res2 [ , c(1, 2, 3)])))

result <- cbind(result, as.data.frame(colMeans(Lasso_res2 [ , c(1, 2, 3)])))

result <- cbind(result, as.data.frame(colMeans(Elastic_res2 [ , c(1, 2, 3)])))

str(result)

resultT <- as.data.frame(t(result))

str(resultT)

row.names(resultT)[1] = "XGboost"

row.names(resultT)[2] = "Ridge"

row.names(resultT)[3] = "Lasso"

row.names(resultT)[4] = "Elastic Net"

str(resultT)

resultT$algorithm <- row.names(resultT)

str(resultT)


df.long <- melt(resultT)

str(df.long)

ggplot(df.long,aes(algorithm,value,fill=variable, color = variable))+ geom_boxplot() + coord_flip() +

theme_bw()  + labs(title = "Cross-validation with k=5") + ylab(label="Values of Performance Statistics")
```



INTERNAL VALIDITY:

Let's explore the internal-validity of the elastic net RR:

Let's now move to elastic nets regression model

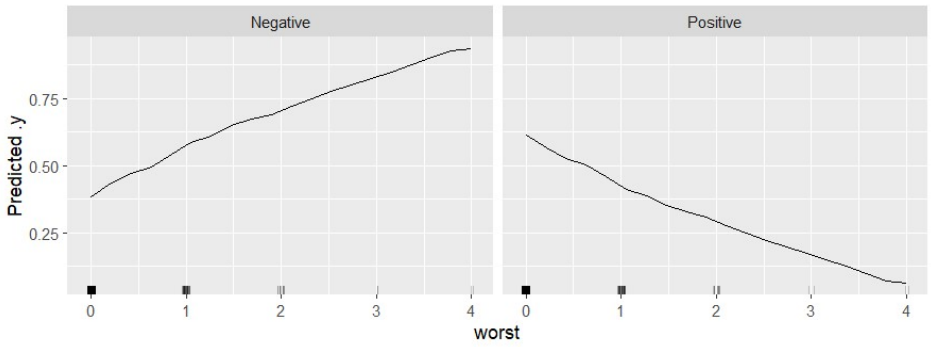
```
# clean the features's name
colnames(train) # matrix
colnames(train) <- make.names(colnames(train), unique = TRUE)
colnames(train)
# convert now the matrix into a data frame
trainDF <- as.data.frame(as.matrix(train))
class(trainDF)


# Let's estimate the model


set.seed(123)
system.time(elastic <-  cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,
                        family="binomial", alpha=0.2, nfolds=5, type.measure="class", trace.it=1 ))


newROLS <-  glmnet(y= Dfm_train@docvars$Sentiment, x=train,
                family="binomial", alpha=0.2, trace.it=1, lambda=elastic $lambda.min)
# predictive function:
predEl <-function(model, newdata){
  newData_x <- as.matrix(newdata)
  results<- predict(model, newData_x, type="class")
  return(results)
}
modEl<-  Predictor$new(newROLS, data = trainDF, y =Dfm_train@docvars$Sentiment, predict.fun = predElas)
```

```
modEl $predict(trainDF[1:5, ])

system.time({
 plan("callr", workers = 4)
 set.seed(123)
 impElastic <- FeatureImp$new(modEl , loss = "ce", n.repetitions=3)
})

head(impElastic$results[,c(1:4)],10)
```

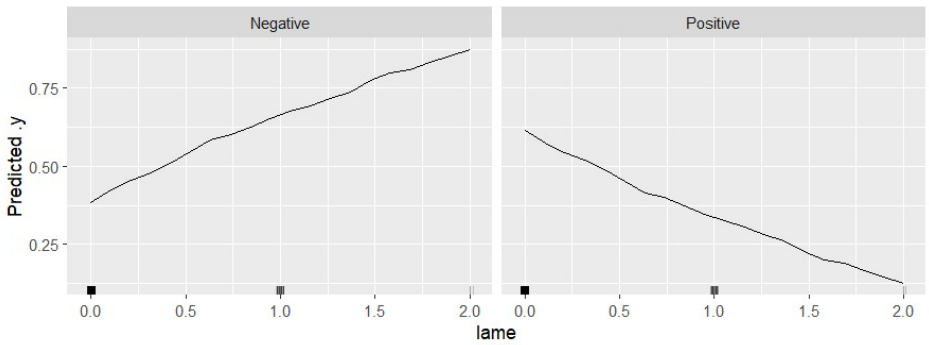
```
> head(impElastic$results[,c(1:4)],10)
   feature importance.05 importance importance.95
1   suppos      1.148214   1.196429      1.212500
2     lame      1.050000   1.178571      1.226786
3    worst      1.110714   1.142857      1.175000
4    other      1.041071   1.089286      1.121429
5    score      1.055357   1.071429      1.087500
6     mess      1.021429   1.053571      1.053571
7    idiot      1.053571   1.053571      1.101786
8  perfect      1.037500   1.053571      1.101786
9  delight      1.021429   1.053571      1.069643
10    pure      1.037500   1.053571      1.101786
>
```

Here above we can see that the most important features in my model are suppose, lame, worst, and so on. Let's plot the compute partial dependence plot for some of the features:

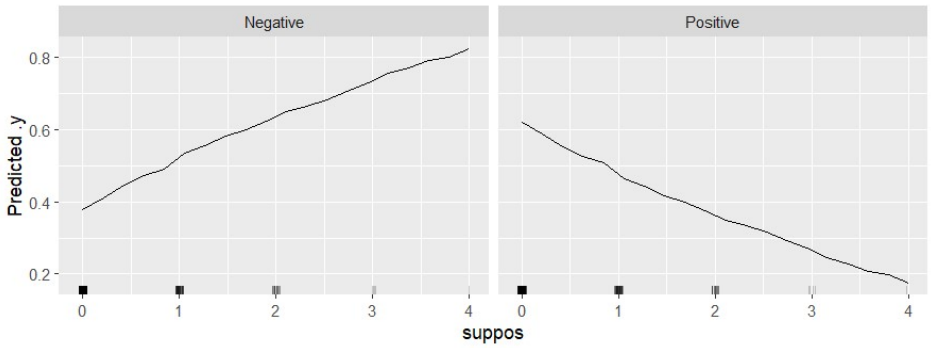
```
plot(FeatureEffect$new(modEla , "worst", method = "pdp"))
```



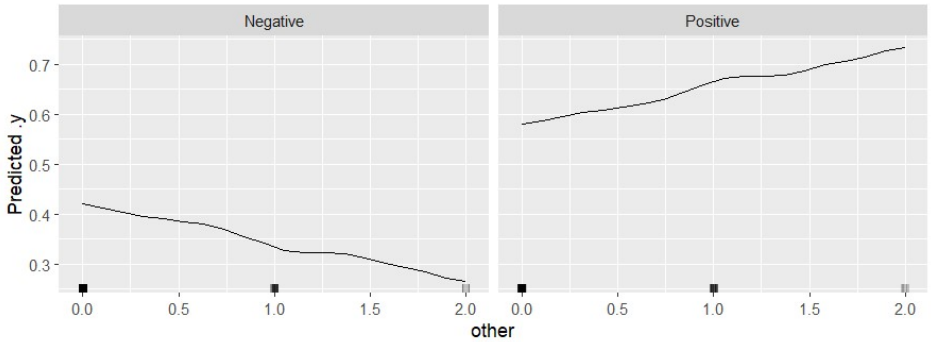

```
plot(FeatureEffect$new(modEla , "lame", method = "pdp"))
```



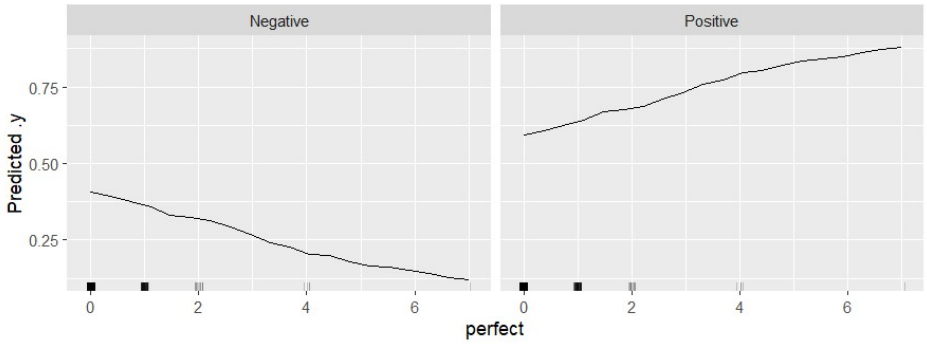

```
plot(FeatureEffect$new(modEla , "suppos", method = "pdp"))
```



plot(FeatureEffect$new(modEla , "other", method = "pdp"))



plot(FeatureEffect$new(modEla , "perfect", method = "pdp"))

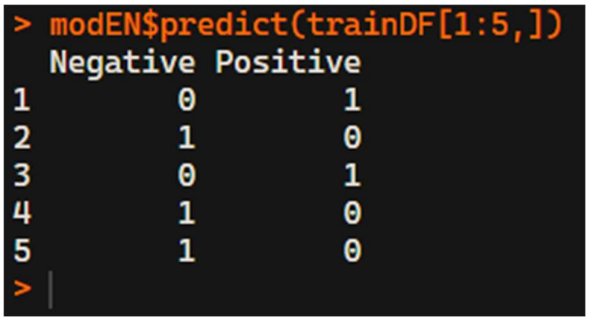


As we can see from these plots, the first three plots have a negative meaning, while the second two have a positive meaning. reading the words carefully, it seems to me that the directions make sense.

Last, I tried to predict which reviews was positive or negative:

modEN <- Predictor$new(elastic_net, data = trainDF, y =Dfm_train@docvars$Sentiment, predict.fun = predEN)

modEN$predict(trainDF[1:5,])



VALIDATION SET:

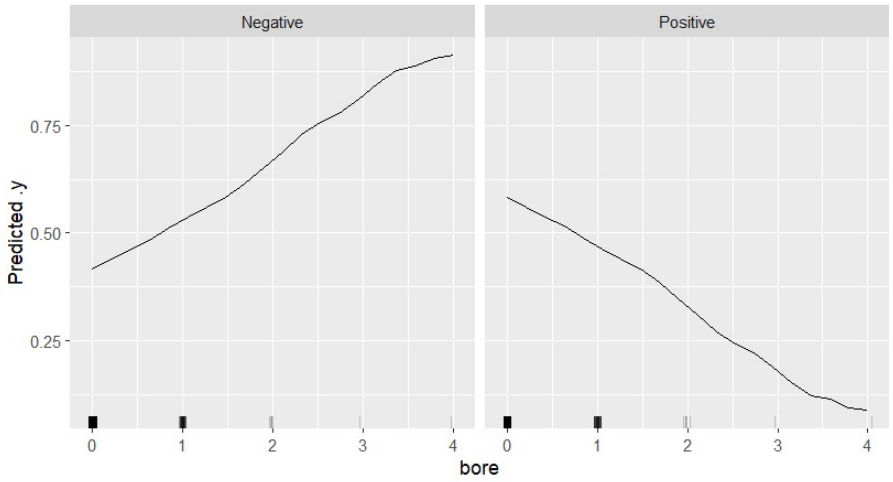
```
#validation

# let's clean the features's name

colnames(ValM)

colnames(ValM) <- make.names(colnames(ValM), unique = TRUE)

colnames(ValM)

# let's convert now the matrix into a data frame

valDF <- as.data.frame(as.matrix(ValM))

class(valDF)


# Let's restimate the Ridge on the "cleaned" matrix

set.seed(123)

system.time(elastic2 <-  cv.glmnet(y= Dfm_trainVal@docvars$Sentiment, x=ValM,

                        family="binomial", alpha=0.2, nfolds=5, type.measure="class"))


newROLS2 <-  glmnet(y= Dfm_trainVal@docvars$Sentiment, x=ValM,

              family="binomial", alpha=0.2, nfolds=5, lambda=elastic2$lambda.min)


# Then employ the below predictive function:

predEL <-function(model, newdata){

  newData_x <- as.matrix(newdata)

  results<- predict(model, newData_x, type="class")

  return(results)

}


system.time({

  plan("callr", workers = 6)

  set.seed(123)

  impElasticval<- FeatureImp$new(modEL2 , loss = "ce", n.repetitions=3)

})
```

head(impElasticval$results[,c(1:4)],10) #validation

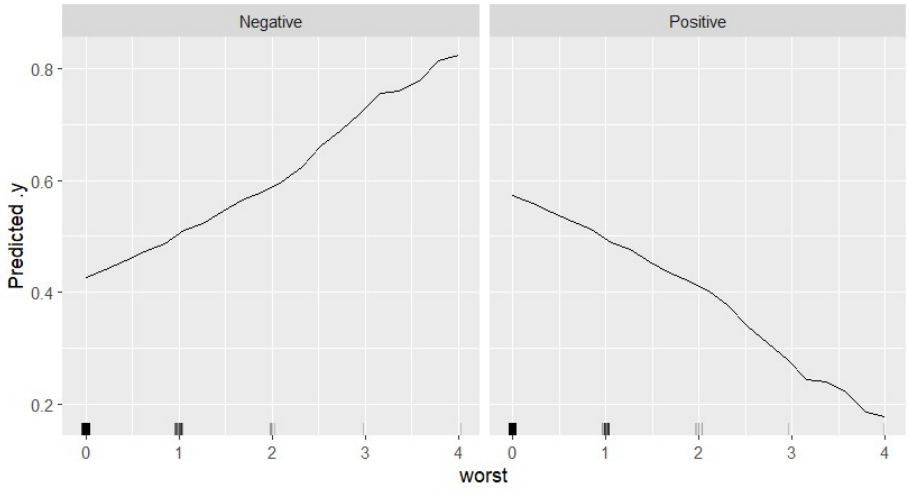
```
> head(impElasticval$results[,c(1:4)],10) # VALIDATION
    feature importance.05 importance importance.95
1      bore      1.179310   1.241379      1.334483
2      wast      1.082759   1.206897      1.268966
3   neither      1.075862   1.137931      1.200000
4    normal      1.103448   1.103448      1.103448
5    beauti      1.041379   1.103448      1.103448
6     worst      1.072414   1.103448      1.165517
7        aw      1.103448   1.103448      1.134483
8      amus      1.103448   1.103448      1.103448
9       edg      1.037931   1.068966      1.068966
10 american      1.037931   1.068966      1.068966
>
```

As you can see, there are no identical features between the validation set and the training set. this is good. the most important words are bore, wast, neither and so on.

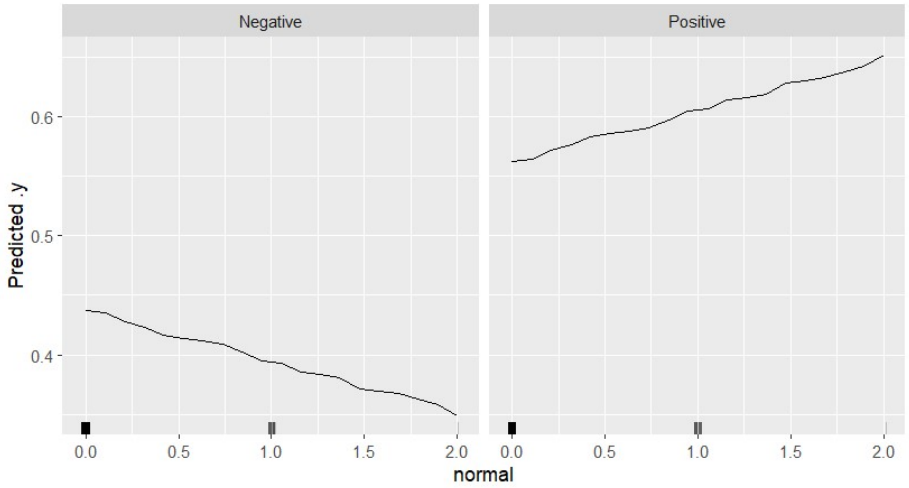
plot(FeatureEffect$new(modEla2,"bore", method = "pdp"))



plot(FeatureEffect$new(modEla2 , "worst", method = "pdp"))



plot(FeatureEffect$new(modEla2 , "normal", method = "pdp"))



Even in this case, I decide to plot the direction for three words: the first two are a clear indicator of a negative meaning, while the third is positive. reading the words carefully, it seems to me that the directions make sense.

Then I tried to display the prediction for the validation dataset, but there was this kind of error I was not able to solve:

modEL2 <- Predictor$new(newROLS2, data = valDF, y =as.factor(Dfm_trainVal@docvars$Sentiment), predict.fun = predEL)

modRid $predict(valDF[1:5,])

```
Errore in `contrasts<-`(`*tmp*`, value = contr.funs[1 + isOF[nn]]) :
  i contrasti si possono applicare solo a variabili factor con 2 o più livelli
>
```

TEST SET:

set.seed(123)

system.time(elastictest <- cv.glmnet(y= Dfm_train@docvars$Sentiment, x=train,

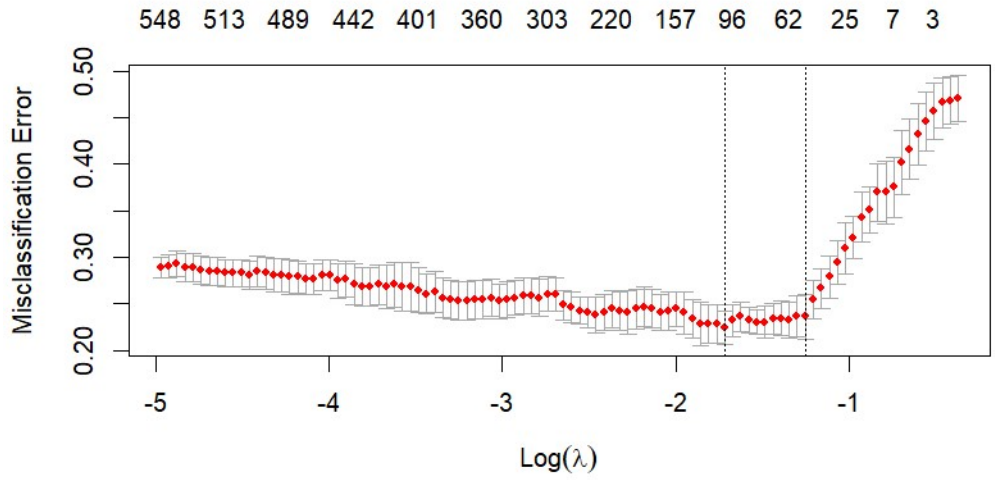
family="binomial", alpha=0.2, nfolds=5, trace.it=1,

type.measure="class"))

ROLS3 <- glmnet(y= Dfm_train@docvars$Sentiment, x=train,

family="binomial", alpha=0.2, trace.it=1, lambda=elastictest$lambda.min)

plot(elastictest)



The plot illustrates the minimum miss-classification error across the lambda values.

The first and second vertical dashed lines represent the lambda value with the minimum miss-classification error and the largest lambda value within one standard error of the minimum miss-classification error.

min(elastictest$cvm)

elastictest$lambda.min

log(elastictest$lambda.min)

elastictest$lambda.1se

log(elastictest$lambda.1se)

```
> min(elastictest$cvm)
[1] 0.2242424
> elastictest$lambda.min
[1] 0.1797671
> log(elastictest$lambda.min)
[1] -1.716093
> elastictest$lambda.1se
[1] 0.2862401
> log(elastictest$lambda.1se)
[1] -1.250924
>
```

And these are the results when I use lambda.min:

system.time(predicted<- predict(ROLS3, test, s = elastictest$lambda.min, type="class"))

table(predicted)

prop.table(table(predicted))

```
> table(predicted)
predicted
Negative Positive
     418      662
> prop.table(table(predicted))
predicted
Negative Positive
0.387037 0.612963
>
```

As can be seen from the output, in the test dataset 38% of the reviews are negative, while 61% are positive.

Last thing I performed was the analysis of the results I predict with the CV for elastic net:

system.time(pred_rr2 <- predict(elastictest , s= elastictest $lambda.min, test,type="class"))

table(pred_rr2)

prop.table(table(pred_rr2))

```
> table(pred_rr2 )
pred_rr2
Negative Positive
     418      662
> prop.table(table(pred_rr2))
pred_rr2
Negative Positive
0.387037 0.612963
>
```

And here the results when I use lambda.1se

system.time(predicted_glment2 <- predict(ROLS3, test, s = as.numeric(elastictest$lambda.1se) , type="class"))

table(predicted_glment2)

prop.table(table(predicted_glment2))

```
> table(predicted_glment2)
predicted_glment2
Negative Positive
     418      662
> prop.table(table(predicted_glment2))
predicted_glment2
Negative Positive
0.387037 0.612963
>
```

As you can se the results are the same.

Honestly, I also tried to perform this coefficient analysis, but I found some error I didn’t manage to solve

 # By focusing on the largest coefficient in absolute value computed by our model with the best lambda, we can have a sense of which are the most important features for our ridge model while being trained on the training-set. For exploring this issue, let's replicate our ridge model by specifying this time also the optimal lambda value via the "lambda" argument

newROLS <- glmnet(y= Dfm_train@docvars$Sentiment, x=train,

family="binomial", alpha=0.2, trace.it=1, lambda=elastictest$lambda.min)

newROLS

```
> newROLS

Call:  glmnet(x = train, y = Dfm_train@docvars$Sentiment, family = "binomial",
alpha = 0.2, lambda = elastictest$lambda.min, trace.it = 1)

   Df  %Dev Lambda
1 107 33.62 0.1798
>
```

coeff_df <- as.data.frame(Dfm_train@Dimnames$features)

str(coeff_df)

names(coeff_df)[1] <- "Feature"

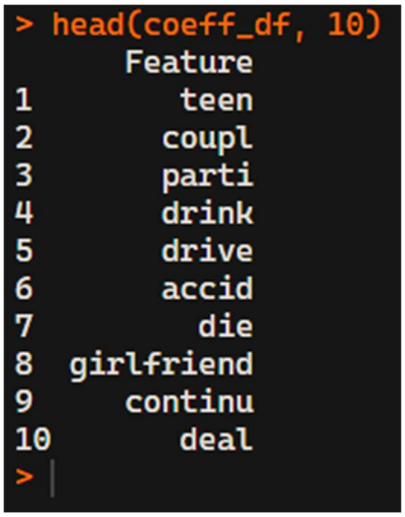
coeff_df$coeff <- newROLS$beta@x

```
> coeff_df$coeff ← newROLS$beta@x
Errore in `$←.data.frame`(`*tmp*`, coeff, value = c(0.0178423707610217,  :
  sostituzione con 107 righe, dati con 1438
>
```

str(coeff_df)

coeff_df<- coeff_df[order(-coeff_df$coef),]

head(coeff_df, 10)



the model recognizes the most important words, but I can't calculate the coefficient in this way. in fact the dataframe has only one column with the name of the features.

END