Spam Detection

Pietro Parenti

2025-05-20

Unprocessed DTM

The machine learning analyses will be conducted twice: once using an unsupervised document-term matrix, and a second time using an optimized DTM (see the Python code for details on the optimization method). At the end, the results from both approaches will be compared.

This first phase will therefore use the raw, unprocessed DTM.

Import and Descriptive Statistics

Import the Document Term Matrix

```
dtm <- read.csv("https://raw.githubusercontent.com/PietroParenti/yt-comments-classifier/main/dtm.csv")
dtm_original=dtm
```

```
dim(dtm_original)
```

```
## [1] 1902 4363
```

The DTM contains 1902 observations (comments) and 4302 columns, meaning that the comments include 4,302 unique words.

Response variable frequency

```
table(dtm_original$spam)
```

```
##
##   0   1
## 951 951
```

The response variable had been balanced during the previous analyses conducted in Python.

Below are the most frequent words in the document-term matrix.

```
# excluding spam
dtm_2 <- dtm_original[ , setdiff(names(dtm_original), "spam")]

# sum on every column
col_sums <- colSums(dtm_2)
```

```r
# sort
top5 <- sort(col_sums, decreasing = TRUE)[1:5]

# top 5
print(top5)
```

```
## this  the  and   to  you
##  711  702  618  617  581
```

```r
rm(col_max_sum,col_sums,top5,dtm_2)
```

Dimensional Reduction

The dataset contains too many variables: running even simple models requires significant computational effort. Therefore, I apply dimensionality reduction techniques to reduce the number of predictors.

Initial number of variables (features):

```r
ncol(dtm)
```

```
## [1] 4363
```

All words in the document term matrix that appear only once are removed.

```r
# excluding spam
X <- dtm[, setdiff(names(dtm), "spam")]

# sum on every column
col_sums <- colSums(X)

# keep only if sum > 1
X_filtered <- X[, col_sums > 1]
dtm <- cbind(X_filtered, spam = dtm$spam)

rm(X,X_filtered,col_sums)

ncol(dtm)
```

```
## [1] 1708
```

Have been removed 2617 words.

Principal Components Analisis

Principal Component Analysis is a dimensionality reduction technique that transforms correlated variables into a smaller set of uncorrelated components, capturing the most variance in the data.

```r
dtm.pc <- prcomp(dtm[, -which(names(dtm) == "spam")],
                 scale. = TRUE)
```

The scale.=TRUE option standardizes each variable so that it has a mean of 0 and a standard deviation of 1. This is important because PCA is highly sensitive to the scale of the variables.

In a document-term matrix, some words may have very high values (frequent terms), while others appear only rarely. Without scaling, the principal components would give disproportionate weight to high-frequency words, potentially distorting the analysis.

```
pcs <- dtm.pc$rotation

parole <- colnames(dtm)[colnames(dtm) != "spam"]

# extracting n words per pc
top_loadings <- function(pc, n = 5) {
  loadings_abs <- abs(pc)
  top_indices <- order(loadings_abs, decreasing = TRUE)[1:n]
  data.frame(
    Words = parole[top_indices],
    Loading = round(pc[top_indices], 4)
  )
}

# number of pcs to visualize
n_pc <- 3

top_words_list <- lapply(1:n_pc, function(i) {
  top_loadings(pcs[, i], n = 5)
})

names(top_words_list) <- paste0("PC", 1:n_pc)

top_words_table <- do.call(cbind, top_words_list)

print(top_words_table[,-1 ])
```

```
##             PC1.Loading PC2.Words PC2.Loading PC3.Words PC3.Loading
## X19255           0.1488      rand      0.1550        to      0.1132
## X229508          0.1488      paul      0.1548       you      0.1115
## X5337555197      0.1488    X1m00s      0.1539       and      0.1031
## X53481           0.1488   morning      0.1539       X35      0.0978
## X5575096797      0.1488   privacy      0.1539 X4netjobs      0.0978
```

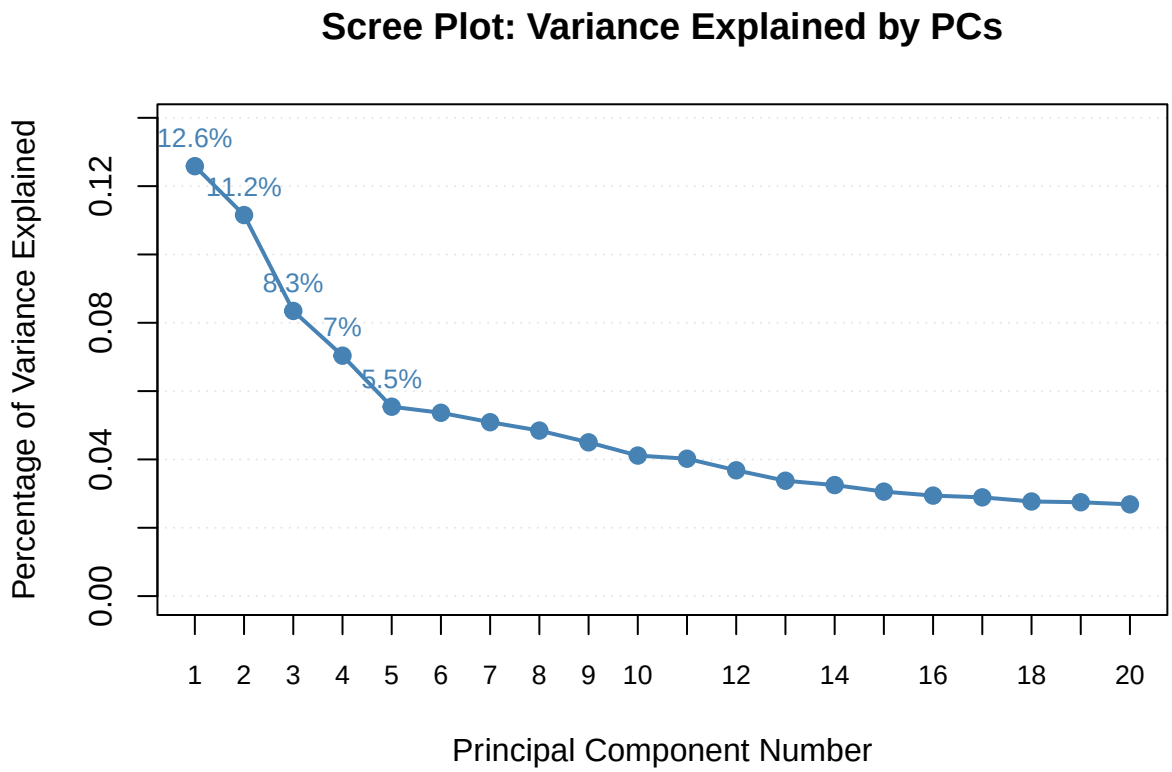
```
rm(n_pc,parole,top_loadings,top_words_list,top_words_table,pcs)
```

We can see above the words that contribute most to the first three principal components. Unfortunately, these components are hard to interpret. Therefore, in the following analyses, we will not provide a detailed interpretation of the significant variables in the models, as doing so would be difficult and potentially misleading.

Percentage of Xs variability explained by each principal component.

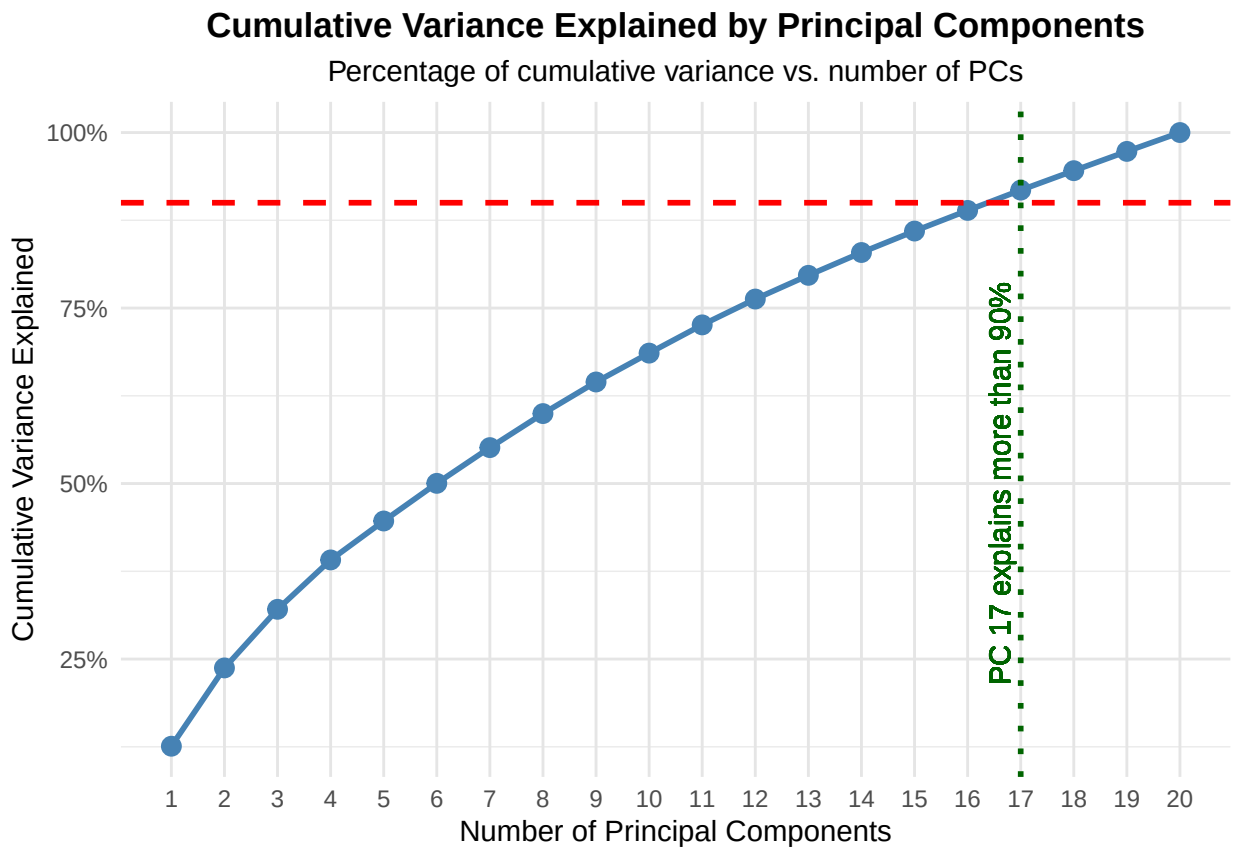
```
sd <- dtm.pc$sdev[1:20]
# Calculate explained variance for each PC
var_explained <- sd^2/sum(sd^2)
```

```r
# Create enhanced scree plot
plot(var_explained,
     type = "o",
     pch = 19,
     col = "steelblue",
     lwd = 2,
     ylab = "Percentage of Variance Explained",
     xlab = "Principal Component Number",
     main = "Scree Plot: Variance Explained by PCs",
     xaxt = "n",
     ylim = c(0, max(var_explained)*1.1),
     panel.first = grid(nx = NA, ny = NULL, col = "gray90", lty = "dotted"))

# Custom x-axis with all PC numbers
axis(1, at = 1:20, labels = 1:20, las = 1, cex.axis = 0.8)


# Add percentage labels for first few PCs
text(1:5, var_explained[1:5],
     labels = paste0(round(var_explained[1:5]*100, 1), "%"),
     pos = 3, cex = 0.8, col = "steelblue")
```



Cumulative plot

```r
# Calculate cumulative variance
cum_var <- cumsum(sd^2)/sum(sd^2)

# Find number of PCs needed to explain 90% of variance
n_pc_90 <- which.max(cum_var >= 0.9)

# Enhanced plot with ggplot2
library(ggplot2)
ggplot(data.frame(PC = 1:20, CumVar = cum_var), aes(x = PC, y = CumVar)) +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(color = "steelblue", size = 3) +
  geom_hline(yintercept = 0.9, color = "red", linewidth = 1, linetype = "dashed") +
  geom_vline(xintercept = n_pc_90, color = "darkgreen", linewidth = 1, linetype = "dotted") +
  geom_text(aes(x = n_pc_90, y = 0.5,
                label = paste("PC", n_pc_90, "explains more than 90%")),
            color = "darkgreen", angle = 90, vjust = -0.5) +
  scale_x_continuous(breaks = 1:20, minor_breaks = NULL) +
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  labs(title = "Cumulative Variance Explained by Principal Components",
       subtitle = "Percentage of cumulative variance vs. number of PCs",
       x = "Number of Principal Components",
       y = "Cumulative Variance Explained") +
  theme_minimal() +
  theme(panel.grid.major = element_line(color = "gray90"),
        plot.title = element_text(face = "bold", hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
```




```
rm(cum_var,elbow_point,n_pc_90,sd,var_explained)
```

I follow the (unwritten) rule of selecting the smallest number of principal components that explain at least 90% of the total variance. In this case, I select M = 17 components.

```
dtm.pc <- dtm.pc$x[,1:17]
dim(dtm.pc)
```

```
## [1] 1902   17
```

Adding spam column.

```
dtm.pc <- as.data.frame(dtm.pc)
dtm.pc <- cbind(dtm.pc,dtm$spam)
names(dtm.pc)[names(dtm.pc) == "dtm$spam"] <- "spam"
dim(dtm.pc)
```

```
## [1] 1902   18
```

Train and Test

The dataset is split into training and test sets: 70% for training and 30% for testing.

The training and test sets are also balanced based on the spam variable. After splitting the data, a check is performed to ensure that both subsets remain balanced.

```r
set.seed(123)

# searching for spam and non spam indexes
index_spam <- which(dtm.pc$spam == 1)
index_ham <- which(dtm.pc$spam == 0)

# sampling
train_spam <- sample(index_spam, size = 0.7 * length(index_spam))
train_ham <- sample(index_ham, size = 0.7 * length(index_ham))

train_index <- c(train_spam, train_ham)

dtm.pc.train <- dtm.pc[train_index, ]
dtm.pc.test  <- dtm.pc[-train_index, ]

# verifing
table(dtm.pc.train$spam) / nrow(dtm.pc.train)
```

```
##
##   0   1
## 0.5 0.5
```

```r
table(dtm.pc.test$spam) / nrow(dtm.pc.test)
```

```
##
##   0   1
## 0.5 0.5
```

```r
rm(index_ham,index_spam,train_ham,train_index,train_spam)
```

Code to save the datasets locally for use in Python analyses. Uncomment the chunk below to execute the saving process.

```r
#write.csv(dtm.pc.test, "C://Users//paren//Desktop//_MAGISTRALE//_Machine_Learning//progetto//dtm.pc.te

#write.csv(dtm.pc.train, "C://Users//paren//Desktop//_MAGISTRALE//_Machine_Learning//progetto//dtm.pc.t
```

Models evaluation

Now principal components extracted from the text data are used as input features to assess the classification accuracy of various machine learning models.

Each model is evaluated using the test error rate.

Linear Model

The first model evaluated is the linear model. However, it is not optimal for classification tasks—even binary ones—because it assumes a continuous response and can produce predicted values outside the [0,1] range, which are not interpretable as probabilities.

```r
model <- lm(spam ~ ., data = dtm.pc.train)

summary(model)
```

```
##
## Call:
## lm(formula = spam ~ ., data = dtm.pc.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.1553 -0.4373 -0.4194  0.5378  0.6633
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.4992709  0.0131798  37.881  < 2e-16 ***
## PC1          0.0028223  0.0016708   1.689  0.09142 .
## PC2          0.0090674  0.0031729   2.858  0.00433 **
## PC3          0.0232737  0.0033131   7.025 3.44e-12 ***
## PC4          0.0066549  0.0033113   2.010  0.04466 *
## PC5          0.0019534  0.0033176   0.589  0.55611
## PC6          0.0015777  0.0030405   0.519  0.60392
## PC7          0.0145530  0.0032556   4.470 8.49e-06 ***
## PC8         -0.0080197  0.0081346  -0.986  0.32438
## PC9          0.0039837  0.0090587   0.440  0.66018
## PC10        -0.0027032  0.0050722  -0.533  0.59416
## PC11        -0.0035161  0.0034351  -1.024  0.30621
## PC12         0.0027073  0.0031084   0.871  0.38394
## PC13         0.0009588  0.0044034   0.218  0.82767
## PC14         0.0038505  0.0034758   1.108  0.26816
## PC15         0.0115178  0.0042083   2.737  0.00629 **
## PC16        -0.0072502  0.0037104  -1.954  0.05091 .
## PC17        -0.0094850  0.0041060  -2.310  0.02104 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4777 on 1312 degrees of freedom
## Multiple R-squared:  0.09947,    Adjusted R-squared:  0.0878
## F-statistic: 8.525 on 17 and 1312 DF,  p-value: < 2.2e-16
```

The linear model fits the binary outcome spam using 17 principal components (PCs) as predictors. The overall model is statistically significant ($F = 8.525$, $p < 2.2e\text{-}16$), but the adjusted R^2 is relatively low (0.088), indicating limited explanatory power.

Some components (PC3 and PC7 in particular) show statistically significant associations with the response variable, suggesting they contribute to differentiating between spam and non-spam comments.

However, interpreting individual PCs is inherently difficult, as each component is a linear combination of thousands of original terms. The direction and magnitude of coefficients do not directly map back to specific words or features, limiting the model's interpretability despite statistical significance. This limitation applies to all models that follow—since they rely on the same transformed input space—but this comment will not be repeated to avoid redundancy.

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
pred <- ifelse(pred > 0.5, 1, 0)
```

Training error rate calculation

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3863636
```

All the training error rates are stored in a data frame to make easier future comparison.

```
TER <- data.frame(
  method = character(),
  training.error.rate = numeric(),
  stringsAsFactors = FALSE
)
```

```
TER <- rbind(TER, data.frame(method = "linear model", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model)
```

Logistic Model

The logistic model is the go-to choice for binary outcome prediction.

```
model <- glm(spam ~ ., data = dtm.pc.train, family = binomial)

summary(model)
```

```
##
## Call:
## glm(formula = spam ~ ., family = binomial, data = dtm.pc.train)
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.540639   0.195414   7.884 3.17e-15 ***
## PC1          2.831487   0.675800   4.190 2.79e-05 ***
## PC2          0.356983   0.252123   1.416 0.156803
## PC3          0.363370   0.114406   3.176 0.001493 **
## PC4          0.008321   0.169068   0.049 0.960748
## PC5          0.067152   0.047542   1.412 0.157805
## PC6          0.044952   0.087299   0.515 0.606610
## PC7          0.650312   0.158214   4.110 3.95e-05 ***
## PC8          0.310750   0.115181   2.698 0.006977 **
## PC9         -0.519193   0.121410  -4.276 1.90e-05 ***
## PC10         0.207208   0.120185   1.724 0.084695 .
## PC11        -0.749183   0.152201  -4.922 8.55e-07 ***
## PC12         0.168543   0.175110   0.962 0.335799
```

```
## PC13         0.357478   0.095373   3.748 0.000178 ***
## PC14         0.004102   0.059876   0.069 0.945375
## PC15         0.364491   0.098098   3.716 0.000203 ***
## PC16        -0.185218   0.043910  -4.218 2.46e-05 ***
## PC17        -0.269365   0.123601  -2.179 0.029309 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1843.8  on 1329  degrees of freedom
## Residual deviance: 1483.4  on 1312  degrees of freedom
## AIC: 1519.4
##
## Number of Fisher Scoring iterations: 9
```

The logistic regression model is statistically significant. Several PCs are highly significant predictors of the spam class.

```
pred <- predict(model, newdata = dtm.pc.test, type= "response")
```

```
pred <- ifelse(pred > 0.5, 1, 0)
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3409091
```

```
TER <- rbind(TER, data.frame(method = "logistic model", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model)
```

Linear Discriminant Analisis

The idea behind LDA is to find a linear combination of the predictor variables that best separates the different classes.

It is tighly connected with the idea of Bayes theorem: it assignes a unit to that class with maximum probability a posteriori. For K classes, we have:

- a likelihood part: a density function for X given that we are in a specific class f_k(X), for k=1,. . . ,K
- prior part: prior distribution pi(k) of class k (prior belief of prob of being in class k)

```
model <- MASS::lda(spam~.,
                   dtm.pc.train)

model
```

```
## Call:
## lda(spam ~ ., data = dtm.pc.train)
```

```
##
## Prior probabilities of groups:
##   0   1
## 0.5 0.5
##
## Group means:
##          PC1        PC2       PC3        PC4         PC5         PC6        PC7
## 0 -0.2380471 -0.5139051 -1.236197 -0.2577027 -0.06406643 -0.04519801 -0.4973509
## 1  0.3380658  0.3495661  1.369787  0.4783245  0.02451277 -0.11166944  0.5374443
##           PC8        PC9        PC10       PC11        PC12        PC13
## 0 -0.02592688  0.1884116 -0.06146389  0.1431529 -0.08683697  0.03089494
## 1  0.21476304 -0.3345043  0.16396834 -0.1294474  0.20484476 -0.06056277
##          PC14       PC15       PC16       PC17
## 0 -0.07647184 -0.2376900  0.1511326  0.2152452
## 1  0.06662259  0.2222481 -0.1279726 -0.2515417
##
## Coefficients of linear discriminants:
##               LD1
## PC1   0.018845817
## PC2   0.060546984
## PC3   0.155408842
## PC4   0.044437528
## PC5   0.013043399
## PC6   0.010535106
## PC7   0.097176568
## PC8  -0.053551085
## PC9   0.026600716
## PC10 -0.018050513
## PC11 -0.023478685
## PC12  0.018077755
## PC13  0.006402046
## PC14  0.025711239
## PC15  0.076909261
## PC16 -0.048412439
## PC17 -0.063335527
```

```
summary(model)
```

```
##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means   34     -none- numeric
## scaling 17     -none- numeric
## lev      2     -none- character
## svd      1     -none- numeric
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

The LDA output shows:

- prior probabilities: set to 0.5 for both classes, reflecting a balanced dataset

- group means: average values of each principal component (PC) per class, larger differences suggest stronger discriminative power
- coefficients of linear discriminants (LD1): weights used to form a linear combination of PCs that best separates the two classes. PCs with larger absolute values contribute more to class separation.

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
pred <- pred$class
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3863636
```

"Curiously, it turns out that the classifcations that we get if we use linear regression to predict a binary response will be the same as for the linear discriminant analysis (LDA)" An Introduction to Statistical Learning, page 132

```
TER <- rbind(TER, data.frame(method = "linear discriminant analisis", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model)
```

Quadratic Discriminant Analisis

```
model <- MASS::qda(spam~.,
                   dtm.pc.train)

model
```

```
## Call:
## qda(spam ~ ., data = dtm.pc.train)
##
## Prior probabilities of groups:
##   0   1
## 0.5 0.5
##
## Group means:
##          PC1        PC2       PC3        PC4         PC5         PC6        PC7
## 0 -0.2380471 -0.5139051 -1.236197 -0.2577027 -0.06406643 -0.04519801 -0.4973509
## 1  0.3380658  0.3495661  1.369787  0.4783245  0.02451277 -0.11166944  0.5374443
##           PC8        PC9        PC10       PC11        PC12        PC13
## 0 -0.02592688  0.1884116 -0.06146389  0.1431529 -0.08683697  0.03089494
## 1  0.21476304 -0.3345043  0.16396834 -0.1294474  0.20484476 -0.06056277
##          PC14       PC15       PC16       PC17
## 0 -0.07647184 -0.2376900  0.1511326  0.2152452
## 1  0.06662259  0.2222481 -0.1279726 -0.2515417
```

```r
summary(model)
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means    34    -none- numeric
## scaling 578    -none- numeric
## ldet      2    -none- numeric
## lev       2    -none- character
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

QDA differs from LDA by allowing each class to have its own covariance matrix. This added flexibility can improve performance when class variances differ.

```r
pred <- predict(model, newdata = dtm.pc.test)
```

```r
pred <- pred$class
```

```r
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3916084
```

```r
TER <- rbind(TER, data.frame(method = "quadratic discriminant analisis", training.error.rate = TER.mod))
```

```r
rm(pred,TER.mod,model)
```

Naive Bayes

Naive Bayes is a simple yet effective classification algorithm based on Bayes' Theorem, with the naive assumption that predictors (features) are conditionally independent given the class.

```r
model <- e1071::naiveBayes(spam~.,
                  dtm.pc.train)

model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   0   1
```

```
## 0.5 0.5
##
## Conditional probabilities:
##    PC1
## Y         [,1]       [,2]
##   0 -0.2380471  0.1210358
##   1  0.3380658 11.2112391
##
##    PC2
## Y         [,1]      [,2]
##   0 -0.5139051 0.5384527
##   1  0.3495661 6.0405160
##
##    PC3
## Y        [,1]     [,2]
##   0 -1.236197 1.444677
##   1  1.369787 7.101280
##
##    PC4
## Y         [,1]      [,2]
##   0 -0.2577027 0.7708406
##   1  0.4783245 6.3769894
##
##    PC5
## Y          [,1]      [,2]
##   0 -0.06406643 0.3963301
##   1  0.02451277 5.6605291
##
##    PC6
## Y          [,1]      [,2]
##   0 -0.04519801 0.4872713
##   1 -0.11166944 6.2443107
##
##    PC7
## Y         [,1]      [,2]
##   0 -0.4973509 0.5224336
##   1  0.5374443 5.9276883
##
##    PC8
## Y          [,1]      [,2]
##   0 -0.02592688 0.7493135
##   1  0.21476304 5.1655732
##
##    PC9
## Y         [,1]      [,2]
##   0  0.1884116 0.5508969
##   1 -0.3345043 4.5103591
##
##    PC10
## Y          [,1]      [,2]
##   0 -0.06146389 0.7981771
##   1  0.16396834 5.6792021
##
##    PC11
```

```
## Y         [,1]      [,2]
##   0  0.1431529 0.5082736
##   1 -0.1294474 5.5677385
##
##    PC12
## Y          [,1]      [,2]
##   0 -0.08683697 0.3978053
##   1  0.20484476 6.0383500
##
##    PC13
## Y          [,1]     [,2]
##   0  0.03089494 2.107415
##   1 -0.06056277 4.815228
##
##    PC14
## Y          [,1]      [,2]
##   0 -0.07647184 0.7500404
##   1  0.06662259 5.4170366
##
##    PC15
## Y         [,1]      [,2]
##   0 -0.2376900 0.6878232
##   1  0.2222481 4.4531536
##
##    PC16
## Y         [,1]     [,2]
##   0  0.1511326 0.865093
##   1 -0.1279726 4.979165
##
##    PC17
## Y         [,1]     [,2]
##   0  0.2152452 0.875717
##   1 -0.2515417 4.495409
```

```
summary(model)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    17     -none- list
## levels     2     -none- character
## isnumeric 17     -none- logical
## call       4     -none- call
```

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.4090909
```

```
TER <- rbind(TER, data.frame(method = "naive bayes", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model)
```

KNN

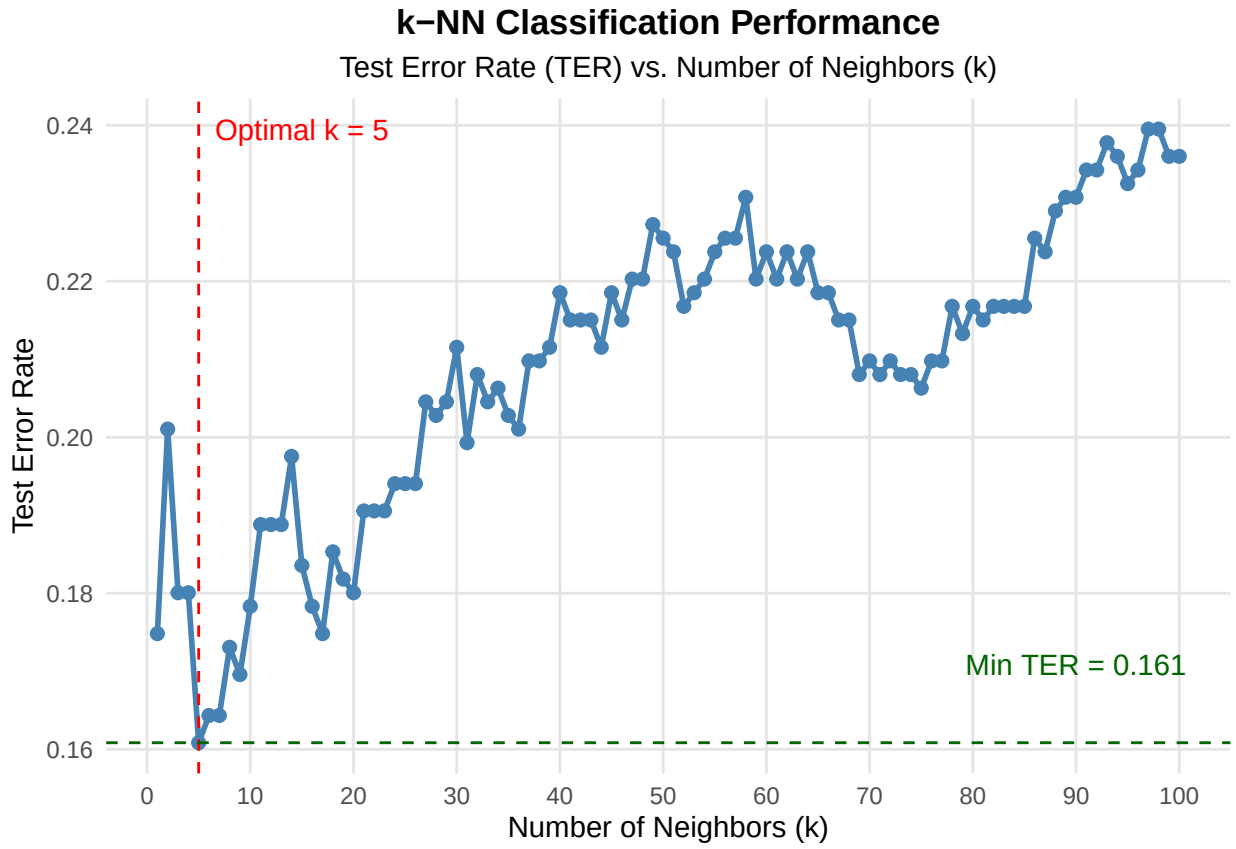
K nearest neighbours (KNN regression) is a non-parametric approach, based on on averages of Ys that are close to the evaluation point of interest.

This for loop performs KNN regression with k ranging from 1 to 100, evaluating the test error rate on the test dataset for each k to identify the value of k that minimizes the test error.

```
set.seed(123)
TER.models=c()

for (k in seq(1, 100, by = 1)) {
  pred <- class::knn(cbind(dtm.pc.train[, !(colnames(dtm.pc.train) == "spam")]),
                     cbind(dtm.pc.test[, !(colnames(dtm.pc.train) == "spam")]),
                     dtm.pc.train$spam,
                     k = k)
  TER.mod=mean(pred!=dtm.pc.test$spam)
  TER.models=c(TER.models,TER.mod)
}

TER.models=cbind(seq(1, 100, by = 1),TER.models)
```

```
library(ggplot2)

# Converti i risultati in un dataframe per ggplot
results_df <- data.frame(
  k = TER.models[, 1],
  TER = TER.models[, 2]
)

# Trova il k ottimale (TER minimo)
optimal_k <- results_df$k[which.min(results_df$TER)]
min_TER <- min(results_df$TER)

# Crea il grafico con ggplot2
ggplot(results_df, aes(x = k, y = TER)) +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(color = "steelblue", size = 2) +
  geom_vline(xintercept = optimal_k, linetype = "dashed", color = "red") +
  geom_hline(yintercept = min_TER, linetype = "dashed", color = "darkgreen") +
  annotate("text",
           x = optimal_k + 10,
           y = max(results_df$TER),
           label = paste("Optimal k =", optimal_k),
           color = "red") +
  annotate("text",
           x = 90,
           y = min_TER + 0.01,
```

```
          label = paste("Min TER =", round(min_TER, 3)),
          color = "darkgreen") +
  labs(title = "k-NN Classification Performance",
       subtitle = "Test Error Rate (TER) vs. Number of Neighbors (k)",
       x = "Number of Neighbors (k)",
       y = "Test Error Rate") +
  scale_x_continuous(breaks = seq(0, 100, by = 10)) +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```



The value of k that minimizes the TER is 5.

```
min(TER.models)
```

```
## [1] 0.1608392
```

```
#k=5
```

Rerunning the KNN regression with k = 5.

```r
set.seed(123)

pred <- class::knn(cbind(dtm.pc.train[, !(colnames(dtm.pc.train) == "spam")]),
                        cbind(dtm.pc.test[, !(colnames(dtm.pc.train) == "spam")]),
                        dtm.pc.train$spam,
                        k = optimal_k)
```

```r
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1625874
```

```r
TER <- rbind(TER, data.frame(method = "KNN", training.error.rate = TER.mod))
```

```r
rm(pred,TER.mod,k,TER.models,results_df,min_TER,optimal_k)
```

Ridge Regression

Ridge regression is a regularized version of linear regression that adds an L2 penalty term to the loss function. This penalty shrinks the regression coefficients towards zero, helping to reduce model complexity and multicollinearity.

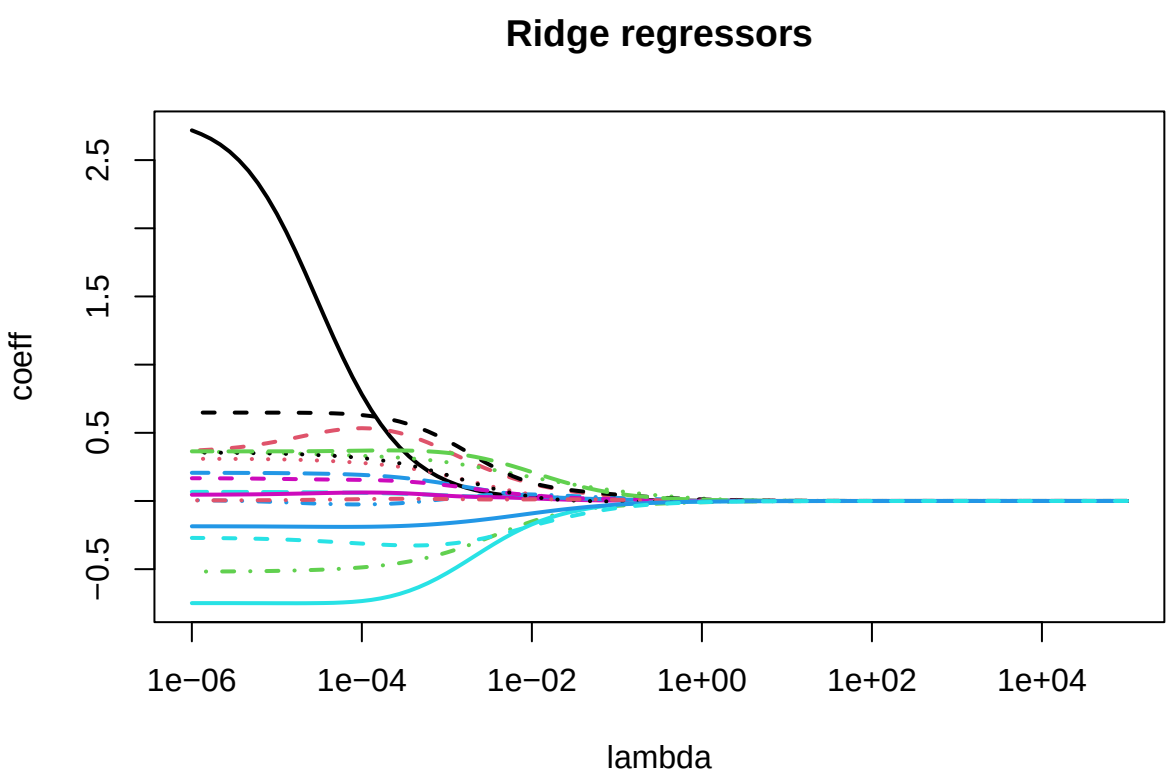
```r
set.seed(123)
lambda.grid <- 10^seq(-6,5,length=100)
# if lambda = 0 then ridge becomes lm
# if lambda --> Inf then all coeffs --> 0 since
# RSS impact is negligible, relative to the
# role played by the penalty
ridge.mod <- glmnet::glmnet(dtm.pc.train[, !(colnames(dtm.pc) == "spam")],
                            dtm.pc.train[,  (colnames(dtm.pc) == "spam")],
                            alpha=0,
                            lambda=lambda.grid,
                            family = "binomial")
dim(coef(ridge.mod)) #     18 coeffs (one intercept + 17pcs)
```

```
## [1]  18 100
```

```r
#                          x 100 lambdas
```

Choosing the best lambda

```r
matplot(ridge.mod$lambda,
        t(as.matrix(ridge.mod$beta)),
        type="l",log="x",lwd=2,
        main="Ridge regressors",
        xlab="lambda",
        ylab="coeff")
```



We use CV for finding the optimal lambda:

```
set.seed(123)

# CV for finding the optimal lambda:
# we can use cv.glmnet in the same package
# by default: 10-fold cv
cv.out <- glmnet::cv.glmnet(as.matrix(dtm.pc.train[, !(colnames(dtm.pc) == "spam")]),
                            as.numeric(dtm.pc.train[,  (colnames(dtm.pc) == "spam")]),
                    alpha=0,
                    lambda=lambda.grid,
                    family="binomial")

cv.out$lambda.min # lambda with min estimated test error
```

```
## [1] 2.154435e-06
```

The lambda of the simpliest model with error within 1 stad dev from minimun error:

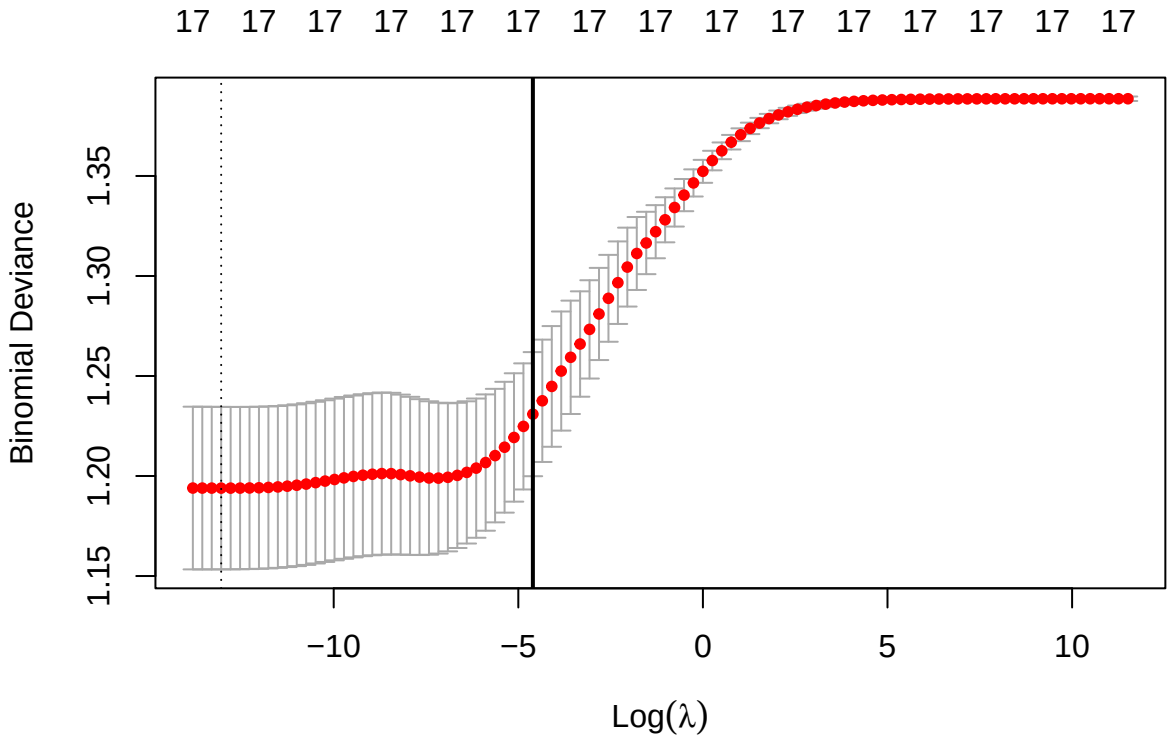
```
cv.out$lambda.1se # lambda of the simplest model
```

```
## [1] 0.01
```

```
# with error within 1 std from minumum error
```

Plot of the CV error: the two vertical lines are log(lambda.min) and log(lambda.1se)

```
# plot the CV error (-+ 1 std) as a function of lambda
plot(cv.out)
# the two verticle lines are log(lambda.min) and
# log(lambda.1se)
abline(v=log(cv.out$lambda.1se),lwd=2,col="black")
```




```
# Previsione delle probabilità sul training set
pred.prob <- predict(cv.out,
                     newx = as.matrix(dtm.pc.test[, !(colnames(dtm.pc) == "spam")]),
                     s = cv.out$lambda.1se,
                     type = "response")
```

```
# Classificazione binaria (cutoff 0.5)
pred <- ifelse(pred.prob > 0.5, 1, 0)

# Test Error Rate
TER.mod <- mean(pred != dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3479021
```

```
TER <- rbind(TER, data.frame(method = "ridge regression", training.error.rate = TER.mod))
```

```
rm(cv.out,model,ridge.mod,std.cof,X,lambda.grid,TER.mod,y,std.coeff,pred,pred.prob,lambda_values,cv_data
```

Lasso Regression

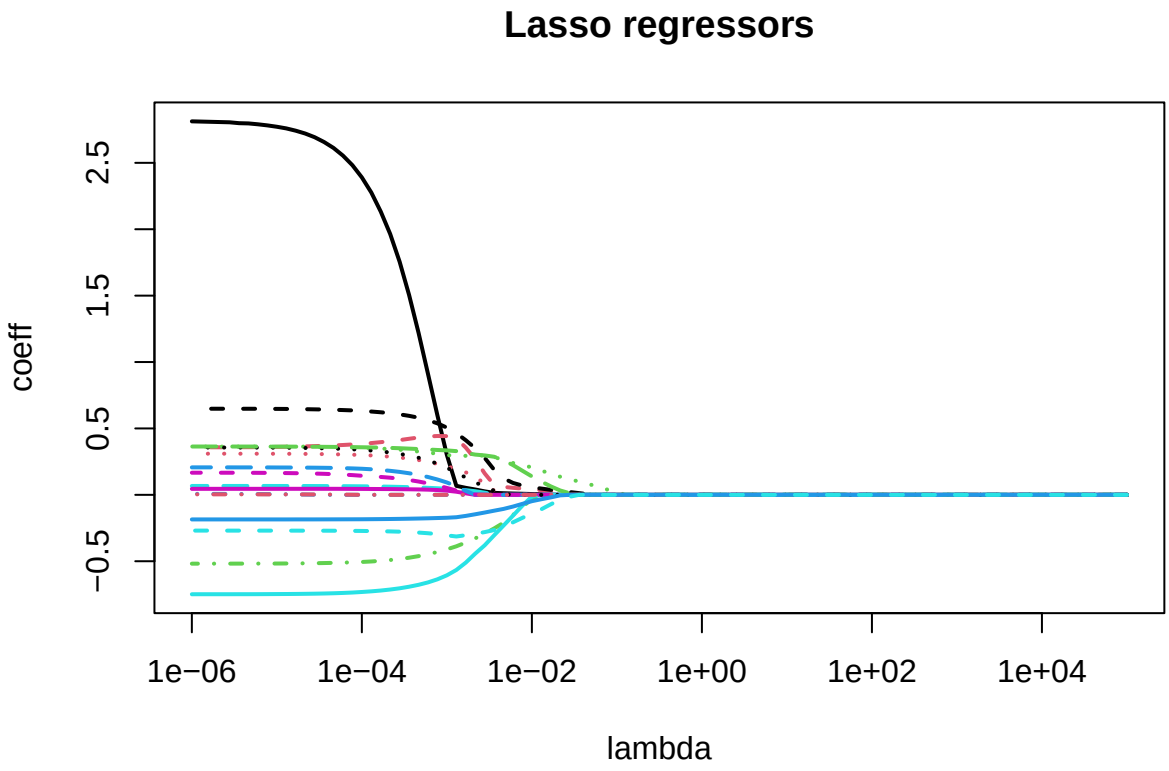
Lasso regression is a regularized version of linear regression that adds an L1 penalty term to the loss function. Like ridge, it shrinks the coefficients to reduce overfitting and handle multicollinearity. However, unlike ridge, lasso can force some coefficients to be exactly zero, effectively performing variable selection.

```
set.seed(123)
lambda.grid <- 10^seq(-6,5,length=100)
# if lambda = 0 then ridge becomes lm
# if lambda --> Inf then all coeffs --> 0 since
# RSS impact is negligible, relative to the
# role played by the penalty
ridge.mod <- glmnet::glmnet(dtm.pc.train[, !(colnames(dtm.pc) == "spam")],
                            dtm.pc.train[,  (colnames(dtm.pc) == "spam")],
                            alpha=1,
                            lambda=lambda.grid,
                            family = "binomial")
dim(coef(ridge.mod)) #     18 coeffs (one intercept + 17pcs)
```

```
## [1]  18 100
```

```
#                         x 100 lambdas
```

```
matplot(ridge.mod$lambda,
        t(as.matrix(ridge.mod$beta)),
        type="l",log="x",lwd=2,
        main="Lasso regressors",
        xlab="lambda",
        ylab="coeff")
```




```
set.seed(123)
# CV for finding the optimal lambda:
# we can use cv.glmnet in the same package
# by default: 10-fold cv
cv.out <- glmnet::cv.glmnet(as.matrix(dtm.pc.train[, !(colnames(dtm.pc) == "spam")]),
                            as.numeric(dtm.pc.train[,  (colnames(dtm.pc) == "spam")]),
                    alpha=1,
                    lambda=lambda.grid,
                    family="binomial")

cv.out$lambda.min # lambda with min estimated test error
```

```
## [1] 5.994843e-05
```

The lambda of the simpliest model with error within 1 stad dev from minimun error:

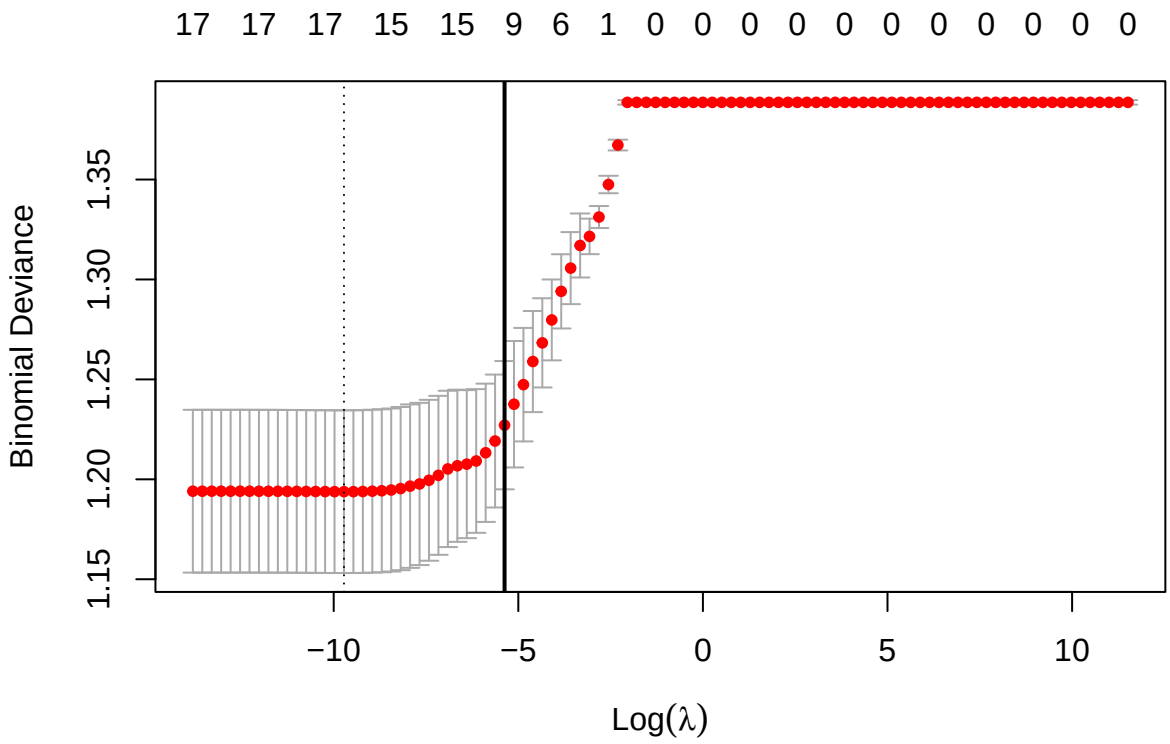
```
cv.out$lambda.1se # lambda of the simplest model
```

```
## [1] 0.004641589
```

```
# with error within 1 std from minumum error
```

Plot of the CV error: the two vertical lines are log(lambda.min) and log(lambda.1se)

```r
# plot the CV error (-+ 1 std) as a function of lambda
plot(cv.out)
# the two verticle lines are log(lambda.min) and
# log(lambda.1se)
abline(v=log(cv.out$lambda.1se),lwd=2,col="black")
```



At the top of this plot, we can observe how the number of coefficients in the model progressively decreases. This happens because, as mentioned earlier, lasso has the ability to shrink some coefficients exactly to zero, effectively removing them from the model. This is in contrast to ridge regression, which only shrinks coefficients toward zero without ever setting them exactly to zero.

```r
# Previsione delle probabilità sul training set
pred.prob <- predict(cv.out,
                     newx = as.matrix(dtm.pc.test[, !(colnames(dtm.pc) == "spam")]),
                     s = cv.out$lambda.1se,
                     type = "response")
```

```r
# Classificazione binaria (cutoff 0.5)
pred <- ifelse(pred.prob > 0.5, 1, 0)

# Test Error Rate
TER.mod <- mean(pred != dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.3461538
```

```
TER <- rbind(TER, data.frame(method = "lasso regression", training.error.rate = TER.mod))
```

```
rm(cv.out,model,ridge.mod,std.cof,X,lambda.grid,TER.mod,y,std.coeff,pred,pred.prob,poly.model)
```

Regression Tree (best)

General idea between tree based models: split the predictor space into regions and then provide the same y_hat for all observations that fall within the same region.

For categorical y: yhat will be a local mode, among those Xs in the region

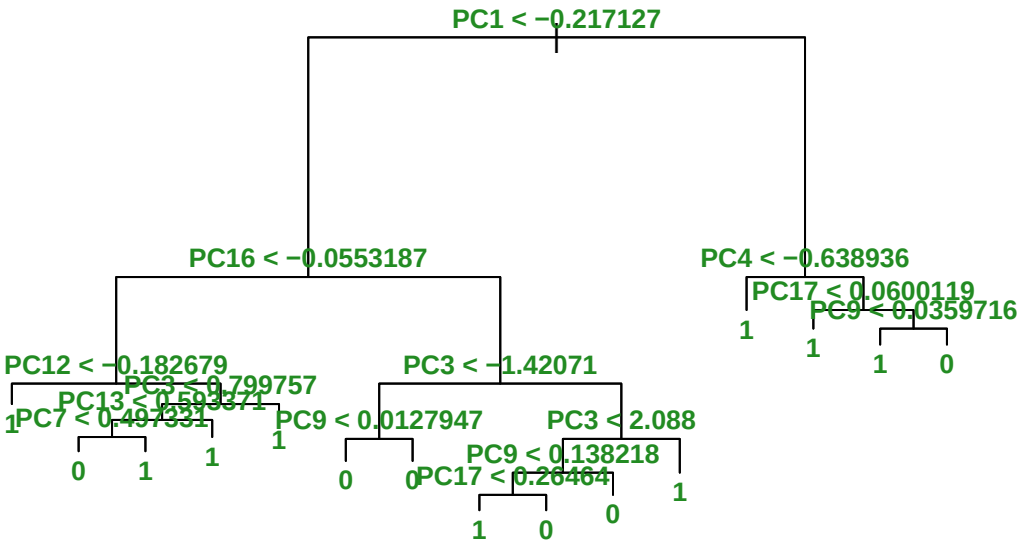
In regression trees we want to find regions of the space of predictors that minimises TER.

```
model <- tree::tree(as.factor(spam) ~ . , dtm.pc.train)

summary(model)
```

```
##
## Classification tree:
## tree::tree(formula = as.factor(spam) ~ ., data = dtm.pc.train)
## Variables actually used in tree construction:
## [1] "PC1"  "PC16" "PC12" "PC3"  "PC13" "PC7"  "PC9"  "PC17" "PC4"
## Number of terminal nodes:  15
## Residual mean deviance:  0.6943 = 913.1 / 1315
## Misclassification error rate: 0.1436 = 191 / 1330
```

Tree to prune:

```
plot(model)

# Aggiungi testo migliorato
text(model,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```



Unpruned tree

```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1800699
```

Searching the best point to prune the tree

Function to prune the large tree and to select the best subtree, according to CV: runs a K-fold cross-validation experiment to find the number of misclassifications as a function of the cost-complexity parameter k.

```
set.seed(1)

model.cv= tree::cv.tree(model,
                        FUN= prune.misclass)
```

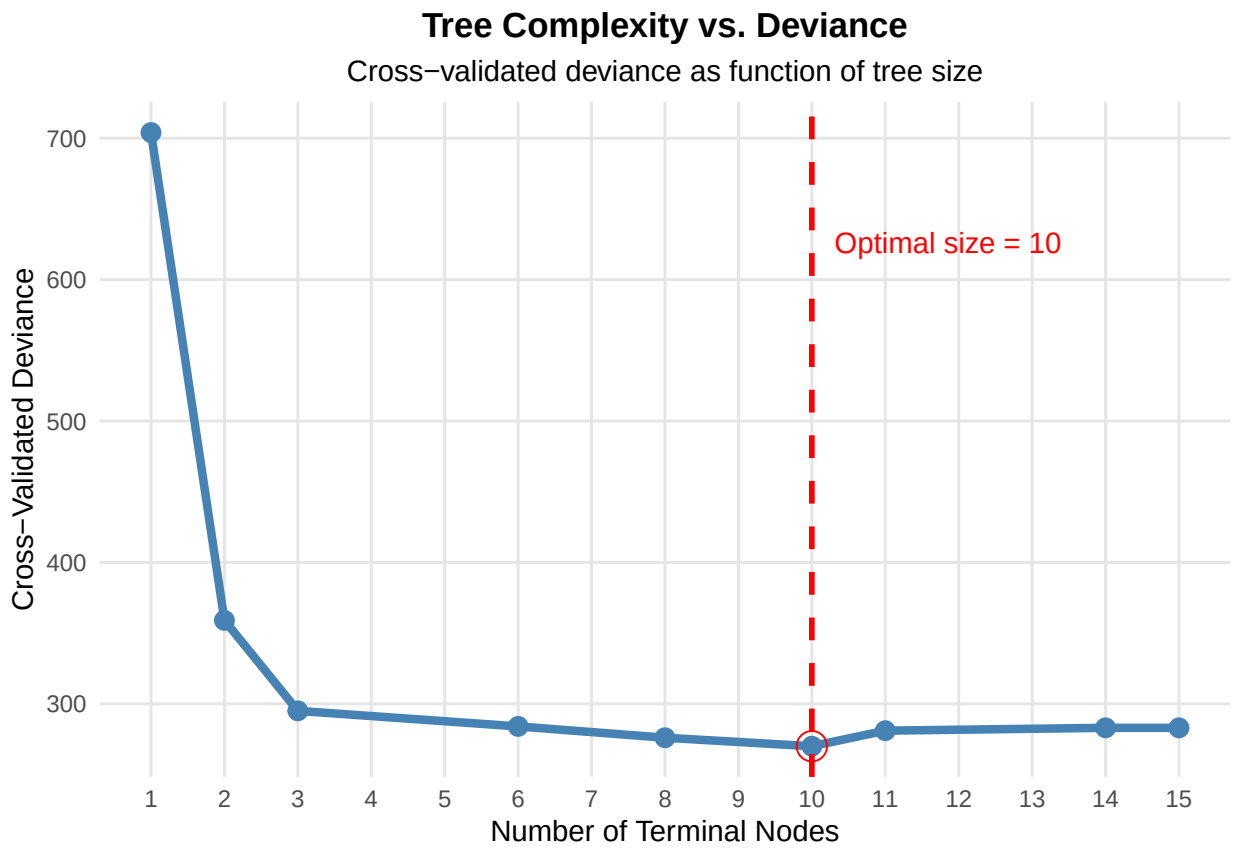
Optimal size of the tree:

```
# Trova la posizione del minimo errore
min_dev_index <- which.min(model.cv$dev)

# Numero ottimale di nodi corrispondente al minimo errore
optimal_size <- model.cv$size[min_dev_index]
optimal_size
```

```
## [1] 10
```

Number of terminal nodes

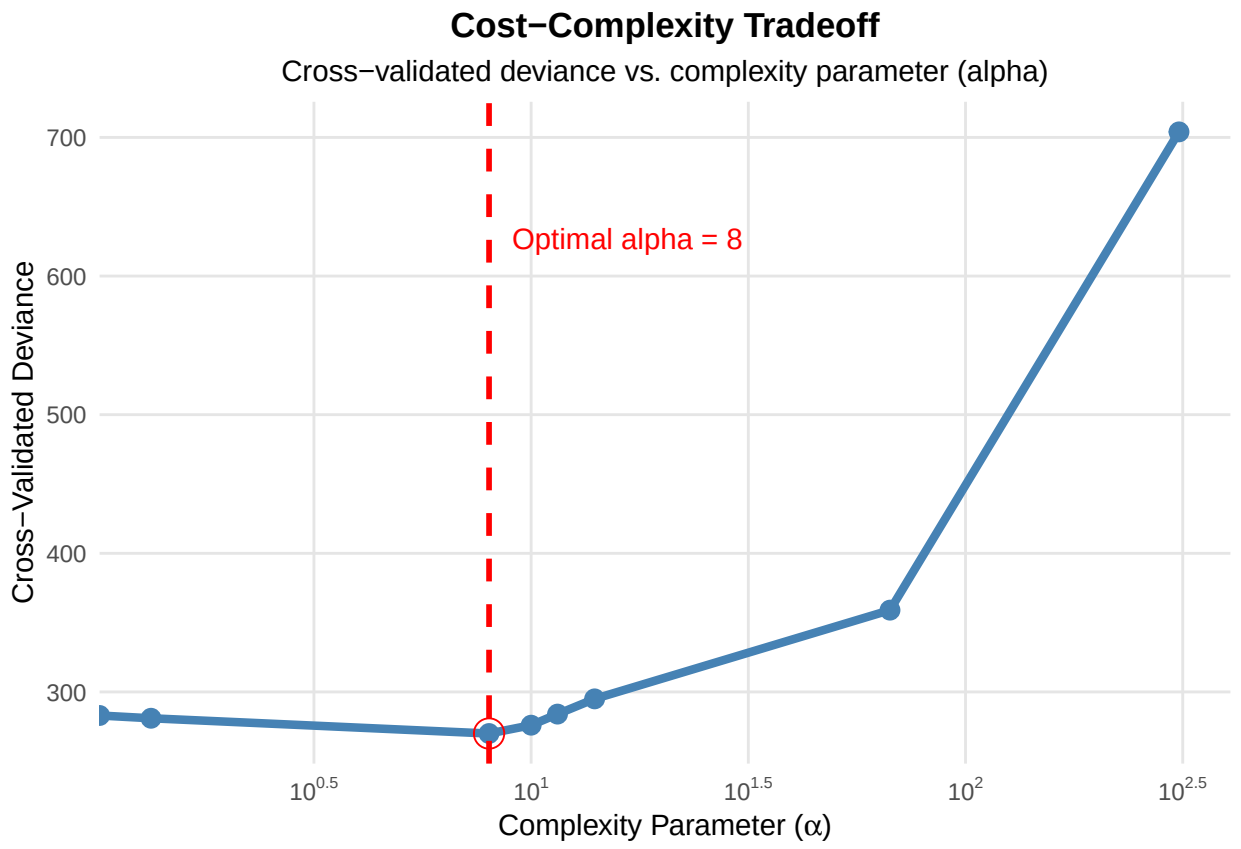
```r
# Crea un dataframe con i dati
cv_data <- data.frame(
  Size = model.cv$size,
  Deviance = model.cv$dev
)

# Trova la dimensione ottimale
optimal_size <- model.cv$size[which.min(model.cv$dev)]

# Plot migliorato con ggplot2
library(ggplot2)
ggplot(cv_data, aes(x = Size, y = Deviance)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  geom_point(color = "steelblue", size = 3, shape = 19) +
  geom_vline(xintercept = optimal_size,
             color = "red",
             linetype = "dashed",
             linewidth = 1) +
  geom_point(data = subset(cv_data, Size == optimal_size),
             color = "red",
             size = 5,
             shape = 1) +
  annotate("text",
           x = optimal_size,
           y = max(cv_data$Deviance)-100,
           label = paste("Optimal size =", optimal_size),
           color = "red",
           vjust = -1,
           hjust = -0.1) +
  scale_x_continuous(breaks = seq(min(cv_data$Size),
                                  max(cv_data$Size),
                                  by = 1)) +
  labs(title = "Tree Complexity vs. Deviance",
       subtitle = "Cross-validated deviance as function of tree size",
       x = "Number of Terminal Nodes",
       y = "Cross-Validated Deviance") +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```



Value of the cost-complexity parameter

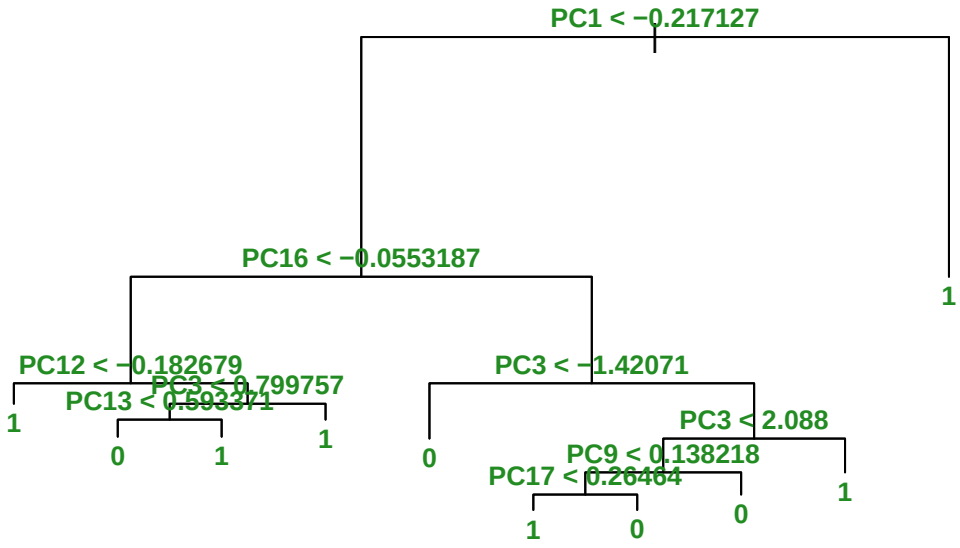
```
# Create a data frame for plotting
ccp_data <- data.frame(
  Alpha = model.cv$k,
  Deviance = model.cv$dev
)

# Find the optimal alpha value
optimal_alpha <- model.cv$k[which.min(model.cv$dev)]

# Enhanced plot with ggplot2
library(ggplot2)
ggplot(ccp_data, aes(x = Alpha, y = Deviance)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  geom_point(color = "steelblue", size = 3) +
  geom_vline(xintercept = optimal_alpha,
             color = "red",
             linetype = "dashed",
             linewidth = 1) +
  geom_point(data = subset(ccp_data, Alpha == optimal_alpha),
             color = "red",
             size = 5,
             shape = 1) +
  annotate("text",
           x = optimal_alpha,
```

```
          y = max(ccp_data$Deviance)-100,
          label = paste("Optimal alpha =", round(optimal_alpha, 4)),
          color = "red",
          vjust = -1,
          hjust = -0.1) +
  scale_x_continuous(trans = 'log10',  # Log scale for alpha values
                     breaks = scales::trans_breaks("log10", function(x) 10^x),
                     labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  labs(title = "Cost-Complexity Tradeoff",
       subtitle = "Cross-validated deviance vs. complexity parameter (alpha)",
       x = expression(paste("Complexity Parameter (", alpha, ")")),
       y = "Cross-Validated Deviance") +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```




```
library(tree)
model.pruned = tree::prune.misclass(model,
                                    best = optimal_size)
```

```
plot(model.pruned)
# Aggiungi testo migliorato
text(model.pruned,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model.pruned, dtm.pc.test, type="class")
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1923077
```

The test error rate is almost the same but the model is simplier.

```
TER <- rbind(TER, data.frame(method = "classification tree best", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model,model.cv,model.pruned,min_dev_index,optimal_size,optimal_alpha,ccp_data,cv_data)
```

Regression Tree (simplier)

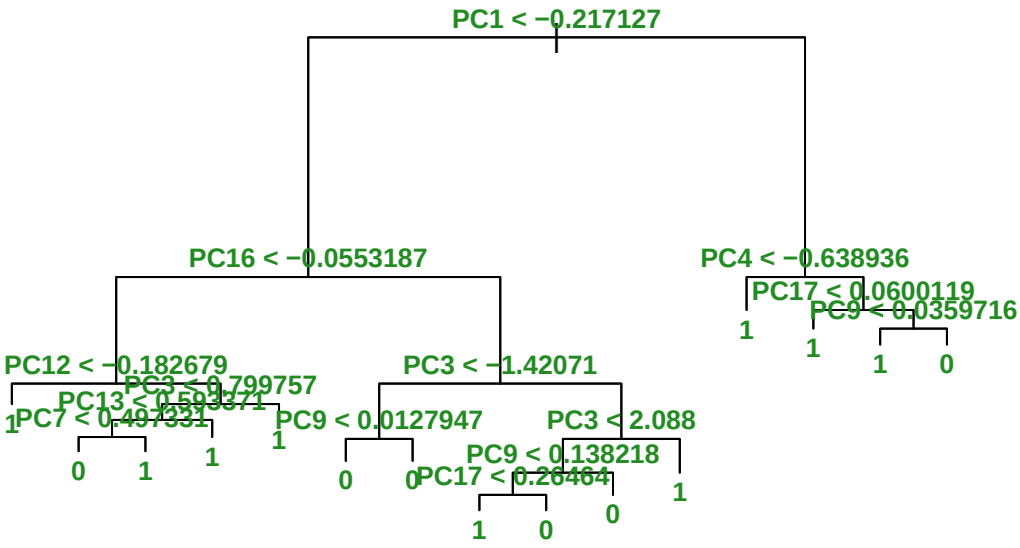
We can further prune the tree: if the plot shows that the error remains similar for a smaller number of terminal nodes, we can opt for a more parsimonious model. Specifically, we select the smallest tree whose cross-validation error is within 1 standard error of the minimum.

```
model <- tree::tree(as.factor(spam) ~ . , dtm.pc.train)

summary(model)
```

```
##
## Classification tree:
## tree::tree(formula = as.factor(spam) ~ ., data = dtm.pc.train)
## Variables actually used in tree construction:
## [1] "PC1"  "PC16" "PC12" "PC3"  "PC13" "PC7"  "PC9"  "PC17" "PC4"
## Number of terminal nodes:  15
## Residual mean deviance:  0.6943 = 913.1 / 1315
## Misclassification error rate: 0.1436 = 191 / 1330
```

Tree to prune:

```
plot(model)
text(model,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1800699
```

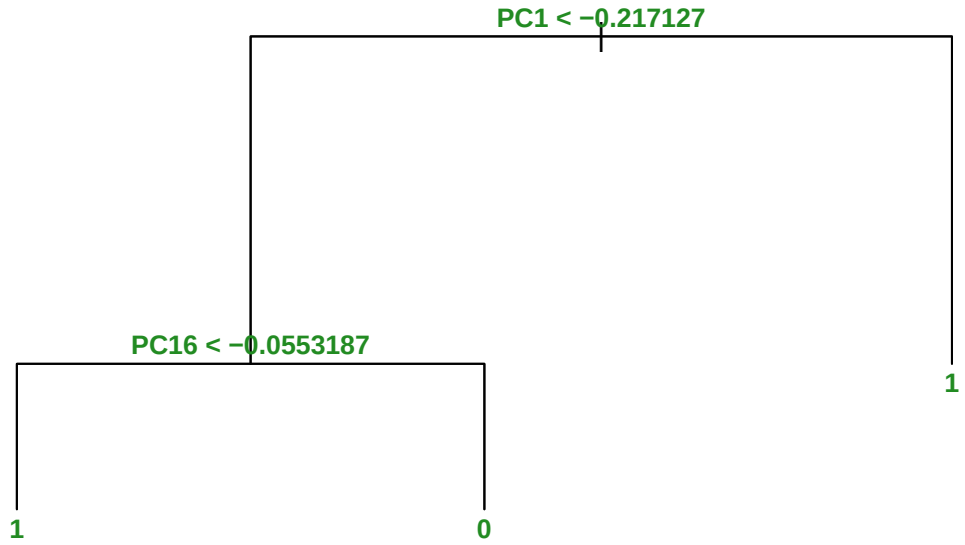
```
set.seed(1)

model.cv= tree::cv.tree(model,
                        FUN= prune.misclass)
```

```
# Calcola la devianza minima e la sua soglia "1 SE"
min_dev <- min(model.cv$dev)
se_threshold <- min_dev + sd(model.cv$dev) / sqrt(length(model.cv$dev))

# Trova il modello più semplice con errore <= soglia
simpler_size <- min(model.cv$size[model.cv$dev <= se_threshold])
simpler_size
```

```
## [1] 3
```

```
library(tree)
model.pruned = tree::prune.misclass(model,
                                    best = simpler_size)
```

```
plot(model.pruned)
text(model.pruned,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model.pruned, dtm.pc.test, type="class")
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.2272727
```

The error rate is a little bit higher, but the model is much simpler (from 10 to 3).

```
TER <- rbind(TER, data.frame(method = "classification tree 1se", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model,model.cv,model.pruned, min_dev, simpler_size, se_threshold)
```

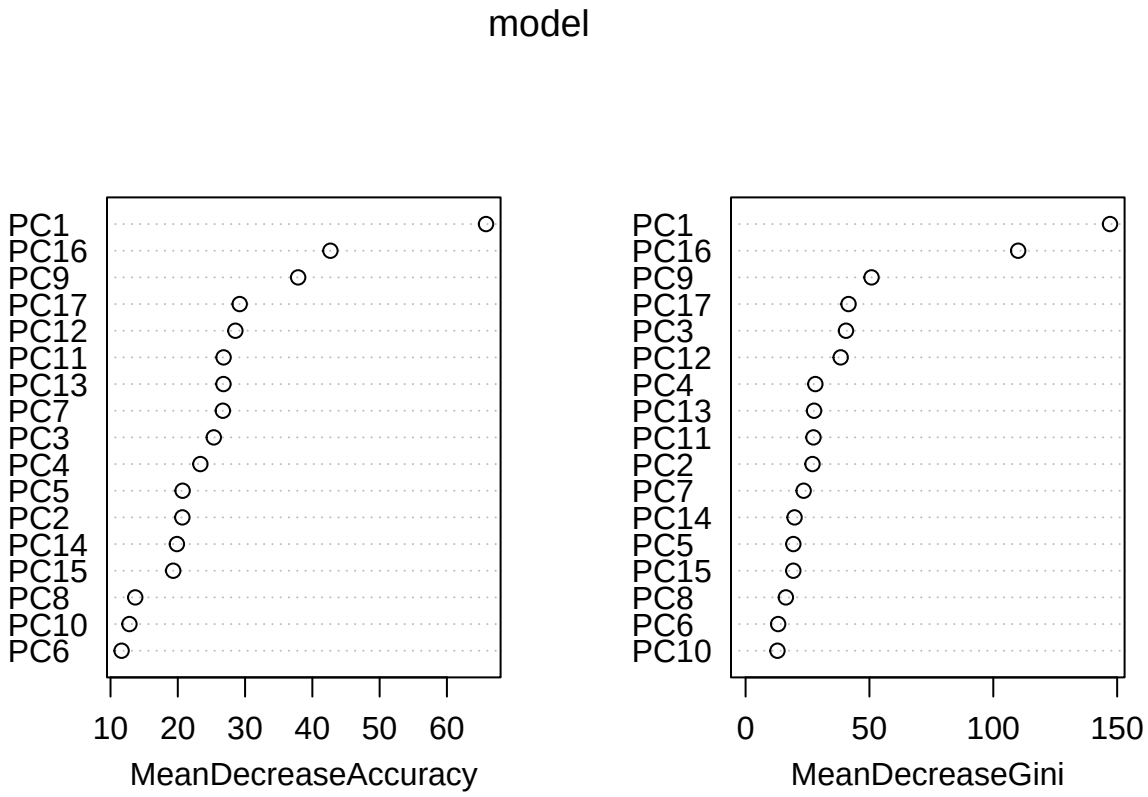
Bagging

We generate B bootstrap samples and fit a decision tree on each of them. The final prediction is obtained by aggregating the results of all trees: taking the majority vote (mode) in classification. Is a random forest with m=p (m is mtry in the formula)

```
library(randomForest)
set.seed(1)

model= randomForest(as.factor(spam) ~ . , dtm.pc.train,
                    mtry=ncol(dtm.pc.train)-1, importance=T)
model
```

```
##
## Call:
##  randomForest(formula = as.factor(spam) ~ ., data = dtm.pc.train,      mtry = ncol(dtm.pc.train) - 1
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 17
##
##         OOB estimate of  error rate: 14.96%
## Confusion matrix:
##     0   1 class.error
## 0 560 105   0.1578947
## 1  94 571   0.1413534
```

```
varImpPlot(model)
```



Two measures of variable importance are reported. The fist is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is permuted. The second is a measure of the total decrease in node impurity that results from splits over that variable, averaged over all trees

```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1311189
```

```
TER <- rbind(TER, data.frame(method = "bagging", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model,model.cv,model.pruned)
```

Random Forest

Using mtry = p (where p is the total number of features), as done above, is equivalent to fitting a Bagged Tree, since each tree considers all variables for splitting. This reduces variability among trees, potentially worsening performance due to lower model diversity. Therefore, we now use a random subset of predictors for each tree, specifically, we adopt the default value mtry = sqrt(p), 4 in this case, which promotes greater tree diversity and often improves predictive accuracy.

```r
library(randomForest)
set.seed(1)

model= randomForest(as.factor(spam) ~ . , dtm.pc.train,
                    importance=T)
model
```

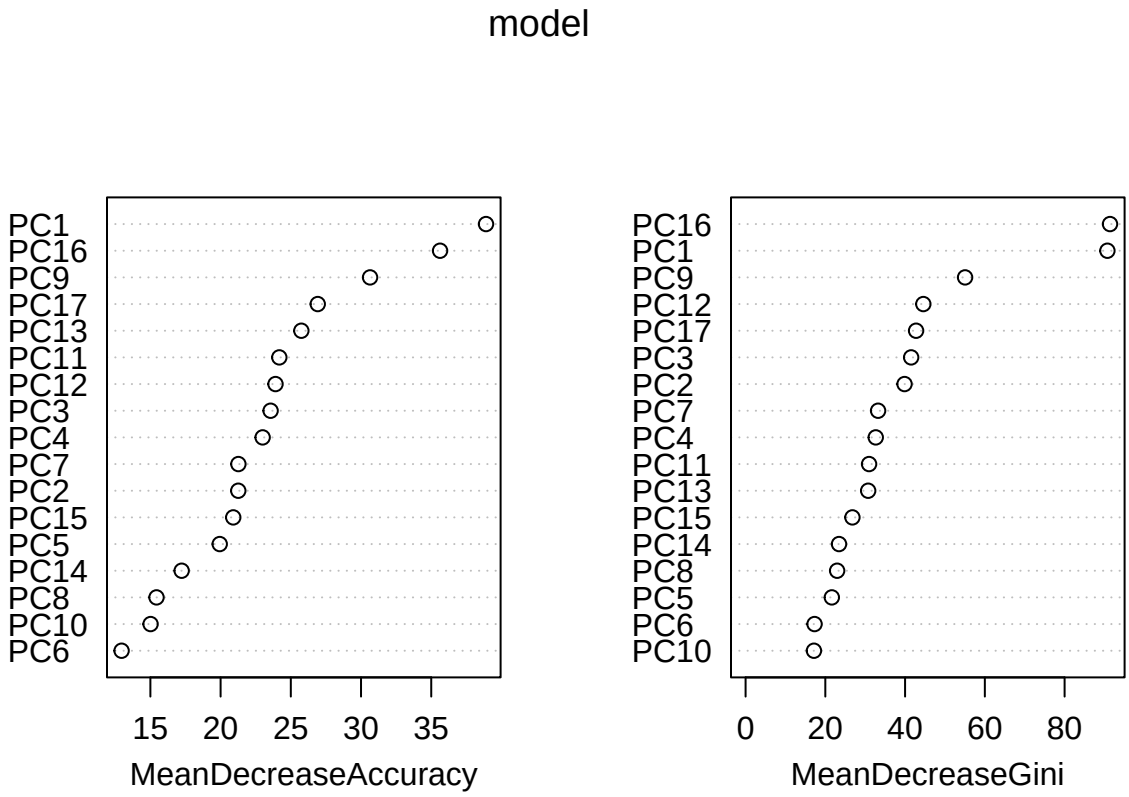
```
##
## Call:
##  randomForest(formula = as.factor(spam) ~ ., data = dtm.pc.train,      importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 13.31%
## Confusion matrix:
##     0   1 class.error
## 0 577  88   0.1323308
## 1  89 576   0.1338346
```

```r
pred= predict(model, dtm.pc.test, type="class")
```

```r
TER.mod=mean(pred!=dtm.pc.test$spam)
TER.mod
```

```
## [1] 0.1311189
```

```r
varImpPlot(model)
```




```
TER <- rbind(TER, data.frame(method = "random forest", training.error.rate = TER.mod))
```

```
rm(pred,TER.mod,model,model.cv,model.pruned)
```

Processed DTM

We now rerun all analyses using the optimized document term matrix. Most comments will not be repeated in order to avoid redundancy.

Import and Descriptive Statistics

Import the Document Term Matrix

```
dtm <- read.csv("https://raw.githubusercontent.com/PietroParenti/yt-comments-classifier/main/dtm2.csv")
dtm_original=dtm
```

```
dim(dtm_original)
```

```
## [1] 1902 1286
```

The new DTM contains 1902 observations (comments) and 1286 columns.

Response variable frequency

```r
table(dtm_original$spam)
```

```
##
##   0   1
## 951 951
```

The response variable had been balanced during the previous analyses conducted in Python.

Below are the most frequent words in the document term matrix.

```r
# excluding spam
dtm_2 <- dtm_original[ , setdiff(names(dtm_original), "spam")]

# sum on every column
col_sums <- colSums(dtm_2)

# sort
top5 <- sort(col_sums, decreasing = TRUE)[1:5]

# top 5
print(top5)
```

```
## check video  song   com    br
##   542   298   288   283   259
```

```r
rm(col_max_sum,col_sums,top5,dtm_2)
```

None of the most frequent words from the previous matrix are present here. This is because we applied English stopwords, and evidently all previously most frequent terms were among them.

Dimensional Reduction

Initial number of variables (features):

```r
ncol(dtm)
```

```
## [1] 1286
```

All words in the document term matrix that appear only once are removed.

```r
# excluding spam
X <- dtm[, setdiff(names(dtm), "spam")]

# sum on every column
col_sums <- colSums(X)

# keep only if sum > 1
X_filtered <- X[, col_sums > 1]
dtm <- cbind(X_filtered, spam = dtm$spam)

rm(X,X_filtered,col_sums)

ncol(dtm)
```

```
## [1] 1286
```

Correctly, no words were removed since, during the creation of the DTM in Python, we had chosen to keep only words with a frequency greater than or equal to 2.

Principal Components Analisis

```
dtm.pc <- prcomp(dtm[, -which(names(dtm) == "spam")],
                 scale. = TRUE)
```

```
pcs <- dtm.pc$rotation

parole <- colnames(dtm)[colnames(dtm) != "spam"]

# extracting n words per pc
top_loadings <- function(pc, n = 5) {
  loadings_abs <- abs(pc)
  top_indices <- order(loadings_abs, decreasing = TRUE)[1:n]
  data.frame(
    Words = parole[top_indices],
    Loading = round(pc[top_indices], 4)
  )
}

# number of pcs to visualize
n_pc <- 3

top_words_list <- lapply(1:n_pc, function(i) {
  top_loadings(pcs[, i], n = 5)
})

names(top_words_list) <- paste0("PC", 1:n_pc)

top_words_table <- do.call(cbind, top_words_list)

print(top_words_table[,-1 ])
```

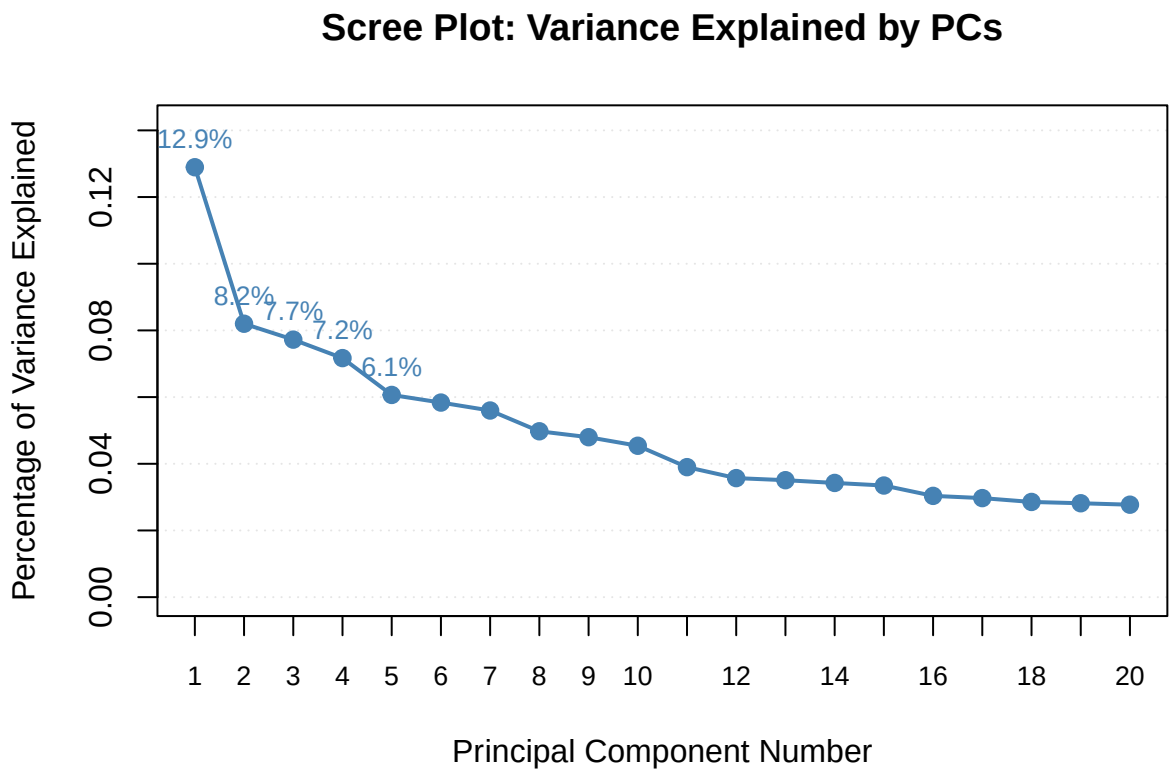
```
##           PC1.Loading   PC2.Words PC2.Loading PC3.Words PC3.Loading
## paul           0.1775       begin      0.2042 countless      -0.158
## rand           0.1766     comfort      0.2042       del      -0.158
## amendment      0.1762   currently      0.2042     drake      -0.158
## americans      0.1762      hiring      0.2042    foward      -0.158
## cares          0.1762 immediately      0.2042       fyi      -0.158
```

```
rm(n_pc,parole,top_loadings,top_words_list,top_words_table,pcs)
```

We can see above the words that contribute most to the first three principal components. Unfortunately, even though the words shown appear more interpretable than in the previous PCs, these components remain hard to interpret. Therefore, in the following analyses, we will not provide a detailed interpretation of the significant variables in the models, as doing so would be difficult and potentially misleading.

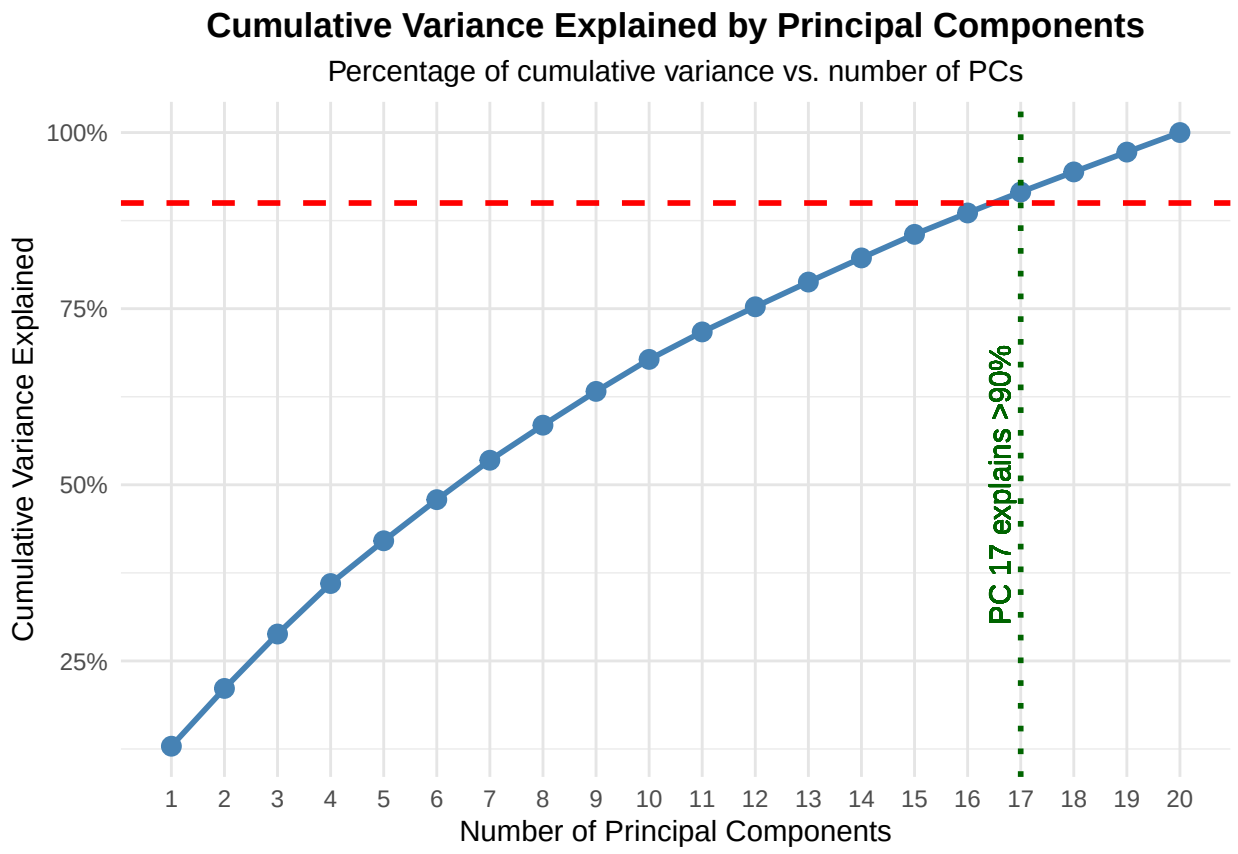
Percentage of Xs variability explained by each principal component.

```r
sd <- dtm.pc$sdev[1:20]
# Calculate explained variance for each PC
var_explained <- sd^2/sum(sd^2)

# Create enhanced scree plot
plot(var_explained,
     type = "o",
     pch = 19,
     col = "steelblue",
     lwd = 2,
     ylab = "Percentage of Variance Explained",
     xlab = "Principal Component Number",
     main = "Scree Plot: Variance Explained by PCs",
     xaxt = "n",
     ylim = c(0, max(var_explained)*1.1),
     panel.first = grid(nx = NA, ny = NULL, col = "gray90", lty = "dotted"))

# Custom x-axis with all PC numbers
axis(1, at = 1:20, labels = 1:20, las = 1, cex.axis = 0.8)


# Add percentage labels for first few PCs
text(1:5, var_explained[1:5],
     labels = paste0(round(var_explained[1:5]*100, 1), "%"),
     pos = 3, cex = 0.8, col = "steelblue")
```



Cumulative plot

```r
# Calculate cumulative variance
cum_var <- cumsum(sd^2)/sum(sd^2)

# Find number of PCs needed to explain 90% of variance
n_pc_90 <- which.max(cum_var >= 0.9)

# Enhanced plot with ggplot2
library(ggplot2)
ggplot(data.frame(PC = 1:20, CumVar = cum_var), aes(x = PC, y = CumVar)) +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(color = "steelblue", size = 3) +
  geom_hline(yintercept = 0.9, color = "red", linewidth = 1, linetype = "dashed") +
  geom_vline(xintercept = n_pc_90, color = "darkgreen", linewidth = 1, linetype = "dotted") +
  geom_text(aes(x = n_pc_90, y = 0.5,
                label = paste("PC", n_pc_90, "explains >90%")),
            color = "darkgreen", angle = 90, vjust = -0.5) +
  scale_x_continuous(breaks = 1:20, minor_breaks = NULL) +
  scale_y_continuous(labels = scales::percent_format(accuracy = 1)) +
  labs(title = "Cumulative Variance Explained by Principal Components",
       subtitle = "Percentage of cumulative variance vs. number of PCs",
       x = "Number of Principal Components",
       y = "Cumulative Variance Explained") +
  theme_minimal() +
  theme(panel.grid.major = element_line(color = "gray90"),
        plot.title = element_text(face = "bold", hjust = 0.5),
        plot.subtitle = element_text(hjust = 0.5))
```




```
# Clean up
rm(cum_var,elbow_point,n_pc_90,sd,var_explained)
```

I follow the (unwritten) rule of selecting the smallest number of principal components that explain at least 90% of the total variance. Like before, I select M = 17 components.

```
dtm.pc <- dtm.pc$x[,1:17]
dim(dtm.pc)
```

```
## [1] 1902   17
```

Adding spam column.

```
dtm.pc <- as.data.frame(dtm.pc)
dtm.pc <- cbind(dtm.pc,dtm$spam)
names(dtm.pc)[names(dtm.pc) == "dtm$spam"] <- "spam"
dim(dtm.pc)
```

```
## [1] 1902   18
```

Train and Test

The dataset is split into training and test sets: 70% for training and 30% for testing.

The training and test sets are also balanced based on the spam variable. After splitting the data, a check is performed to ensure that both subsets remain balanced.

```r
set.seed(123)

# searching for spam and non spam indexes
index_spam <- which(dtm.pc$spam == 1)
index_ham <- which(dtm.pc$spam == 0)

# sampling
train_spam <- sample(index_spam, size = 0.7 * length(index_spam))
train_ham <- sample(index_ham, size = 0.7 * length(index_ham))

train_index <- c(train_spam, train_ham)

dtm.pc.train <- dtm.pc[train_index, ]
dtm.pc.test  <- dtm.pc[-train_index, ]

# verifing
table(dtm.pc.train$spam) / nrow(dtm.pc.train)
```

```
##
##   0   1
## 0.5 0.5
```

```r
table(dtm.pc.test$spam) / nrow(dtm.pc.test)
```

```
##
##   0   1
## 0.5 0.5
```

```r
rm(index_ham,index_spam,train_ham,train_index,train_spam)
```

Code to save the datasets locally for use in Python analyses. Uncomment the chunk below to execute the saving process.

```r
#write.csv(dtm.pc.test, "C://Users//paren//Desktop//_MAGISTRALE//_Machine_Learning//progetto//dtm2.pc.t
#write.csv(dtm.pc.train, "C://Users//paren//Desktop//_MAGISTRALE//_Machine_Learning//progetto//dtm2.pc.
```

Models evaluation

Now principal components extracted from the text data are used as input features to assess the classification accuracy of various machine learning models.

Each model is evaluated using the test error rate.

Linear Model

```r
model <- lm(spam ~ ., data = dtm.pc.train)

summary(model)
```

```
##
## Call:
## lm(formula = spam ~ ., data = dtm.pc.train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.3559 -0.4290 -0.4077  0.5386  0.5849
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.5013053  0.0130458  38.427  < 2e-16 ***
## PC1          0.0070237  0.0031797   2.209 0.027353 *
## PC2          0.0189030  0.0033906   5.575 3.00e-08 ***
## PC3         -0.0175413  0.0033819  -5.187 2.48e-07 ***
## PC4          0.0122570  0.0035298   3.472 0.000532 ***
## PC5          0.0083365  0.0034701   2.402 0.016427 *
## PC6         -0.0136955  0.0034260  -3.998 6.76e-05 ***
## PC7          0.0146708  0.0041506   3.535 0.000423 ***
## PC8          0.0117699  0.0054789   2.148 0.031877 *
## PC9          0.0238890  0.0120064   1.990 0.046833 *
## PC10         0.0018974  0.0046430   0.409 0.682861
## PC11        -0.0164263  0.0042229  -3.890 0.000105 ***
## PC12        -0.0065602  0.0050207  -1.307 0.191572
## PC13        -0.0153765  0.0042905  -3.584 0.000351 ***
## PC14        -0.0014373  0.0042017  -0.342 0.732347
## PC15        -0.0056366  0.0043365  -1.300 0.193899
## PC16        -0.0208798  0.0054841  -3.807 0.000147 ***
## PC17        -0.0006418  0.0044694  -0.144 0.885845
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4732 on 1312 degrees of freedom
## Multiple R-squared:  0.1166, Adjusted R-squared:  0.1051
## F-statistic: 10.18 on 17 and 1312 DF,  p-value: < 2.2e-16
```

The linear model fits the binary outcome spam using 17 principal components (PCs) as predictors. The overall model is statistically significant ($F = 10.18$, $p < 2.2e\text{-}16$), but the adjusted R^2 is relatively low (0.105), indicating limited explanatory power.

Many components show statistically significant associations with the response variable, suggesting they contribute to differentiating between spam and non-spam comments.

However, like before, interpreting individual PCs is inherently difficult, as each component is a linear combination of thousands of original terms. The direction and magnitude of coefficients do not directly map back to specific words or features, limiting the model's interpretability despite statistical significance. This limitation applies to all models that follow—since they rely on the same transformed input space—but this comment will not be repeated to avoid redundancy.

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
pred <- ifelse(pred > 0.5, 1, 0)
```

Training error rate calculation

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3653846
```

All the training error rates are stored in a data frame to make easier future comparison.

```
TER2 <- data.frame(
  method = character(),
  training.error.rate = numeric(),
  stringsAsFactors = FALSE
)
```

```
TER2 <- rbind(TER2, data.frame(method = "linear model", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model)
```

Logistic Model

```
model <- glm(spam ~ ., data = dtm.pc.train, family = binomial)

summary(model)
```

```
##
## Call:
## glm(formula = spam ~ ., family = binomial, data = dtm.pc.train)
##
## Coefficients:
##             Estimate Std. Error z value Pr(>|z|)
## (Intercept)  1.91237    0.20125   9.502  < 2e-16 ***
## PC1          1.09698    0.27692   3.961 7.45e-05 ***
## PC2          0.85275    0.24956   3.417 0.000633 ***
## PC3         -0.27991    0.13911  -2.012 0.044201 *
## PC4          0.06715    0.13501   0.497 0.618907
## PC5          0.60319    0.13911   4.336 1.45e-05 ***
## PC6         -1.37696    0.21898  -6.288 3.21e-10 ***
## PC7          0.53922    0.12079   4.464 8.04e-06 ***
## PC8          0.23621    0.13984   1.689 0.091199 .
## PC9         -0.26983    0.11128  -2.425 0.015323 *
## PC10         0.44214    0.15330   2.884 0.003926 **
## PC11        -0.27179    0.13476  -2.017 0.043705 *
## PC12        -0.36617    0.20377  -1.797 0.072344 .
## PC13        -0.21901    0.07047  -3.108 0.001883 **
```

```
## PC14        -0.02034    0.10402  -0.196 0.844948
## PC15         0.17637    0.08972   1.966 0.049338 *
## PC16        -1.27911    0.18400  -6.952 3.61e-12 ***
## PC17        -0.13298    0.05149  -2.583 0.009797 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1843.8  on 1329  degrees of freedom
## Residual deviance: 1398.4  on 1312  degrees of freedom
## AIC: 1434.4
##
## Number of Fisher Scoring iterations: 8
```

The logistic regression model is statistically significant. Several PCs are highly significant predictors of the spam class.

```
pred <- predict(model, newdata = dtm.pc.test, type= "response")
```

```
pred <- ifelse(pred > 0.5, 1, 0)
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3076923
```

```
TER2 <- rbind(TER2, data.frame(method = "logistic model", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model)
```

Linear Discriminant Analisis

```
model <- MASS::lda(spam~.,
                   dtm.pc.train)

model
```

```
## Call:
## lda(spam ~ ., data = dtm.pc.train)
##
## Prior probabilities of groups:
##   0   1
## 0.5 0.5
##
## Group means:
##          PC1        PC2        PC3        PC4        PC5        PC6        PC7
## 0 -0.3019911 -0.6775705  0.5818581 -0.3713276 -0.2127065  0.4217374 -0.2875456
## 1  0.1452761  0.6310567 -0.7090797  0.5508954  0.1245839 -0.4869704  0.2581268
```

```
##           PC8         PC9       PC10       PC11        PC12       PC13
## 0 -0.08778847 -0.02546723 -0.1787317  0.3199373  0.09445205  0.2798083
## 1  0.08516167 -0.18547545  0.2022250 -0.3340194 -0.06534247 -0.3360551
##          PC14       PC15       PC16         PC17
## 0 -0.02185869  0.1224762  0.2547146 -0.002253088
## 1  0.03174480 -0.1589007 -0.2138884  0.015810322
##
## Coefficients of linear discriminants:
##               LD1
## PC1   0.043741758
## PC2   0.117722145
## PC3  -0.109242018
## PC4   0.076332936
## PC5   0.051917146
## PC6  -0.085291412
## PC7   0.091365223
## PC8   0.073299647
## PC9   0.148773650
## PC10  0.011816255
## PC11 -0.102298083
## PC12 -0.040854750
## PC13 -0.095760703
## PC14 -0.008951199
## PC15 -0.035103258
## PC16 -0.130033218
## PC17 -0.003996709
```

```
summary(model)
```

```
##         Length Class  Mode
## prior    2     -none- numeric
## counts   2     -none- numeric
## means   34     -none- numeric
## scaling 17     -none- numeric
## lev      2     -none- character
## svd      1     -none- numeric
## N        1     -none- numeric
## call     3     -none- call
## terms    3     terms  call
## xlevels  0     -none- list
```

The LDA output shows:

- prior probabilities: set to 0.5 for both classes, reflecting a balanced dataset
- group means: average values of each principal component (PC) per class, larger differences suggest stronger discriminative power
- coefficients of linear discriminants (LD1): weights used to form a linear combination of PCs that best separates the two classes. PCs with larger absolute values contribute more to class separation.

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
pred <- pred$class
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3653846
```

"Curiously, it turns out that the classifcations that we get if we use linear regression to predict a binary response will be the same as for the linear discriminant analysis (LDA)" An Introduction to Statistical Learning, page 132

```
TER2 <- rbind(TER2, data.frame(method = "linear discriminant analisis", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model)
```

Quadratic Discriminant Analisis

```
model <- MASS::qda(spam~.,
                   dtm.pc.train)

model
```

```
## Call:
## qda(spam ~ ., data = dtm.pc.train)
##
## Prior probabilities of groups:
##   0   1
## 0.5 0.5
##
## Group means:
##          PC1        PC2        PC3        PC4        PC5        PC6        PC7
## 0 -0.3019911 -0.6775705  0.5818581 -0.3713276 -0.2127065  0.4217374 -0.2875456
## 1  0.1452761  0.6310567 -0.7090797  0.5508954  0.1245839 -0.4869704  0.2581268
##           PC8         PC9       PC10       PC11        PC12       PC13
## 0 -0.08778847 -0.02546723 -0.1787317  0.3199373  0.09445205  0.2798083
## 1  0.08516167 -0.18547545  0.2022250 -0.3340194 -0.06534247 -0.3360551
##          PC14       PC15       PC16         PC17
## 0 -0.02185869  0.1224762  0.2547146 -0.002253088
## 1  0.03174480 -0.1589007 -0.2138884  0.015810322
```

```
summary(model)
```

```
##         Length Class  Mode
## prior     2    -none- numeric
## counts    2    -none- numeric
## means    34    -none- numeric
## scaling 578    -none- numeric
## ldet      2    -none- numeric
## lev       2    -none- character
```

```
## N         1    -none- numeric
## call      3    -none- call
## terms     3    terms  call
## xlevels   0    -none- list
```

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
pred <- pred$class
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3618881
```

```
TER2 <- rbind(TER2, data.frame(method = "quadratic discriminant analisis", training.error.rate = TER2.m
```

```
rm(pred,TER2.mod,model)
```

Naive Bayes

```
model <- e1071::naiveBayes(spam~.,
                  dtm.pc.train)

model
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##   0   1
## 0.5 0.5
##
## Conditional probabilities:
##    PC1
## Y         [,1]      [,2]
##   0 -0.3019911 0.2329948
##   1  0.1452761 5.8365323
##
##    PC2
## Y         [,1]      [,2]
##   0 -0.6775705 0.3853568
##   1  0.6310567 5.4749095
##
##    PC3
## Y         [,1]      [,2]
##   0  0.5818581 0.5990746
```

```
##   1 -0.7090797 5.7809872
##
##    PC4
## Y         [,1]      [,2]
##   0 -0.3713276 0.4686997
##   1  0.5508954 5.7249282
##
##    PC5
## Y         [,1]      [,2]
##   0 -0.2127065 0.4427985
##   1  0.1245839 5.4024812
##
##    PC6
## Y         [,1]      [,2]
##   0  0.4217374 0.3525088
##   1 -0.4869704 5.3621131
##
##    PC7
## Y         [,1]      [,2]
##   0 -0.2875456 0.4490929
##   1  0.2581268 5.2800635
##
##    PC8
## Y          [,1]      [,2]
##   0 -0.08778847 0.5101687
##   1  0.08516167 5.2251569
##
##    PC9
## Y          [,1]      [,2]
##   0 -0.02546723 0.6114545
##   1 -0.18547545 2.8393641
##
##    PC10
## Y         [,1]      [,2]
##   0 -0.1787317 0.4742955
##   1  0.2022250 4.7703829
##
##    PC11
## Y         [,1]      [,2]
##   0  0.3199373 0.4613105
##   1 -0.3340194 4.3347938
##
##    PC12
## Y          [,1]      [,2]
##   0  0.09445205 0.2314885
##   1 -0.06534247 3.6675926
##
##    PC13
## Y         [,1]      [,2]
##   0  0.2798083 0.4469709
##   1 -0.3360551 4.2548974
##
##    PC14
## Y          [,1]      [,2]
```

```
##   0 -0.02185869 0.7454947
##   1  0.03174480 4.5113691
##
##    PC15
## Y         [,1]     [,2]
##   0  0.1224762 1.098905
##   1 -0.1589007 4.250949
##
##    PC16
## Y         [,1]      [,2]
##   0  0.2547146 0.5099359
##   1 -0.2138884 3.3226994
##
##    PC17
## Y           [,1]     [,2]
##   0 -0.002253088 2.049828
##   1  0.015810322 3.603573
```

```
summary(model)
```

```
##           Length Class  Mode
## apriori    2     table  numeric
## tables    17     -none- list
## levels     2     -none- character
## isnumeric 17     -none- logical
## call       4     -none- call
```

```
pred <- predict(model, newdata = dtm.pc.test)
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3811189
```

```
TER2 <- rbind(TER2, data.frame(method = "naive bayes", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model)
```

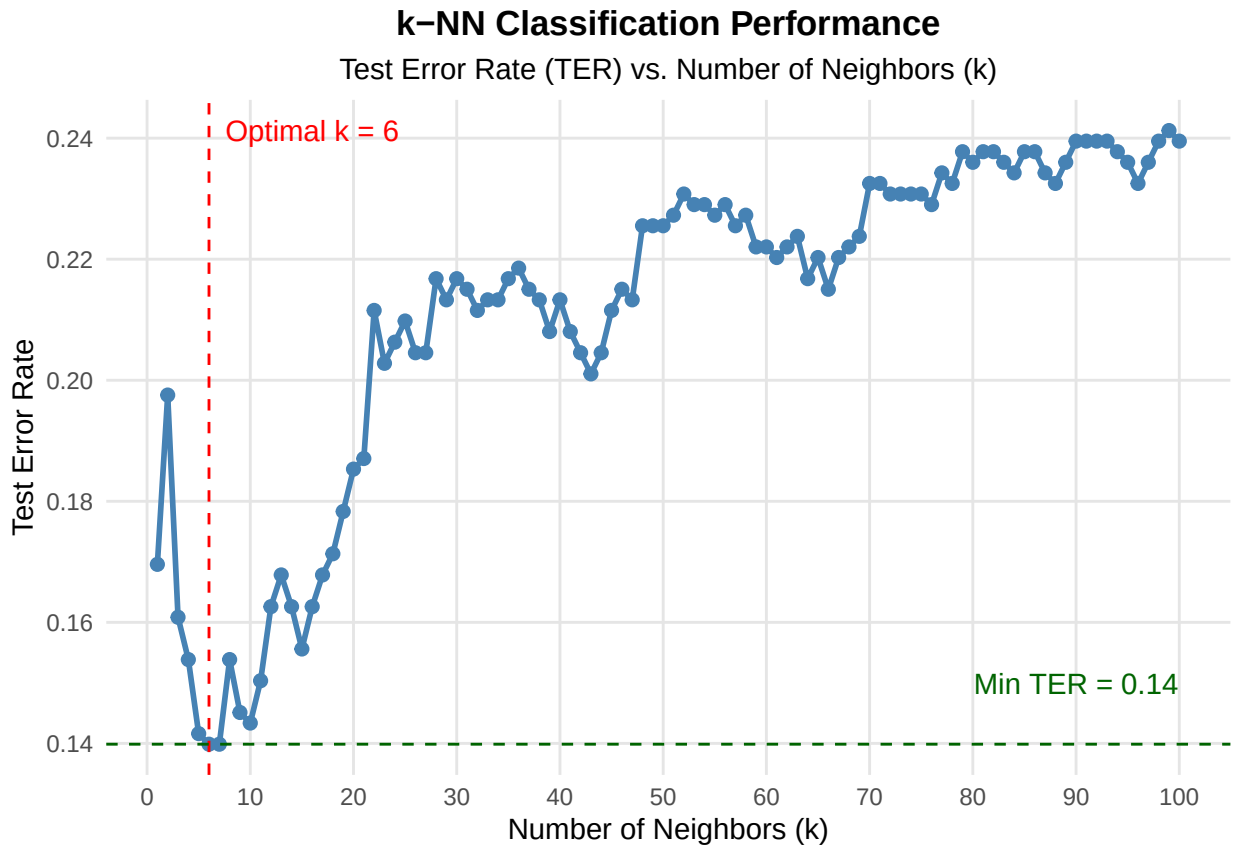
KNN

Searching the best k between 1 and 100.

```
set.seed(123)
TER2.models=c()

for (k in seq(1, 100, by = 1)) {
  pred <- class::knn(cbind(dtm.pc.train[, !(colnames(dtm.pc.train) == "spam")]),
                     cbind(dtm.pc.test[, !(colnames(dtm.pc.train) == "spam")]),
                     dtm.pc.train$spam,
                     k = k)
```

```
  TER2.mod=mean(pred!=dtm.pc.test$spam)
  TER2.models=c(TER2.models,TER2.mod)
}

TER2.models=cbind(seq(1, 100, by = 1),TER2.models)
```

```
library(ggplot2)

# Convert results to dataframe for ggplot
results_df <- data.frame(
  k = TER2.models[, 1],
  TER2 = TER2.models[, 2]
)

# Find optimal k (minimum TER2)
optimal_k <- results_df$k[which.min(results_df$TER2)]
min_TER2 <- min(results_df$TER2)

# Create plot with ggplot2
ggplot(results_df, aes(x = k, y = TER2)) +
  geom_line(color = "steelblue", linewidth = 1) +
  geom_point(color = "steelblue", size = 2) +
  geom_vline(xintercept = optimal_k, linetype = "dashed", color = "red") +
  geom_hline(yintercept = min_TER2, linetype = "dashed", color = "darkgreen") +
  annotate("text",
           x = optimal_k + 10,
           y = max(results_df$TER2),
           label = paste("Optimal k =", optimal_k),
           color = "red") +
  annotate("text",
           x = 90,
           y = min_TER2 + 0.01,
           label = paste("Min TER =", round(min_TER2, 3)),
           color = "darkgreen") +
  labs(title = "k-NN Classification Performance",
       subtitle = "Test Error Rate (TER) vs. Number of Neighbors (k)",
       x = "Number of Neighbors (k)",
       y = "Test Error Rate") +
  scale_x_continuous(breaks = seq(0, 100, by = 10)) +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```



The value of k that minimizes the TER is 6.

```
min(TER2.models)
```

```
## [1] 0.1398601
```

```
#k=6
```

Rerunning the KNN regression with k = 6.

```
set.seed(123)

pred <- class::knn(cbind(dtm.pc.train[, !(colnames(dtm.pc.train) == "spam")]),
                        cbind(dtm.pc.test[, !(colnames(dtm.pc.train) == "spam")]),
                        dtm.pc.train$spam,
                        k = optimal_k)
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1520979
```

```
TER2 <- rbind(TER2, data.frame(method = "KNN", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,k,TER2.models,results_df,min_TER2,optimal_k)
```

Ridge Regression

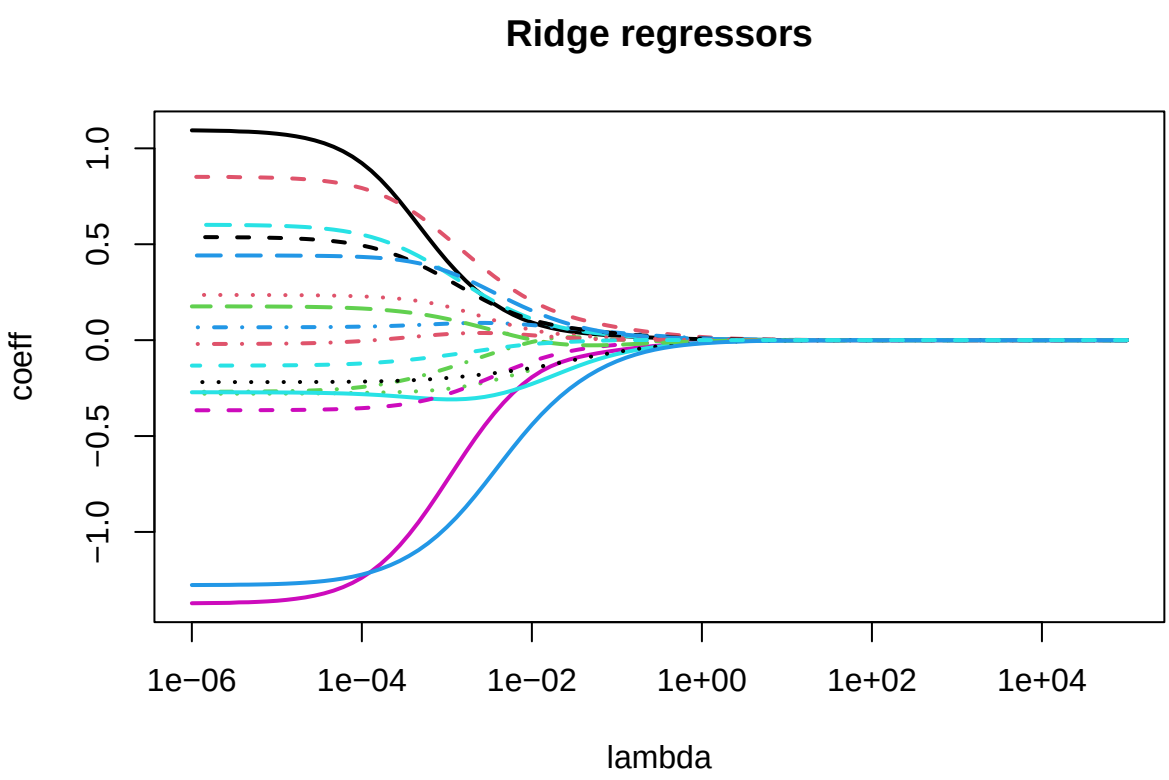
```
set.seed(123)
lambda.grid <- 10^seq(-6,5,length=100)
# if lambda = 0 then ridge becomes lm
# if lambda --> Inf then all coeffs --> 0 since
# RSS impact is negligible, relative to the
# role played by the penalty
ridge.mod <- glmnet::glmnet(dtm.pc.train[, !(colnames(dtm.pc) == "spam")],
                            dtm.pc.train[,  (colnames(dtm.pc) == "spam")],
                            alpha=0,
                            lambda=lambda.grid,
                            family = "binomial")
dim(coef(ridge.mod)) #     18 coeffs (one intercept + 17pcs)
```

```
## [1]  18 100
```

```
#                          x 100 lambdas
```

Choosing the best lambda

```
matplot(ridge.mod$lambda,
        t(as.matrix(ridge.mod$beta)),
        type="l",log="x",lwd=2,
        main="Ridge regressors",
        xlab="lambda",
        ylab="coeff")
```



We use CV for finding the optimal lambda:

```
set.seed(123)

# CV for finding the optimal lambda:
# we can use cv.glmnet in the same package
# by default: 10-fold cv
cv.out <- glmnet::cv.glmnet(as.matrix(dtm.pc.train[, !(colnames(dtm.pc) == "spam")]),
                            as.numeric(dtm.pc.train[,  (colnames(dtm.pc) == "spam")]),
                    alpha=0,
                    lambda=lambda.grid,
                    family="binomial")

cv.out$lambda.min # lambda with min estimated test error
```

```
## [1] 0.0003593814
```

The lambda of the simpliest model with error within 1 stad dev from minimun error:

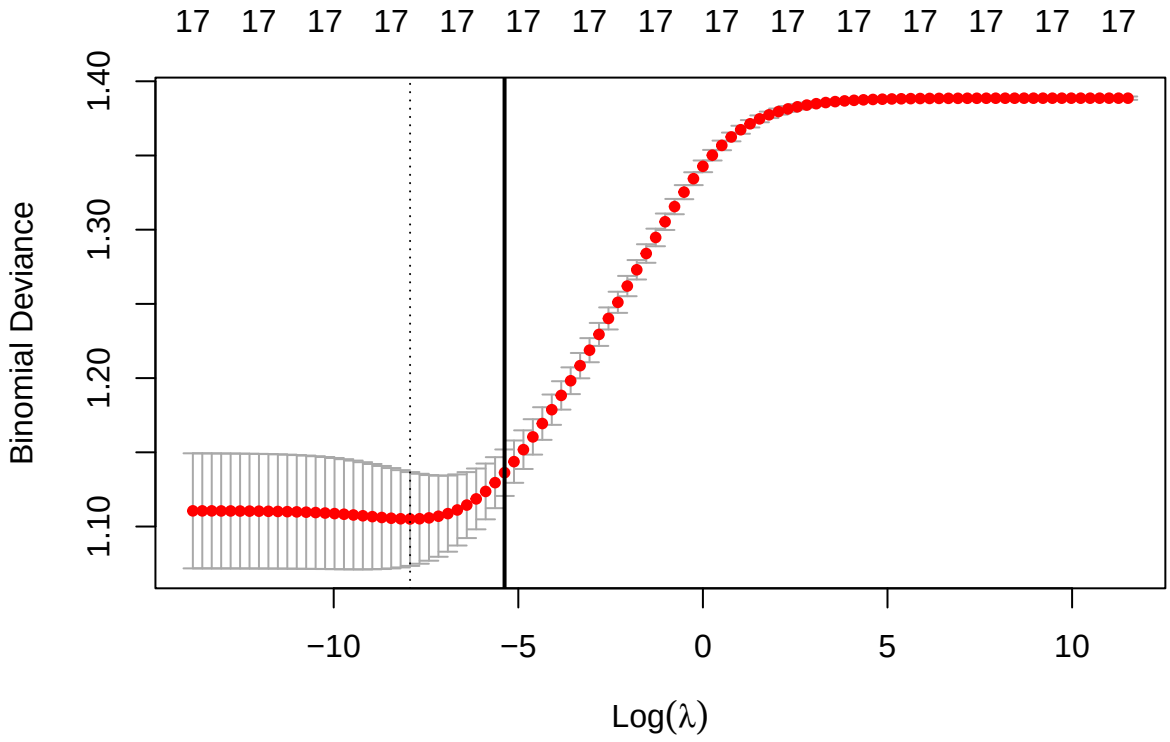
```
cv.out$lambda.1se # lambda of the simplest model
```

```
## [1] 0.004641589
```

```
# with error within 1 std from minumum error
```

Plot of the CV error: the two vertical lines are log(lambda.min) and log(lambda.1se)

```
# plot the CV error (-+ 1 std) as a function of lambda
plot(cv.out)
# the two verticle lines are log(lambda.min) and
# log(lambda.1se)
abline(v=log(cv.out$lambda.1se),lwd=2,col="black")
```




```
# Previsione delle probabilità sul training set
pred.prob <- predict(cv.out,
                     newx = as.matrix(dtm.pc.test[, !(colnames(dtm.pc) == "spam")]),
                     s = cv.out$lambda.1se,
                     type = "response")
```

```
# Classificazione binaria (cutoff 0.5)
pred <- ifelse(pred.prob > 0.5, 1, 0)

# Test Error Rate
TER2.mod <- mean(pred != dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3304196
```

```
TER2 <- rbind(TER2, data.frame(method = "ridge regression", training.error.rate = TER2.mod))
```

```
rm(cv.out,model,ridge.mod,std.cof,X,lambda.grid,TER2.mod,y,std.coeff,pred,pred.prob,lambda_values,cv_da
```

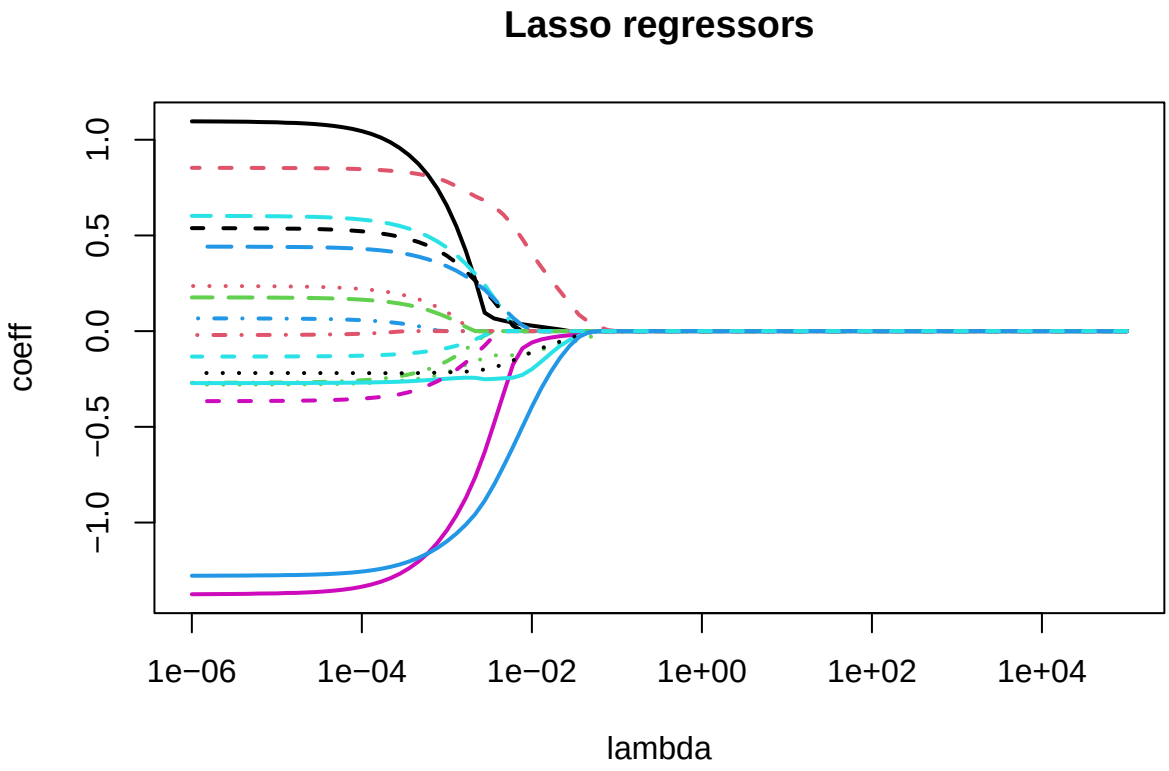
Lasso Regression

```
set.seed(123)
lambda.grid <- 10^seq(-6,5,length=100)
# if lambda = 0 then ridge becomes lm
# if lambda --> Inf then all coeffs --> 0 since
# RSS impact is negligible, relative to the
# role played by the penalty
ridge.mod <- glmnet::glmnet(dtm.pc.train[, !(colnames(dtm.pc) == "spam")],
                            dtm.pc.train[,  (colnames(dtm.pc) == "spam")],
                            alpha=1,
                            lambda=lambda.grid,
                            family = "binomial")
dim(coef(ridge.mod)) #     18 coeffs (one intercept + 17pcs)
```

```
## [1]  18 100
```

```
#                          x 100 lambdas
```

```
matplot(ridge.mod$lambda,
        t(as.matrix(ridge.mod$beta)),
        type="l",log="x",lwd=2,
        main="Lasso regressors",
        xlab="lambda",
        ylab="coeff")
```




```
set.seed(123)

# CV for finding the optimal lambda:
# we can use cv.glmnet in the same package
# by default: 10-fold cv
cv.out <- glmnet::cv.glmnet(as.matrix(dtm.pc.train[, !(colnames(dtm.pc) == "spam")]),
                            as.numeric(dtm.pc.train[,  (colnames(dtm.pc) == "spam")]),
                    alpha=1,
                    lambda=lambda.grid,
                    family="binomial")

cv.out$lambda.min # lambda with min estimated test error
```

```
## [1] 0.0007742637
```

The lambda of the simpliest model with error within 1 stad dev from minimun error:

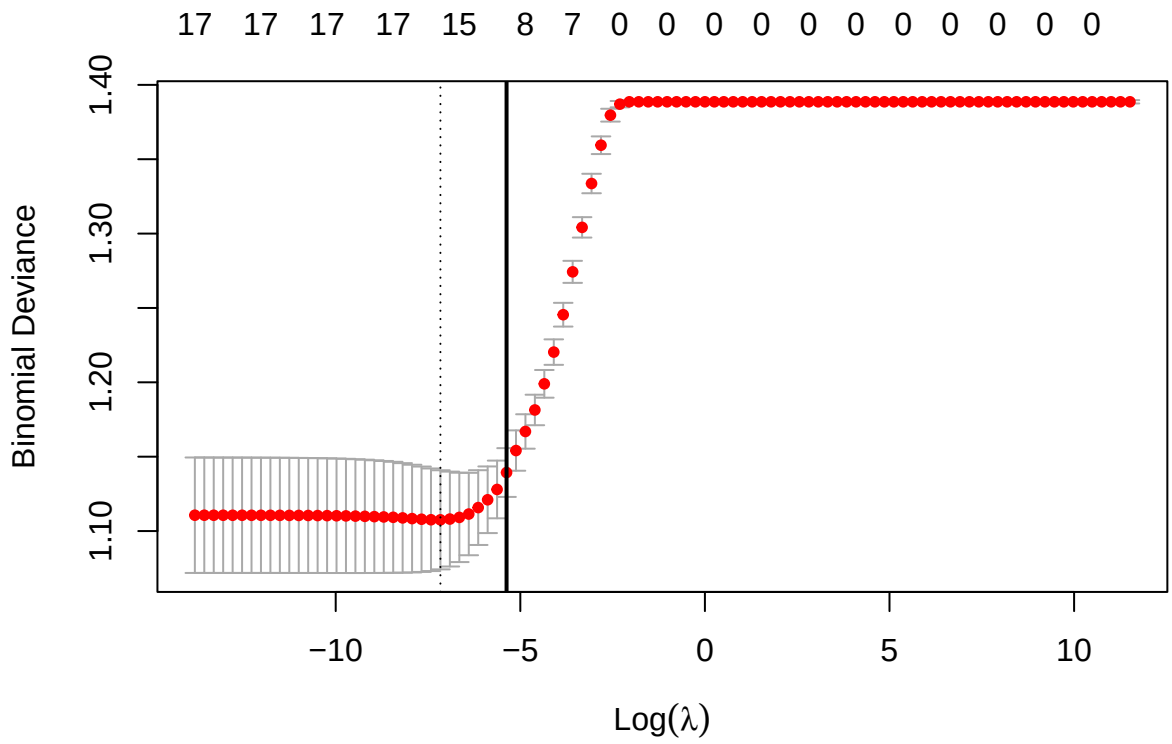
```
cv.out$lambda.1se # lambda of the simplest model
```

```
## [1] 0.004641589
```

```
# with error within 1 std from minumum error
```

Plot of the CV error: the two vertical lines are log(lambda.min) and log(lambda.1se)

```r
# plot the CV error (-+ 1 std) as a function of lambda
plot(cv.out)
# the two verticle lines are log(lambda.min) and
# log(lambda.1se)
abline(v=log(cv.out$lambda.1se),lwd=2,col="black")
```



At the top of this plot, we can observe how the number of coefficients in the model progressively decreases. This happens because, as mentioned earlier, lasso has the ability to shrink some coefficients exactly to zero, effectively removing them from the model. This is in contrast to ridge regression, which only shrinks coefficients toward zero without ever setting them exactly to zero.

```r
# Previsione delle probabilità sul training set
pred.prob <- predict(cv.out,
                     newx = as.matrix(dtm.pc.test[, !(colnames(dtm.pc) == "spam")]),
                     s = cv.out$lambda.1se,
                     type = "response")
```

```r
# Classificazione binaria (cutoff 0.5)
pred <- ifelse(pred.prob > 0.5, 1, 0)

# Test Error Rate
TER2.mod <- mean(pred != dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.3304196
```

```r
TER2 <- rbind(TER2, data.frame(method = "lasso regression", training.error.rate = TER2.mod))
```

```r
rm(cv.out,model,ridge.mod,std.cof,X,lambda.grid,TER2.mod,y,std.coeff,pred,pred.prob,poly.model)
```

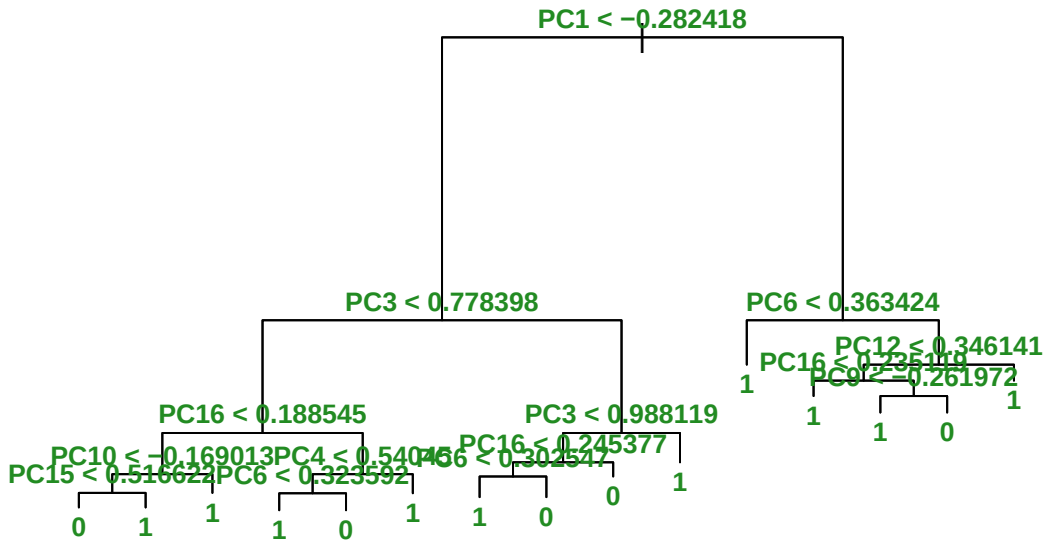
Regression Tree (best)

```r
model <- tree::tree(as.factor(spam) ~ . , dtm.pc.train)

summary(model)
```

```
##
## Classification tree:
## tree::tree(formula = as.factor(spam) ~ ., data = dtm.pc.train)
## Variables actually used in tree construction:
## [1] "PC1"  "PC3"  "PC16" "PC10" "PC15" "PC4"  "PC6"  "PC12" "PC9"
## Number of terminal nodes:  15
## Residual mean deviance:  0.7186 = 945 / 1315
## Misclassification error rate: 0.1459 = 194 / 1330
```

Tree to prune:

```r
plot(model)

# Aggiungi testo migliorato
text(model,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```



Unpruned tree

```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1958042
```

Searching the best point to prune the tree

```
set.seed(1)

model.cv= tree::cv.tree(model,
                        FUN= prune.misclass)
```

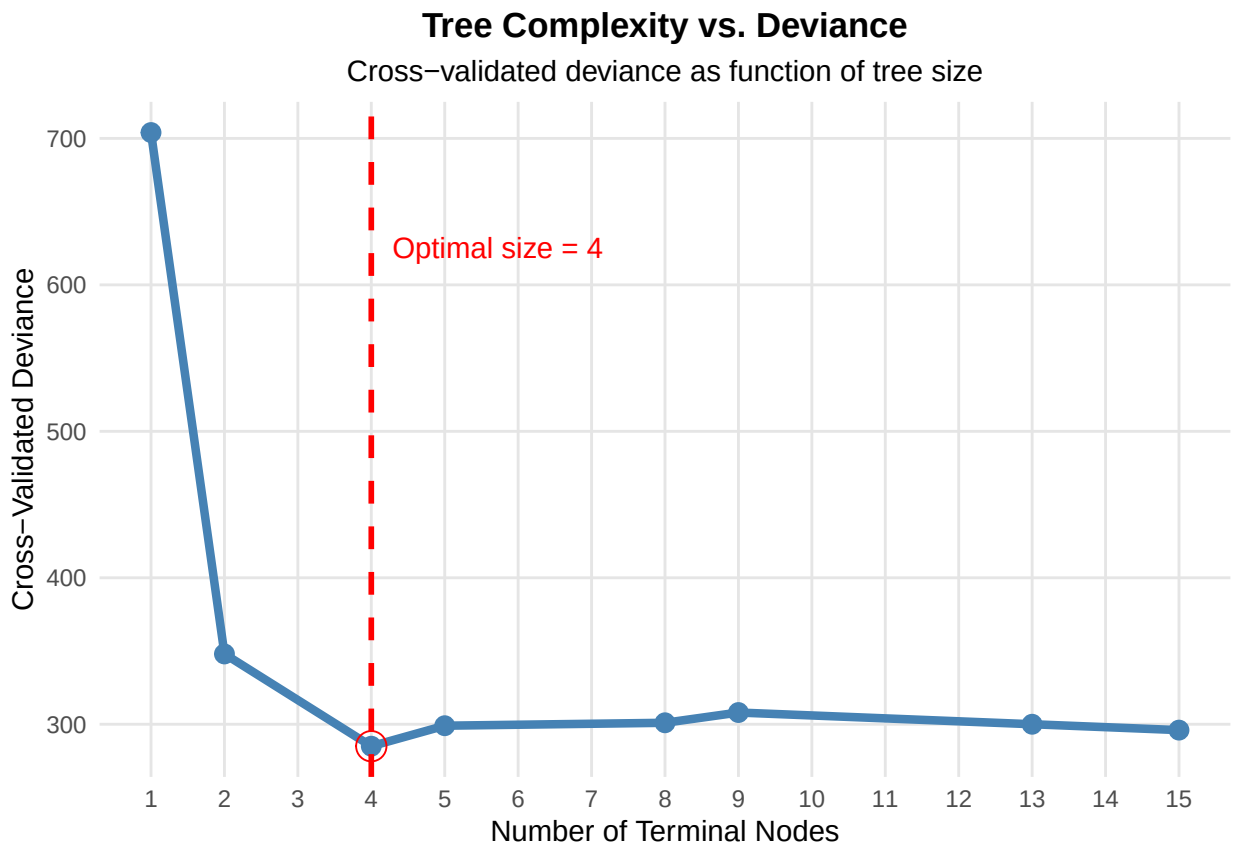
Optimal size of the tree:

```
# Trova la posizione del minimo errore
min_dev_index <- which.min(model.cv$dev)

# Numero ottimale di nodi corrispondente al minimo errore
optimal_size <- model.cv$size[min_dev_index]
optimal_size
```

```
## [1] 4
```

Number of terminal nodes

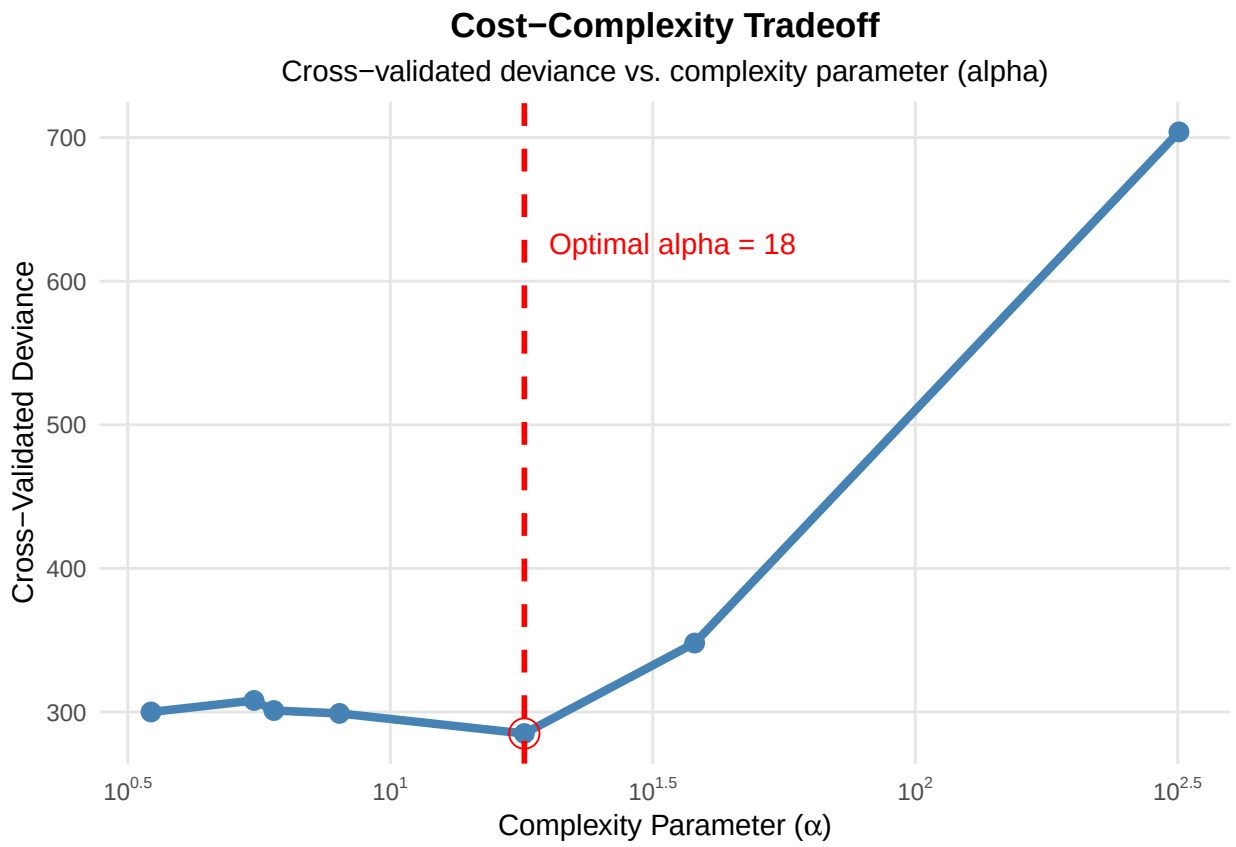
```
cv_data <- data.frame(
  Size = model.cv$size,       # Numero di nodi terminali
  Deviance = model.cv$dev      # Devianza
)

# Improved plot with ggplot2
library(ggplot2)
ggplot(cv_data, aes(x = Size, y = Deviance)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  geom_point(color = "steelblue", size = 3, shape = 19) +
  geom_vline(xintercept = optimal_size,
             color = "red",
             linetype = "dashed",
             linewidth = 1) +
  geom_point(data = subset(cv_data, Size == optimal_size),
             color = "red",
             size = 5,
             shape = 1) +
  annotate("text",
           x = optimal_size,
           y = max(cv_data$Deviance)-100,
           label = paste("Optimal size =", optimal_size),
           color = "red",
           vjust = -1,
           hjust = -0.1) +
  scale_x_continuous(breaks = seq(min(cv_data$Size),
                                  max(cv_data$Size),
                                  by = 1)) +
  labs(title = "Tree Complexity vs. Deviance",
       subtitle = "Cross-validated deviance as function of tree size",
       x = "Number of Terminal Nodes",  # Fixed typo in "Terminal"
       y = "Cross-Validated Deviance") +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```



Value of the cost-complexity parameter

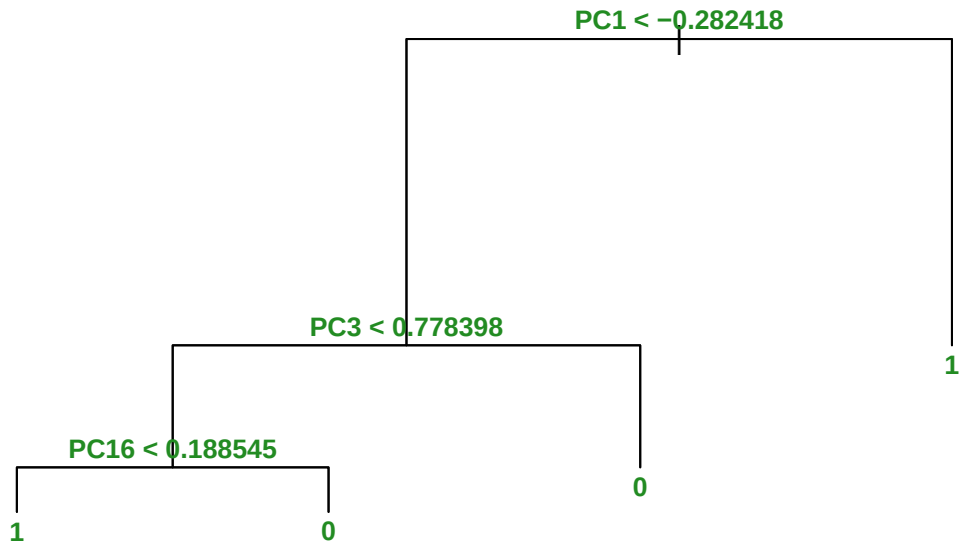
```r
# Create a data frame for plotting
ccp_data <- data.frame(
  Alpha = model.cv$k,
  Deviance = model.cv$dev
)

# Find the optimal alpha value
optimal_alpha <- model.cv$k[which.min(model.cv$dev)]

# Enhanced plot with ggplot2
library(ggplot2)
ggplot(ccp_data, aes(x = Alpha, y = Deviance)) +
  geom_line(color = "steelblue", linewidth = 1.5) +
  geom_point(color = "steelblue", size = 3) +
  geom_vline(xintercept = optimal_alpha,  # CORRETTO: xintercept invece di xintercept
             color = "red",
             linetype = "dashed",
             linewidth = 1) +
  geom_point(data = subset(ccp_data, Alpha == optimal_alpha),
             color = "red",
             size = 5,
             shape = 1) +
  annotate("text",
           x = optimal_alpha,
```

```
          y = max(ccp_data$Deviance)-100,
          label = paste("Optimal alpha =", round(optimal_alpha, 4)),
          color = "red",
          vjust = -1,
          hjust = -0.1) +
  scale_x_continuous(trans = 'log10',
                     breaks = scales::trans_breaks("log10", function(x) 10^x),
                     labels = scales::trans_format("log10", scales::math_format(10^.x))) +
  labs(title = "Cost-Complexity Tradeoff",
       subtitle = "Cross-validated deviance vs. complexity parameter (alpha)",  # CORRETTO: parameter i
       x = expression(paste("Complexity Parameter (", alpha, ")")),  # CORRETTO: Parameter invece di Pa
       y = "Cross-Validated Deviance") +
  theme_minimal() +
  theme(
    panel.grid.major = element_line(color = "gray90"),
    panel.grid.minor = element_blank(),
    plot.title = element_text(face = "bold", hjust = 0.5),
    plot.subtitle = element_text(hjust = 0.5)
  )
```




```
library(tree)
model.pruned = tree::prune.misclass(model,
                                    best = optimal_size)
```

```
plot(model.pruned)
# Aggiungi testo migliorato
text(model.pruned,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model.pruned, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1958042
```

The test error rate is the same but the model is simplier.

```
TER2 <- rbind(TER2, data.frame(method = "classification tree best", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model,model.cv,model.pruned,min_dev_index,optimal_size,optimal_alpha,ccp_data,cv_data)
```

Regression Tree (simplier)

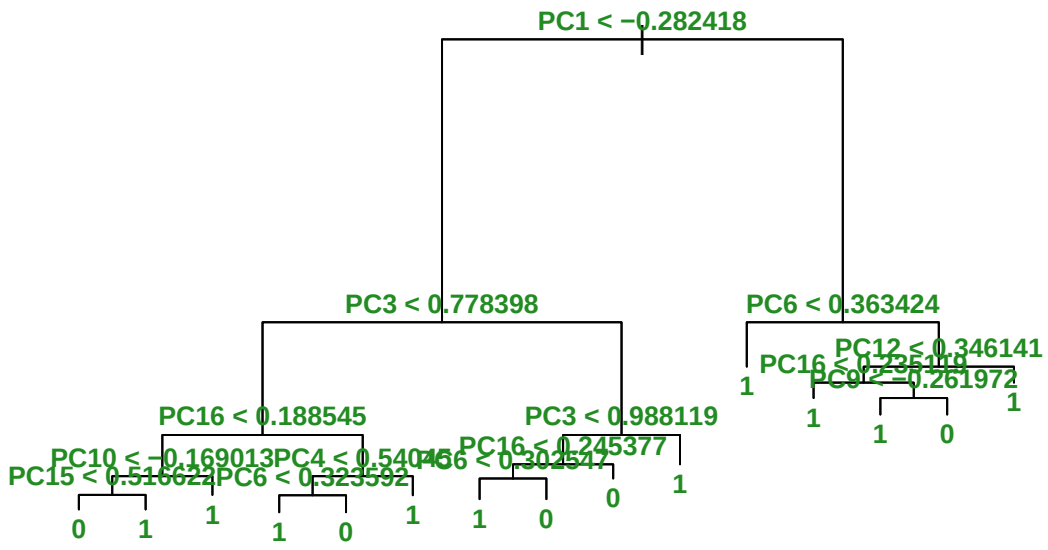
We can further prune the tree: if the plot shows that the error remains similar for a smaller number of terminal nodes, we can opt for a more parsimonious model. Specifically, we select the smallest tree whose cross-validation error is within 1 standard error of the minimum.

```
model <- tree::tree(as.factor(spam) ~ . , dtm.pc.train)

summary(model)
```

```
##
## Classification tree:
## tree::tree(formula = as.factor(spam) ~ ., data = dtm.pc.train)
## Variables actually used in tree construction:
## [1] "PC1"  "PC3"  "PC16" "PC10" "PC15" "PC4"  "PC6"  "PC12" "PC9"
## Number of terminal nodes:  15
## Residual mean deviance:  0.7186 = 945 / 1315
## Misclassification error rate: 0.1459 = 194 / 1330
```

Tree to prune:

```
plot(model)
text(model,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1958042
```

```
set.seed(1)

model.cv= tree::cv.tree(model,
                        FUN= prune.misclass)
```

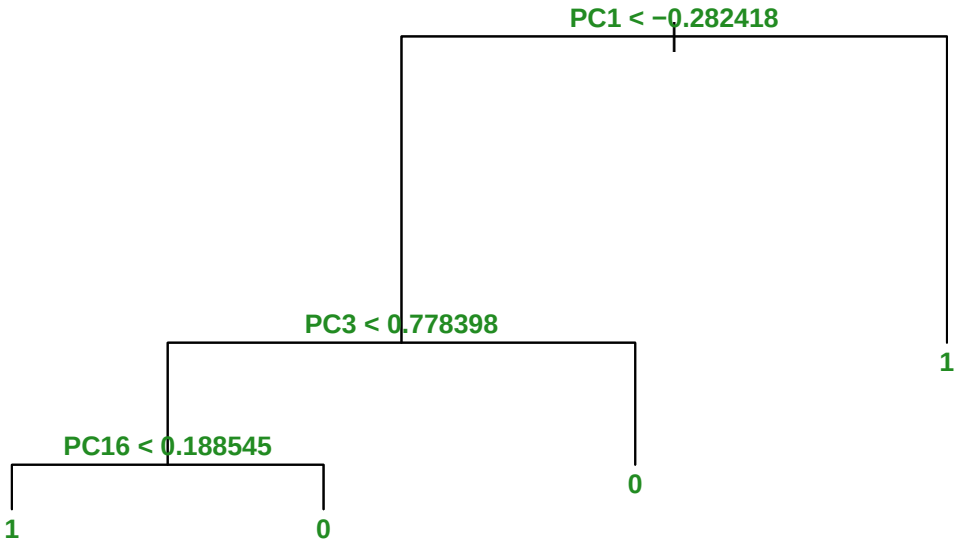
```
# Calcola la devianza minima e la sua soglia "1 SE"
min_dev <- min(model.cv$dev)
se_threshold <- min_dev + sd(model.cv$dev) / sqrt(length(model.cv$dev))

# Trova il modello più semplice con errore <= soglia
simpler_size <- min(model.cv$size[model.cv$dev <= se_threshold])
simpler_size
```

```
## [1] 4
```

The best model and the 1-SE model are the same.

```
library(tree)
model.pruned = tree::prune.misclass(model,
                                    best = simpler_size)
```

```
plot(model.pruned)
text(model.pruned,
     digits = 3,
     cex = 0.85,
     col = "forestgreen",
     font = 2,
     use.n = TRUE,
     fancy = TRUE,
     bg = rgb(0.95, 0.95, 0.85))
```




```
pred= predict(model.pruned, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1958042
```

The error rate is a little bit higher, but the model is much simpler (from 10 to 3).

```
TER2 <- rbind(TER2, data.frame(method = "classification tree 1se", training.error.rate = TER2.mod))
```

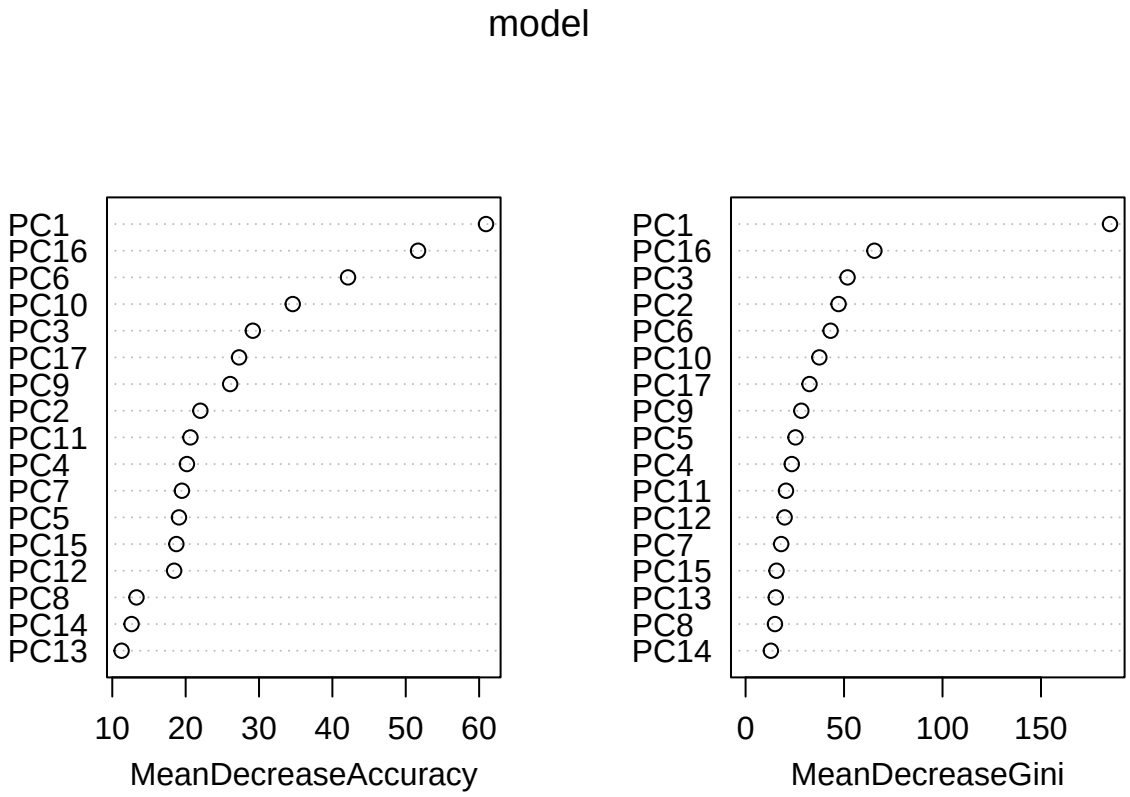
```
rm(pred,TER2.mod,model,model.cv,model.pruned, min_dev, simpler_size, se_threshold)
```

Bagging

```
library(randomForest)
set.seed(1)

model= randomForest(as.factor(spam) ~ . , dtm.pc.train,
                    mtry=ncol(dtm.pc.train)-1, importance=T)
model
```

```
##
## Call:
##  randomForest(formula = as.factor(spam) ~ ., data = dtm.pc.train,      mtry = ncol(dtm.pc.train) - 1
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 17
##
##         OOB estimate of  error rate: 14.51%
## Confusion matrix:
##     0   1 class.error
## 0 564 101   0.1518797
## 1  92 573   0.1383459
```

```
varImpPlot(model)
```




```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1293706
```

```
TER2 <- rbind(TER2, data.frame(method = "bagging", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model,model.cv,model.pruned)
```

Random Forest

```
library(randomForest)
set.seed(1)

model= randomForest(as.factor(spam) ~ . , dtm.pc.train,
                    importance=T)
model
```

```
##
## Call:
```

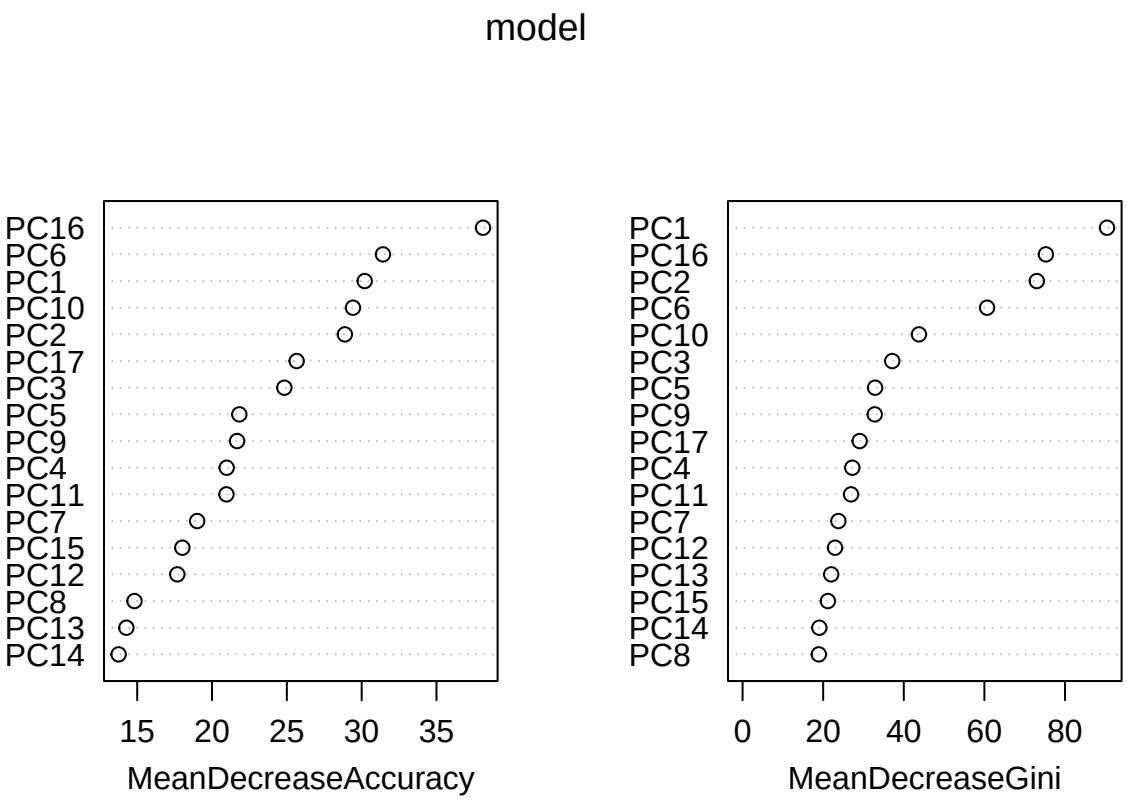
```
##  randomForest(formula = as.factor(spam) ~ ., data = dtm.pc.train,      importance = T)
##                Type of random forest: classification
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##         OOB estimate of  error rate: 14.14%
## Confusion matrix:
##     0   1 class.error
## 0 569  96   0.1443609
## 1  92 573   0.1383459
```

```
pred= predict(model, dtm.pc.test, type="class")
```

```
TER2.mod=mean(pred!=dtm.pc.test$spam)
TER2.mod
```

```
## [1] 0.1171329
```

```
varImpPlot(model)
```




```
TER2 <- rbind(TER2, data.frame(method = "random forest", training.error.rate = TER2.mod))
```

```
rm(pred,TER2.mod,model,model.cv,model.pruned)
```

Comparison

```
TER <- cbind(TER, TER2[, 2, drop = FALSE])


colnames(TER)[2] <- "ter unproc dtm"
colnames(TER)[3] <- "ter proc dtm"

rm(TER2)
TER
```

```
##                             method ter unproc dtm ter proc dtm
## 1                     linear model      0.3863636    0.3653846
## 2                   logistic model      0.3409091    0.3076923
## 3     linear discriminant analisis      0.3863636    0.3653846
## 4  quadratic discriminant analisis      0.3916084    0.3618881
## 5                      naive bayes      0.4090909    0.3811189
## 6                              KNN      0.1625874    0.1520979
## 7                 ridge regression      0.3479021    0.3304196
## 8                 lasso regression      0.3461538    0.3304196
## 9         classification tree best      0.1923077    0.1958042
## 10         classification tree 1se      0.2272727    0.1958042
## 11                        bagging      0.1311189    0.1293706
## 12                  random forest      0.1311189    0.1171329
```

Almost all classification models perform better when using the processed DTM. This suggests that the preprocessing choices, such as removing stopwords and filtering low-frequency terms, were effective, even though we later applied Principal Component Analysis on top of the DTM.

Regarding model performance, tree based models and KNN classification achieved the best results in classifying YouTube comments as spam or not.

In particular, the best performing model was a Random Forest, which is an ensemble method that improves upon individual decision trees by averaging multiple trees built on bootstrap samples and random subsets of features. In this case, the test error rate was:

```
TER[12,3]
```

```
## [1] 0.1171329
```

Another model that performed well, as previously mentioned, was K Nearest Neighbors, specifically the 6-nearest neighbors classifier. As the name suggests, it classifies a new observation based on the majority class (mode) among its six closest neighbors in the space of Xs.

The test error rate for this classifier was:

```
TER[6,3]
```

```
## [1] 0.1520979
```

The ‘TER’ data frame is extracted in Python for further comparison. Uncomment the code if needed.

```
#write.csv(TER, "C://Users//paren//Desktop//_MAGISTRALE//_Machine_Learning//progetto//TER.csv", row.nam
```