ASSIGNMENT 3

STAT 702 – Data Mining

SUMUKH SAGAR MANJUNATH

1. **Based on these data, construct a classification tree for predicting whether an email is "spam" based on other variables. Use 10-fold cross-validation with the 1-SE rule to find the optimal value of the complexity parameter.**

```
> library(rpart)
> col_names = read.csv("names.csv",header=F)
> spam = read.csv("spamdata.txt",header=F)
> names(spam) = sapply((1:nrow(col_names)),function(i) toString(col_names[i,1
]))
> spam$is_spam = factor(spam$is_spam, levels=0:1, labels=c("not_spam", "spam"
))
> is.factor(spam$is_spam)
[1] TRUE
> set.seed(1)
> my.control = rpart.control(xval=10,cp=0)
> cfit = rpart(is_spam ~ ., data=spam, method="class", control = my.control)
> plotcp(cfit)
```

```
> unpruned_spam = printcp(cfit)

Classification tree:
rpart(formula = is_spam ~ ., data = spam, method = "class", control = my.control)

Variables actually used in tree construction:
 [1] capital_run_length_average capital_run_length_longest capital_run_length_total   char_freq_!
 [5] char_freq_$                char_freq_(                char_freq_;                word_freq_1999
 [9] word_freq_650              word_freq_address          word_freq_data             word_freq_edu
[13] word_freq_email            word_freq_font             word_freq_free             word_freq_george
[17] word_freq_hp               word_freq_internet         word_freq_money            word_freq_our
[21] word_freq_over             word_freq_re               word_freq_remove           word_freq_technology
[25] word_freq_will             word_freq_you              word_freq_your

Root node error: 1813/4601 = 0.39404

n= 4601

           CP nsplit rel error  xerror     xstd
1  0.47655819      0   1.00000 1.00000 0.018282
2  0.14892443      1   0.52344 0.55378 0.015453
3  0.04302261      2   0.37452 0.45615 0.014366
4  0.03088803      4   0.28847 0.30888 0.012232
5  0.01047987      5   0.25758 0.27910 0.011705
6  0.00827358      6   0.24710 0.26751 0.011489
7  0.00717044      7   0.23883 0.25924 0.011331
8  0.00529509      8   0.23166 0.24986 0.011147
9  0.00441258     14   0.19581 0.23607 0.010867
10 0.00358522     15   0.19140 0.22780 0.010694
11 0.00275786     19   0.17705 0.22339 0.010600
12 0.00257400     22   0.16878 0.21622 0.010445
13 0.00220629     25   0.16106 0.21125 0.010335
14 0.00211436     27   0.15665 0.21125 0.010335
15 0.00165472     33   0.14396 0.21236 0.010360
16 0.00110314     36   0.13900 0.20629 0.010224
17 0.00082736     43   0.13127 0.20243 0.010136
18 0.00055157     47   0.12796 0.20188 0.010124
19 0.00036771     53   0.12466 0.20463 0.010187
20 0.00000000     62   0.12135 0.20574 0.010212
```

```
> unpruned_spam = as.data.frame(unpruned_spam)
> oneSE_xerr = min(unpruned_spam$xerror) + unpruned_spam$xstd[unpruned_spam$x
error == min(unpruned_spam$xerror)]
> oneSE_xerr
[1] 0.2119991
> optim_cp = max(unpruned_spam$CP[unpruned_spam$xerror < oneSE_xerr])
> optim_cp
[1] 0.002206288
```

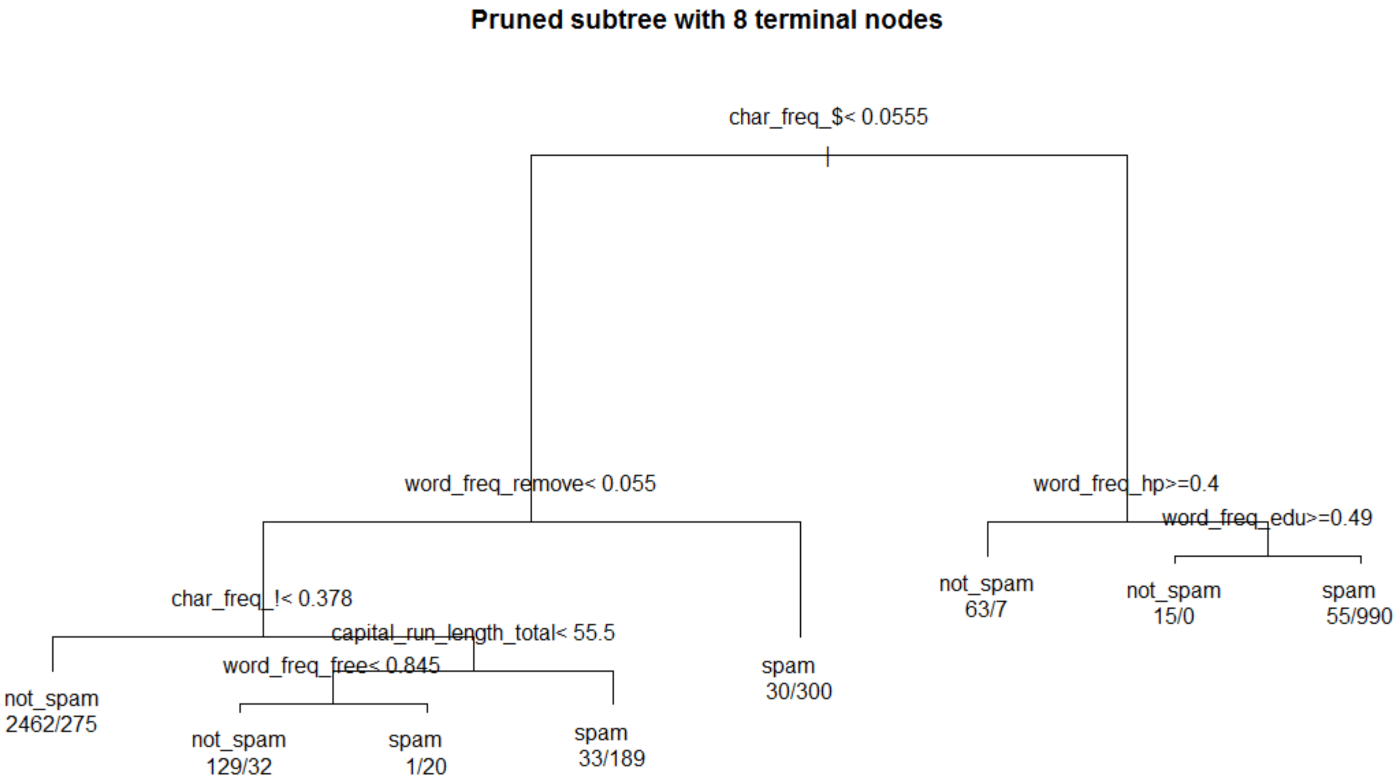
What's your estimate of the misclassification rate of the optimal tree? What are the false positive and false negative error rates?

```
> pruned_spam = prune(cfit, cp = optim_cp)
> predicted_spam = predict(pruned_spam, type="vector") - 1
> observed_spam = as.numeric(spam$is_spam) - 1
> missclass = dim(spam[(predicted_spam != observed_spam),])[1]
> missclass
[1] 292
> total_obs = dim(spam)[1]
> total_obs
[1] 4601
> missclass_rate = (missclass / total_obs)* 100
> missclass_rate
[1] 6.346446
> false_positive = dim(spam[predicted_spam == 1 & observed_spam == 0,])[1]
> false_positive
[1] 105
> false_negative = dim(spam[predicted_spam == 0 & observed_spam == 1,])[1]
> false_negative
[1] 187
> yes_spam = dim(spam[spam$is_spam == "spam",])[1]
> yes_spam
[1] 1813
> no_spam = dim(spam[spam$is_spam == "not_spam",])[1]
> no_spam
[1] 2788
> false_positive_rate = false_positive / no_spam
> false_positive_rate
[1] 0.03766141
> false_negative_rate = false_negative / yes_spam
> false_negative_rate
[1] 0.103144
```

How many terminal nodes does your optimal tree have?

```
> terminal_nodes = unpruned_spam$nsplit[unpruned_spam$CP==optim_cp] + 1
> terminal_nodes
[1] 26
```

Plot the optimal tree. If it is too large, plot a subtree of the optimal tree that has at most 8 terminal nodes.

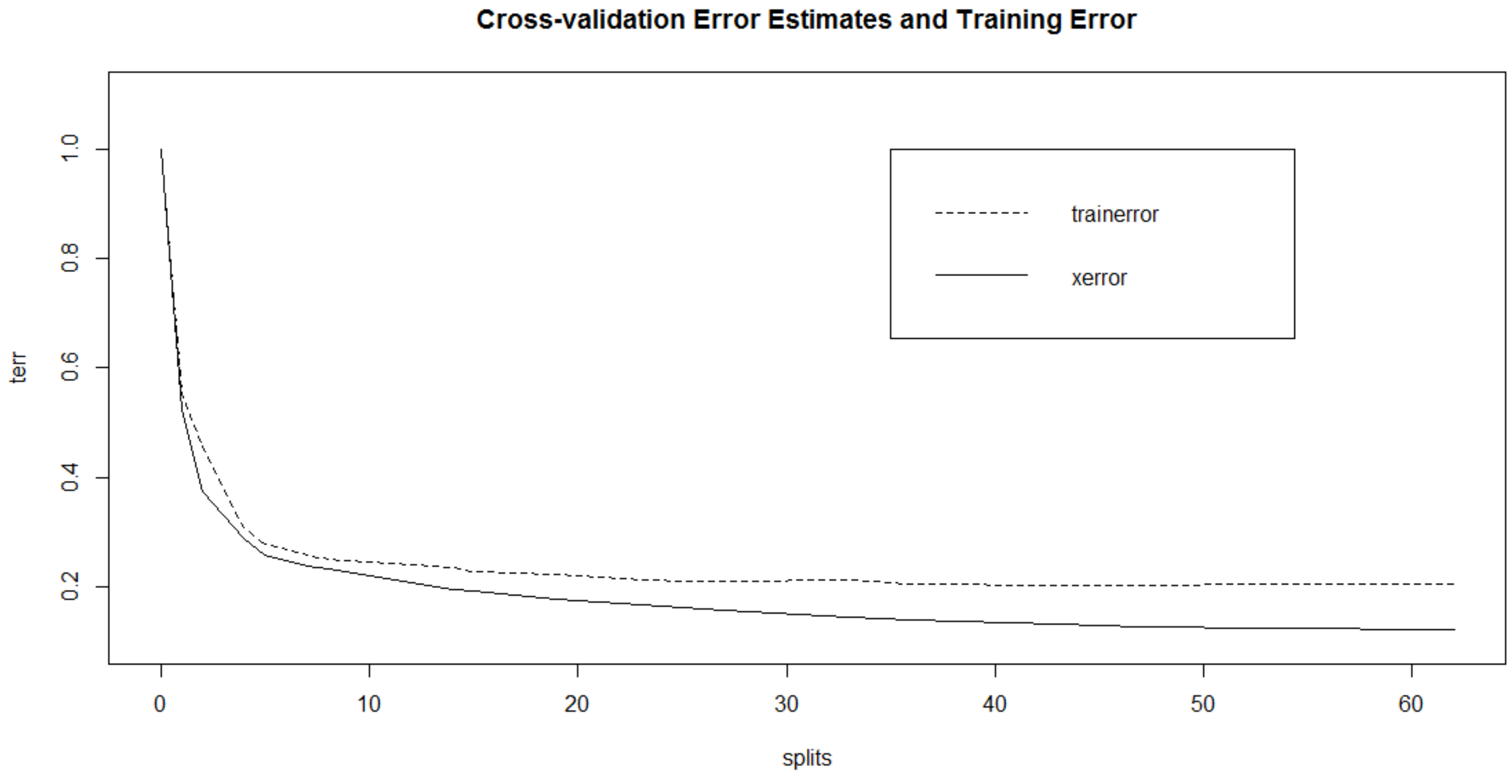


What are some of the variables that were used in tree construction?

```
Variables actually used in tree construction:
 [1] capital_run_length_average capital_run_length_longest capital_run_length_total   char_freq_!
 [5] char_freq_$                char_freq_(                char_freq_;                word_freq_1999
 [9] word_freq_650              word_freq_address          word_freq_data             word_freq_edu
[13] word_freq_email            word_freq_font             word_freq_free             word_freq_george
[17] word_freq_hp               word_freq_internet         word_freq_money            word_freq_our
[21] word_freq_over             word_freq_re               word_freq_remove           word_freq_technology
[25] word_freq_will             word_freq_you              word_freq_your
```

Plot cross validation estimates of errors and training errors of the sequence of pruned trees against the trees' complexity (i.e. number of splits). Compare the two curves (one based on cross validated errors and the other based on training errors.)

```
> splits <- cfit$cptable[, 2]
> terr <- cfit$cptable[, 3]
> xerr <- cfit$cptable[, 4]
> plot(splits, terr, ylim=c(0.1, 1.1), type="l")
> lines(splits, xerr, lty=2)
> title("Cross-validation Error Estimates and Training Error")
```



From the plot above, we see that the Cross validation error decreases more drastically than the training error. So, cross validation error is a better choice for pruning.

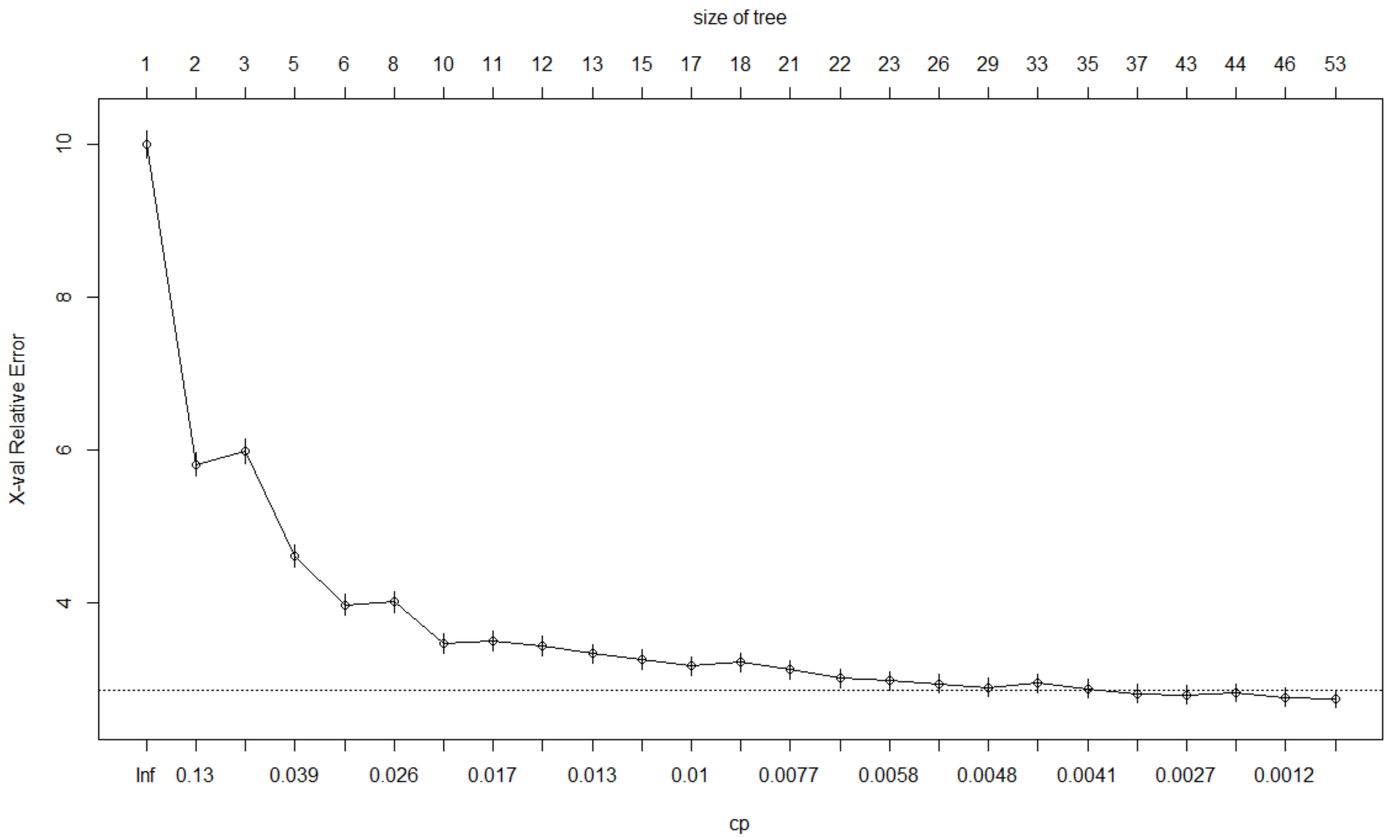
2. Your classifier in part (1) can be used as a spam filter. One of the possible disadvantages of such a spam filter is that it might filter out too many good (non-spam) emails. Therefore, a better spam filter might be the one which penalizes false positive errors more heavily than false negative errors.

```
> library(rpart)
> col_names = read.csv("names.csv",header=F)
> spam = read.csv("spamdata.txt",header=F)
> names(spam) = sapply((1:nrow(col_names)),function(i) toString(col_names[i,1
]))
> spam$is_spam = factor(spam$is_spam, levels=0:1, labels=c("not_spam", "spam"
))
> is.factor(spam$is_spam)
[1] TRUE
> set.seed(1)
> lmat = matrix(c(0,1,10,0), nrow=2, byrow=F)
> my.control = rpart.control(xval=10,cp=0)
> cfit1 = rpart(is_spam ~ ., data=spam, method="class", control = my.control,
parms=list(loss=lmat))
> unpruned_spam = printcp(cfit1)
> unpruned_spam = as.data.frame(unpruned_spam)
```

```
> unpruned_spam = printcp(cfit1)

Classification tree:
rpart(formula = is_spam ~ ., data = spam, method = "class", parms = list(loss = lmat),
    control = my.control)

Variables actually used in tree construction:
 [1] capital_run_length_average capital_run_length_longest capital_run_length_total
 [4] char_freq_!                char_freq_#                char_freq_$
 [7] char_freq_(                char_freq_;                word_freq_000
[10] word_freq_650              word_freq_address          word_freq_all
[13] word_freq_edu              word_freq_font             word_freq_free
[16] word_freq_george           word_freq_hp               word_freq_internet
[19] word_freq_money            word_freq_order            word_freq_our
[22] word_freq_people           word_freq_project          word_freq_re
[25] word_freq_remove           word_freq_technology       word_freq_you
[28] word_freq_your

Root node error: 1813/4601 = 0.39404
```


```
Root node error: 1813/4601 = 0.39404
```


```
n= 4601

           CP nsplit rel error  xerror    xstd
1  0.18422504      0   1.00000 10.0000 0.18282
2  0.08825152      1   0.81577  5.8097 0.15682
3  0.05681191      2   0.72752  5.9801 0.15851
4  0.02647546      4   0.61390  4.6056 0.14382
5  0.02619967      5   0.58742  3.9691 0.13543
6  0.02482074      7   0.53502  4.0072 0.13598
7  0.01765030      9   0.48538  3.4732 0.12809
8  0.01599559     10   0.46773  3.4997 0.12852
9  0.01434087     11   0.45174  3.4280 0.12740
10 0.01185880     12   0.43740  3.3309 0.12581
11 0.01047987     14   0.41368  3.2537 0.12455
12 0.00992830     16   0.39272  3.1726 0.12316
13 0.00827358     17   0.38279  3.2173 0.12390
14 0.00717044     20   0.35797  3.1197 0.12222
15 0.00606729     21   0.35080  3.0116 0.12031
16 0.00551572     22   0.34473  2.9746 0.11963
17 0.00496415     25   0.32763  2.9371 0.11894
18 0.00468836     28   0.31274  2.8935 0.11815
19 0.00441258     32   0.28682  2.9432 0.11904
20 0.00386100     34   0.27799  2.8742 0.11774
21 0.00330943     36   0.27027  2.8114 0.11652
22 0.00220629     42   0.24600  2.7954 0.11622
23 0.00165472     43   0.24379  2.8229 0.11673
24 0.00091929     45   0.24049  2.7639 0.11560
25 0.00000000     52   0.23331  2.7402 0.11507

> oneSE_xerr = min(unpruned_spam$xerror) + unpruned_spam$xstd[unpruned_spam$x
error == min(unpruned_spam$xerror)]
> optim_cp = max(unpruned_spam$CP[unpruned_spam$xerror < oneSE_xerr])
> optim_cp
[1] 0.003309432
```



What's your estimate of the misclassification rate of the optimal tree? What are the false positive and false negative error rates?

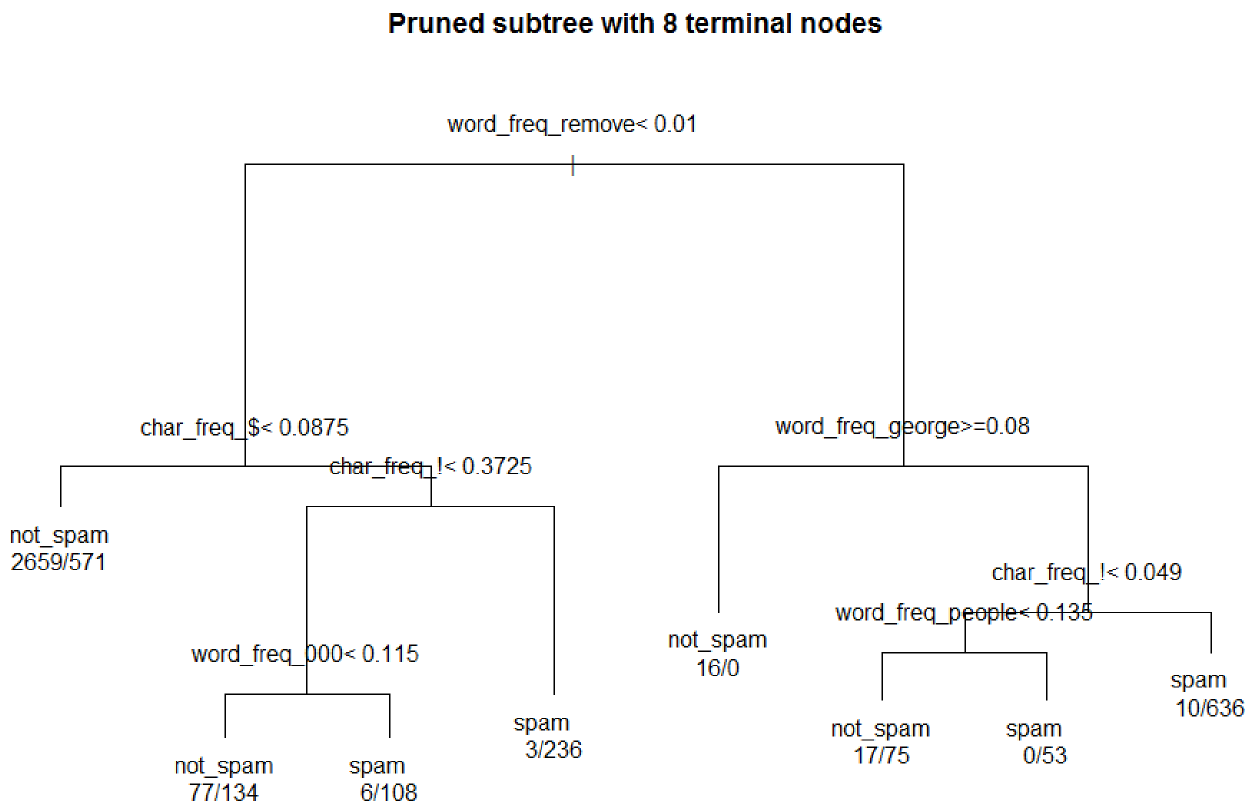
```
> pruned_spam = prune(cfit1, cp = optim_cp)
> predicted_spam = predict(pruned_spam, type="vector") - 1
> observed_spam = as.numeric(spam$is_spam) - 1
> missclass = dim(spam[(predicted_spam != observed_spam),])[1]
> missclass
[1] 445
> total_obs = dim(spam)[1]
> total_obs
[1] 4601
> missclass_rate = (missclass / total_obs)* 100
> missclass_rate
[1] 9.67181
> false_positive = dim(spam[predicted_spam == 1 & observed_spam == 0,])[1]
> false_positive
[1] 5
> false_negative = dim(spam[predicted_spam == 0 & observed_spam == 1,])[1]
> false_negative
[1] 440
> yes_spam = dim(spam[spam$is_spam == "spam",])[1]
> yes_spam
[1] 1813
> no_spam = dim(spam[spam$is_spam == "not_spam",])[1]
> no_spam
[1] 2788
> false_positive_rate = false_positive / no_spam
> false_positive_rate
[1] 0.0017934
```

```
> false_negative_rate = false_negative / yes_spam
> false_negative_rate
[1] 0.2426917
```

How many terminal nodes does your optimal tree have?

```
> terminal_nodes = unpruned_spam$nsplit[unpruned_spam$CP==optim_cp] + 1
> terminal_nodes
[1] 37
```

Plot the optimal tree. If it is too large, plot a subtree of the optimal tree that has at most 8 terminal nodes.

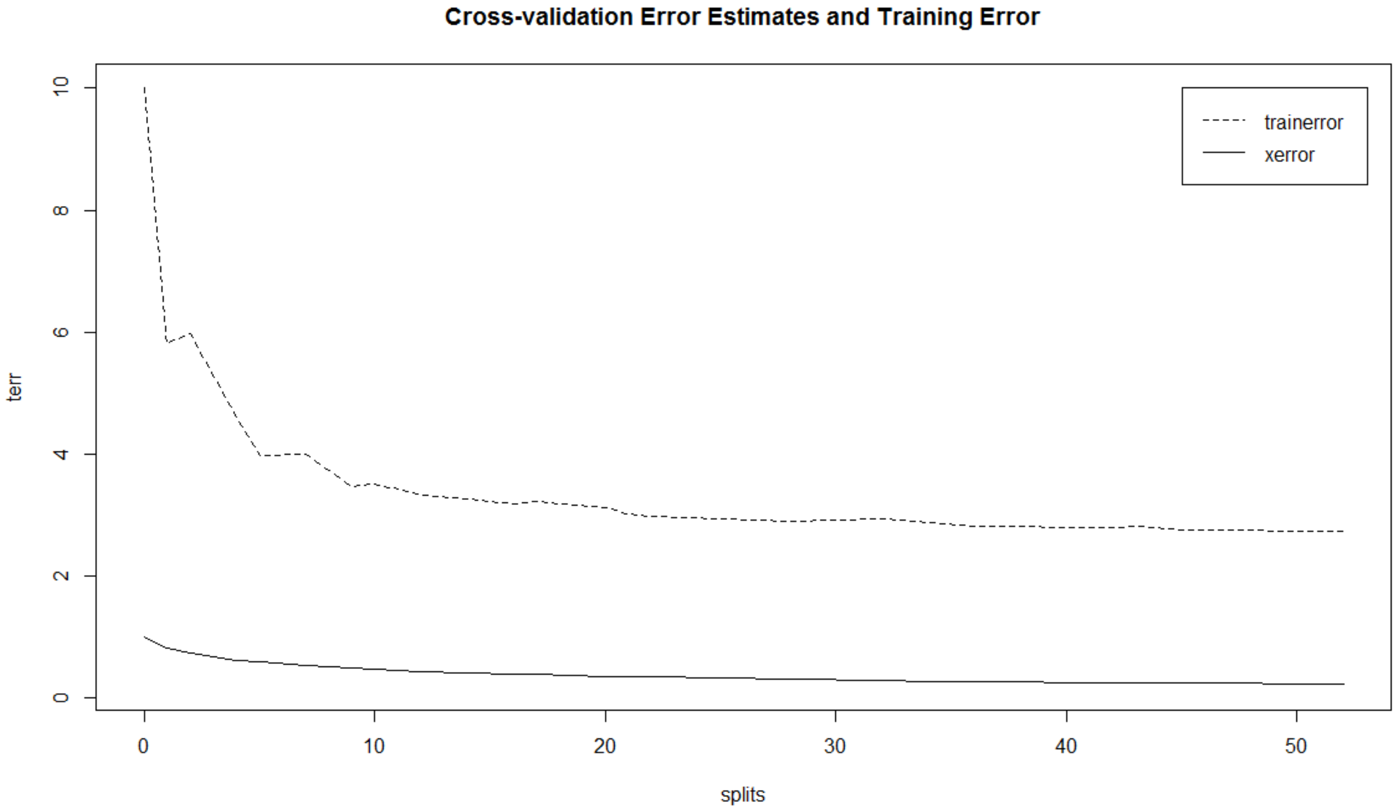



```
> pruned_at_8 = prune(cfit1, cp=unpruned_spam$CP[unpruned_spam$nsplit==7])
> plot(pruned_at_8, margin=0.1)
> text(pruned_at_8, use.n=T)
> title("Pruned subtree with 8 terminal nodes")
```

What are some of the variables that were used in tree construction?

```
Variables actually used in tree construction:
 [1] capital_run_length_average capital_run_length_longest capital_run_length_total   char_freq_!
 [5] char_freq_#                char_freq_$                char_freq_(                char_freq_;
 [9] word_freq_000              word_freq_650              word_freq_address          word_freq_all
[13] word_freq_edu              word_freq_font             word_freq_free             word_freq_george
[17] word_freq_hp               word_freq_internet         word_freq_money            word_freq_order
[21] word_freq_our              word_freq_people           word_freq_project          word_freq_re
[25] word_freq_remove           word_freq_technology       word_freq_you              word_freq_your
```

Plot cross validation estimates of errors and training errors of the sequence of pruned trees against the trees' complexity (i.e. number of splits). Compare the two curves (one based on cross validated errors and the other based on training errors.)



From the plot above, we see that the Cross validation error decreases more drastically than the training error. So, cross validation error is a better choice for pruning.

Which of the two classifiers would you prefer to use as a spam filter and why?
Classifier with the loss matrix is favored because it has lower false positive error meaning chances of falsely classifying a genuine mail as spam is less. This is a desirable expectation from a spam filter since classifying a non-spam mail as spam is more severe than classifying spam mails as non-spam.