STAT250-HW3

Yu Pei
SID999438479

1 Introduction

This is one of the questions in Kaggle.We have the whole data set from the beginning to Sep,2012, including postid, postcreationdate, reputation, the content of question, tags and the outcome–whether the post is closed and the closed type. Our goal is to use the existing data to come up with a classification model that can be used to predict whether a post will be closed in the future.

2 Boosting Implementation

In order to better understand the algorithms that we are using, I implement the Adaboost.M1 algorithms with minimum tuning parameters. I used the one described in Algorithm 10.1 from ESL, and using a single tree as base learner. Using the same testing data as described in ESL, we can cut down the error rate dramatically.

```
mybo = fit.ada(train, test[, -1])

## Loading required package:  rpart

table(y.test, mybo)  # boost with 100 iter

##       mybo
## y.test   -1    1
##     -1 4818   87
##     1    82 5013

models = rpart(y ~ ., data = train, method = "class")
singletree = predict(models, test, type = "class")
table(y.test, singletree)  #single run

##       singletree
## y.test   -1    1
##     -1 4661  244
##     1   356 4739
```

If we add some code to store the cumulative boosting result, we can get a similar plot as Figure10.2 in ESL.

3 Data exploration

This part I didn't do much. Since the questions are closed manually, we can assume certain patterns about a question will be more likely end up being closed. Like reputation, owner creation date to post creation date, tags numbers,the length of the bodys, certain symbols like '!' etc. And besides that we can extract information from the content of text, some words might violate the rules in Stack-overflow so will caused the post being closed.

From the web I found the source code from the 10th competitor, he use basically the same set of features.

4 NLP Processing

The processing using tm package in R is reasonably fast.With around 14,000 entries, it can complete within a minute. The conversion from markdown format to extract the html ¡p¿ tags involves 2 packages,markdown and XML, but it also is quite fast. But the memory consumption is a potential issue when dealing with the whole data set.

Also as the previously mentioned author,I didn't do much about the source code content besides taking the length of the code. But for the paragraph content, we can use the build-in capacity to do part of speech and other kind of preprocessing in a dummy way. This process is not perfect, we get a lot of words stick together, thus loosing some information. In the end, we end up with a sparse matrix with word counts for each entry for each word. Then we proceed to model building.

5 Statistical/ML algorithms Classification

As the features grow larger, it becomes much harder to train a sophisticated model in a single machine. THere are several difficulties in experimenting in this part, some models have huge amount of tuning parameter, which make it harder to adjust. Second, the easiest way to speed up the training is to seek existing parallelized implementation. I compromised with the existing tools in R and small data set to see how well I can get with small data set. I trained the following random forest with randomForest package with 500 word feature in 12 hours. The testing error rate is around 25%–33%. One thing I want to utilize is the population ratio between open and closed posts.But I can't figure out how to incoporate that into the models.

```r
load("myforest.rda")  #the fitted model
myforest$confusion
```

```
##       0     1 class.error
## 0 47004 23132      0.3298
## 1 17929 52207      0.2556

myforest$ntree

## [1] 500
```

In conclusion, there are a lot can be done and this whole process is full of excitement.

6 Appendix source code

```
#This is the scripts to analyse the stack-overflow question in Kaggle.
#It consist of
#1,converting the BodyMarkdown to html then parse out the text content
#2,perform basic NLP analysis using tm package in R
#3, statistical/ machine learning methods to predict closed questions
#Note, only the training-sample.csv is used in here.

library(markdown)
library(XML)
library(tm)


#from markdown package, fragment.only
#to exclude the CSS etc. to save memory space.

train = read.csv('train-sample.csv',
colClasses=c(rep('NULL',7),'character',rep('NULL',7)))

train = unlist(train)
tohtml = sapply(train,function(x)markdownToHTML(text=x,fragment.only=TRUE))

library(parallel) # try to parallel this converting process
 cl = makeCluster(4,'FORK')
 htmllist = clusterSplit(cl,tohtml)

getvec = function(htmls){
  sapply(htmls,function(html){
    parsed = htmlParse(html)
    # get all the paragraphs. Paste them together

    plist = getNodeSet(parsed,'//p')
    texts = sapply(plist,xmlValue)
    paste(texts,collapse='')
  })
}

result.list = clusterApply(cl, htmllist, getvec)
results = unlist(result.list) # 14000+ entries of strings

status = read.csv('train-sample.csv',
colClasses=c(rep('NULL',14), 'factor'))

status = unlist(status)
stopCluster(cl)
open = results[status=='open']
# simplify into a two class classification problem
closed = results[status != 'open']

####tm analysis

cb = c(opencorpus, closedcorpus)#create corpus from all the entries
cb = tm_map(cb, tolower)#preprocessing, Somehow the stemming doesn't work
cb = tm_map(cb, removeWords, stopwords('english'))
cb = tm_map(cb, removePunctuation)
cb = tm_map(cb, stripWhitespace)
dtm = DocumentTermMatrix(cb,control=list(wordLengths=c(3,20), #Tfidf
              removeNumbers=TRUE, weighting=weightTfIdf,minDocFreq=30))

dtm2 = DocumentTermMatrix(cb,control=
        list(wordLengths=c(3,20), #just term freq
             removeNumbers=TRUE, weighting=weightTf,minDocFreq=30))

dtm3 = removeSparseTerms(dtm2,0.99)
# cut down the sparsity, just keep some commom words

save(dtm2,dtm,dtm3,file='sparsetable.rda')

fac = as.factor(rep(c(1,0), each=70136))
#Fit a regular random forest using randomForest packages in R
#Takes more than 10 hours to finish a prediction with 500+ features.
```