HW 03 report

Yilun Zhang Stu ID: 999486337

3/5/2014

1 Data processing

First I look into train-sample.csv. The data contains 15 variables. But I think some of these provides no information in our question. The information provided by postId and userId can be shown by ReputationAtPostCreation and OwnerUndeletedAnswerCountAtPostTime. If we take them into account, the number of factor levels will be too large. There are 3 variables related to time, namely PostCreationDate, OwnerCreationDate, PostClosedDate. Individually, They are poor predictor of our problem. But How long(if any) it takes to close the question, the the user's age on SO really matters. If a question is closed in a very short time after it's post, it is more likely a "stupid" question. And a user who use SO for a long time is intend to ask questions whick can be anwsered. There are 5 tags. Each of them contain more than a thousand distinct types. If they are regarded as factors it will be meaningless, beacause most of them are actually appeared only one time. One way is to treat them as "words" and fit random froest logistic regression or doing TF/ITF in LDA OR QDA. Another is using the frequency the tag appears in the sample. Intuitively, if the question has same fetures with others, it's more likely to be stupid, because duplicate closed case is ruled out of out sample. For instance, a man asked a question about C. There are a lot of other questions taged "C" in the SO already, meaning anwsers of "common questions" are already exist. If this question is not duplicate with others, it's probably not a "common one", therefore is probably can't be a good one.

The difficult thing is Nature language processing. OpenNLP package in R is very powerful,

but also computational intensive. I firstly use markdownToHTML() function in R to convert BodyMarkdown to HTML(got this idea on piazza). Then use htmlParse() getNodeSet() xmlValue() to get plain text. Then use functions in openNLP package to generate word frequency table. Here is my code:

```
options( java.parameters = "-Xmx4g" )
library(openNLP)
library(NLP)
library(markdown)
library(XML)
invmarkdown<-sapply(data$BodyMarkdown,function(x)
  markdownToHTML(text=x,fragment.only=TRUE))   #convert to html
toText<-function(h)  #convert to plain text
{
  doc=htmlParse(h)
  node=getNodeSet(doc,path="//p")
  k=sapply(node,xmlValue)
paste(k,collapse=' ')
}
library(parallel)
cl <- makeCluster(detectCores(),"FORK")

bodytexts=clusterApply(invmarkdown,toText)
posTitles =
  function(txt)
  {
    str =as.String(txt)
    sent_ann = Maxent_Sent_Token_Annotator()
    s = sent_ann(str)
    word_ann = Maxent_Word_Token_Annotator()
    w = annotate(str, list(sent_ann, word_ann))
```

```
    pos_ann = Maxent_POS_Tag_Annotator()
    pos = annotate(str, pos_ann, w)

    i = pos$type == 'word'
    ww = str[pos][i]
    cbind(ww, unlist(pos$features[i]))
  }

getdf<-function(v){  #get frequency table
  a=clusterApply(cl, v, posTitles)
  for(i in 1:length(a)){
    j=a[[i]][,2] %in% c("JJ", "JJR", "JJS", "LS", "MD", "NN",
 #drop meaningless terms
                        "NNS", "NNP", "NNPS", "RBR", "RB", "VB",
                        "VBD", "VBG", "VBN", "VBP", "VBZ" )
    a[[i]]=a[[i]][j,]
  }
  for(i in 1:length(a)) a[[i]]= paste(a[[i]][,1],a[[i]][,2])
# paste word and its pos
  b=sapply(a,unique)
  b=unlist(b,use.names = F)
  m <- as.data.frame(setNames(replicate(length(b),numeric(0), simplify = F), b))
  for(i in 1:length(a)){
    tt=table(a[[i]])
    k=tt[b]
    m=rbind(m,k)
  }                                  #generae new data frame
  m[is.na(m)]=0
  m
```

```
}
```

The idea is using posTitles function from you to get words and their position, paste them as a new charactor. However, it seems take forever to do even a small sample of observations(1000). So what I do is just counting the words in the body and titles. If a question is expressed detailedly(more word), it's less possible to be closed. Code:

```
#data processing
setwd("/media/Windows8_OS/linuxdl")
data=read.csv("train-sample.csv",stringsAsFactors=F)
names(data)
options( java.parameters = "-Xmx4g" )
library(openNLP)
library(NLP)
library(markdown)
library(XML)
invmarkdown<-sapply(data$BodyMarkdown,function(x)
  markdownToHTML(text=x,fragment.only=TRUE))
toText<-function(h){
  doc=htmlParse(h)
  node=getNodeSet(doc,path="//p")
  k=sapply(node,xmlValue)
  paste(k,collapse=' ')
}
postday=as.Date(data$PostClosedDate,format=
'%m/%d/%Y')-as.Date(data$PostCreationDate,format='%m/%d/%Y')
head(postday)
userage=as.Date(data$PostCreationDate,format=
'%m/%d/%Y')-as.Date(data$OwnerCreationDate,format='%m/%d/%Y')
bodytexts=sapply(invmarkdown,toText)
bodyWordcount=sapply(strsplit(data$BodyMarkdown," "),length)
```

```
tileWordcout=sapply(strsplit(data$Title," "),length)
tagFreq<-function(v){
tt1=table(v)
stt1=sort(tt1,decreasing=T)
stt1[v]
}
names(bodytexts)
length(userage)
dat=data.frame(POSDAYS=postday,USERAGE=userage,REPUTANTION=
as.numeric(data$ReputationAtPostCreation),
              UNDELETTIME=as.numeric(data$OwnerUndeletedAnswerCountAtPostTime),
              TITLEWORDS=tileWordcout,BODYWORDS=bodyWordcount,
              TAG1FREQ=tagFreq(data$Tag1),
              TAG2FREQ=tagFreq(data$Tag2),TAG3FREQ=tagFreq(data$Tag3),
              TAG4FREQ=tagFreq(data$Tag4),TAG5FREQ=tagFreq(data$Tag5),
              ISOPEN=(data$OpenStatus=="open"),stringsAsFactors=F)
names(dat)
dat$STATUS=rep(NA,140272)
dat$STATUS[dat$ISOPEN]="open"
dat$STATUS[!dat$ISOPEN]="closed"
dat=dat[,-12]
dat=dat[,-1]
dat[,1]=as.integer(dat[,1])
dat[is.na(dat)]=0
save(dat,file="dat.rda")
i=sample(1:140272)
dat=dat[i,]
train=dat[14028:140272,]
test=dat[1:14027,]
```

2 machine learning methods

I used single classification tree and logistic regression to predict the test data. However, the performance of both are poor. The misclassification rate of single tree is0.365, and logistic regression is 0.478. . Then I implement boosting method with a single tree to see whether there is any improvement. The result is with misclassification rate of 0.312. It's slightly improved.

3 Conclusion

In conclusion, the result of classification is not good. I think it's because I not use the information in Titles and Bodies. I only use the number of words of them, and this is not efficient or even somehow misleading(considering the small difference between single tree and boosting).

```
library(rpart)
classtree=rpart(STATUS~., data=train,method='class')
pred=predict(classtree,test[,-11], type = 'class')


table(truth = test$STATUS, predict= pred)


logreg=glm(I(STATUS=="open")~., data=train,family="binomial")
summary(logreg)
a=apply(test[,-11],1,function(x) sum(x*logreg$coefficients[-1])-0.43345)
pred1=ifelse(a>0.5,"open","closed")
table(truth = test$STATUS, predict= pred1)




boost<-function(dat,j){
```

```
  #initialization
  n=nrow(dat)
 wei=rep(1/n,n)
 a=rep(0,j)
 fitlist=list()
 #looping
 for(i in 1:j){
   fit=rpart(STATUS~., data=dat,weights = wei ,method = 'class')
   pred=predict(fit,type='class')
   mis=(pred!=dat$STATUS)
   err=sum(wei[mis])/sum(wei)
   alpha=log((1-err)/err)
   wei=wei*exp(alpha*mis)
   fitlist[[i]]=fit
 a[i]=alpha
 }
return(list(fit = fitlist, alpha = a))
}

train$STATUS=ifelse(train$STATUS=="open",1,-1)
test$STATUS=ifelse(test$STATUS=="open",1,-1)
l=boost(train,100)

fitlist=l$fit
alpha=l$alpha
predmatrix=matrix(rep(0,100*nrow(test)),ncol=100)
for(i in 1:100){pred=predict(fitlist[[i]],test[,-11],type='class')
  predmatrix[, i] = alpha[i] * ifelse(pred == 1, 1, -1)
}
pred1=apply(predmatrix,1,sum)
```

```
result=ifelse(pred1>0,1,-1)
table(result,test$STATUS)
```