# bag text

## bag text时，DocumentTermMatrix中，使用逆文本频率指数（Inverse document frequency 缩写为IDF）

我们很容易发现，如果一个关键词只在很少的网页中出现，我们通过它就容易锁定搜索目标，它的权重也就应该大。反之如果一个词在大量网页中出现，我们看到它 仍然不很清楚要找什么内容，因此它应该小。概括地讲，假定一个关键词 ｗ 在 Ｄｗ 个网页中出现过，那么 Ｄｗ 越大，ｗ 的权重越小，反之亦然。

## N-gram

N-gram是一种词汇预测算法，它使用可能性方法去预测n-1个单词之后的下一个单词，认为下一个单词的可能性与一系列单词的可能性密切相关。常用的是二元的Bi-Gram和三元的Tri-Gram。

部分模型中，用bigram\_proba [-1,1] 和 trigram\_proba [-1,1] 作为新属性：a numerical approach from bigrams and trigrams from the Headline Corpus to compute two variables bigram\_proba and trigram\_proba [-1,1]. Don't get fooled by the name, they are not probabilities at all but I was too lazy to change name. Those variables are built using the mean of popularity for every ngram weighted by their frequencies

用到Rweka包：

Weka是一个机器学习开源项目的简称（Waikato Environment for Knowledge Analysis，[http://www.cs.waikato.ac.nz/~ml/weka/](http://www.cs.waikato.ac.nz/%7Eml/weka/)）。 Weka项目从1992年开始，由新西兰政府支持，现在已在机器学习领域大名鼎鼎。Weka里有非常全面的机器学习算法，包括数据预处理、分类、回归、聚类、关联规则等。Weka的图形界面对不会写程序的人来说非常方便，而且提供“KnowledgeFlow” 功能，允许将多个步骤组成一个工作流。另外，Weka也允许在命令行执行命令

**RWeka** (<http://cran.r-project.org/web/packages/RWeka/index.html>)是R提供的和Weka的接口函数包RWeka

用法参见：<http://blog.csdn.net/lilanfeng1991/article/details/39957065>

require(RWeka)

library(rJava)

TrigramTokenizer <- function(x) RWeka::NGramTokenizer(x, RWeka::Weka\_control(min = 1, max = 3))

dtm <- DocumentTermMatrix(CorpusHeadline, control = list(tokenize = TrigramTokenizer))

# Feature enginerring

## 从已有的属性中，生成一些新的属性，突出与Popular的关系：

Also of note in the **Feature Engineering hint category**, think of considering exploring the Headlines for a few evocative "hot button" words that could act as 'click bait' and encourage people to read the article and then respond with their point of view on the matter. I did just that and found some significant variables that surprisingly did not emerge in the Corpus.

Another hint - think about people's raw needs and most heartfelt beliefs that have caused people to react over the test of time.

### 观察字段值和目标的关系：

table(NewsTrain$SectionName, NewsTrain$Popular) --- 绝对值

|  |  |  |
| --- | --- | --- |
|  | 0 | 1 |
|  | 2171 | 129 |
| Arts | 625 | 50 |
| Business Day | 999 | 93 |
| Crosswords/Games | 20 | 103 |
| Health | 74 | 120 |
| Magazine | 31 | 0 |
| Multimedia | 139 | 2 |
| N.Y. / Region | 181 | 17 |
| Open | 4 | 0 |
| Opinion | 182 | 425 |
| Sports | 1 | 0 |
| Style | 2 | 0 |
| Technology | 280 | 50 |
| Travel | 116 | 1 |
| U.S. | 405 | 100 |
| World | 209 | 3 |

tapply(newsTrain$Popular,newsTrain$SectionName,mean) --- 1所占百分比

|  |  |
| --- | --- |
|  | 0.056086957 |
| Arts | 0.074074074 |
| Business Day | 0.085164835 |
| Crosswords/Games | 0.837398374 |
| Health | 0.618556701 |
| Magazine | 0 |
| Multimedia | 0.014184397 |
| N.Y./Region | 0.085858586 |
| Open | 0 |
| Opinion | 0.700164745 |
| Sports | 0 |
| Style | 0 |
| Technology | 0.151515152 |
| Travel | 0.008547009 |
| U.S. | 0.198019802 |
| World | 0.014150943 |

The more variation there is, the more predictive power section name has. And ideally the most and least popular sections would have the most entries --- 可考虑新增对应的标志属性

对于长text字段中单词提炼出与pupular相关的方式，没有省事的方法，只能尝试观察文本和popular的关系：look through data and see what words are correlated to popularity. One example, it seemed to me that articles with Headlines containing Obama or President are more likely to be popular. So created a variable using Mystique's feedback. You can try several such combinations。

### 通过word cloud

made a subset of the popular posts and built a word cloud from headline & snippet (abstract is redundant to snippet), then explored and extracted some words with grepl. some of the words came from the word clouds, some from my own logical guesses (which i cross-checked with prop.table and table)

word cloud的使用：

library(tm)

library(SnowballC)

corpus = Corpus(VectorSource(tweets$Tweet ))

corpus = tm\_map(corpus, tolower)

corpus = tm\_map(corpus, PlainTextDocument)

corpus = tm\_map(corpus, removePunctuation)

removedCorpus = tm\_map(corpus, removeWords, stopwords("english"))

dtm = DocumentTermMatrix(removedCorpus)

allTweets = as.data.frame(as.matrix(dtm))

install.packages("wordcloud")

library(wordcloud)

wordcloud(colnames(allTweets), colSums(allTweets), scale=c(2,0.25))

i explored various words and groups of words in my popular subset and i looked at word clouds for the testing set of both headline and snippet. when looking at those, i explored frequent words with this basic line of code (but also look at table() without the prop.table function to see how many observations you are actually capturing, sometimes it could be just 1 or 2 and therefore not useful, want to catch things in the 100s at least):

一边看word cloud, 一边看单词出现的频度，如果只是出现了很少的次数，是没有用的。

prop.table(table(NewsTrain$Popular, grepl("word | otherword", NewsTrain$Headline, ignore.case=TRUE)),2)

### 文字中的符号如”?”和stopwords表中的一些单词

* made HeadlineIsQuestion and SnippetIsQuestion variables, and they both seemed to be significant

NewsTrain$HeadlineIsQuestion = as.factor(as.numeric(grepl("\\?", NewsTrain$Headline)))  
NewsTest$HeadlineIsQuestion = as.factor(as.numeric(grepl("\\?", NewsTest$Headline)))

* added variables for some typical stopwords extracted from the headline/snippet – "why" "where" and "how" seemed to be significant

# 其他

## [k-fold cross validation](http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29#k-fold_cross-validation)

<https://www.kaggle.com/c/15-071x-the-analytics-edge-competition-spring-2015/forums/t/13752/do-you-want-a-better-model>

Here is a function I wrote to make this task easier. Use it to test improvements in your model. Note that it uses the helper function conf.matrix which is included below. Also note that this function won't work with glm (sorry).

# Parameters:  
# data - data frame containing your training data  
# model - the function to call to make the model (e.g. rpart, gbm, knn3 etc.)  
# pargs - a list of arguments for the predict function  
# ... - parameters for the model  
# Examples:  
# kfold(TrainData,randomForest,pargs=list(type="prob"),nodesize=1,ntree=500)  
# kfold(TrainData,rpart,method="class",cp=0.01)  
kfold = function(data,model,pargs=NULL,...) {  
require(caret)  
# k is the number of folds, 5 or 10 are common choices  
k = 5  
metrics = list(k)  
means = numeric(5)  
folds = createFolds(data$Popular,k=k,list=TRUE,returnTrain=TRUE)  
for (i in 1:length(folds)) {  
dfT = data[folds[[i]],]  
dfP = data[folds[[i]]\*-1,]  
# change the formula (Popular~.) in the line below if you need to  
m = model(Popular~.,data=dfT,...)  
p = do.call(predict,c(list(object=m,newdata=dfP),pargs))  
if (!is.vector(p))  
if (ncol(p)>1)  
p = p[,2]  
mk = conf.matrix(dfP$Popular,p,0.5)$metrics  
# remove the # in front of the print statement if you want to see   
# data for each iteration  
#print(mk)  
metrics[[i]] = mk  
}  
for (i in 1:length(metrics)) {  
means = means + metrics[[i]]  
}  
means = means / k  
names(means) = names(metrics[[1]])  
list(means=means)  
}

conf.matrix = function (outcomes,predictions,cutoff) {  
require(ROCR)  
if (class(predictions) == "factor") {  
auc=NA  
t = table(outcomes,predictions)  
}  
else {  
auc = performance(prediction(predictions,outcomes),"auc")@y.values[[1]]  
t = table(outcomes,predictions=predictions >= cutoff)  
if (cutoff > max(predictions)) {  
t = cbind(t,"TRUE"=c(0,0))  
}  
else if (min(predictions) > cutoff)  {  
t = cbind("FALSE"=c(0,0),t)  
}  
}  
acc = (t[1,1] + t[2,2]) / length(predictions)  
sen = t[2,2] / (t[2,1] + t[2,2])  
spec = t[1,1] / (t[1,1] + t[1,2])  
baseline.accuracy = max(t[1,1]+t[1,2],t[2,1]+t[2,2])/length(predictions)  
list(confusion.matrix=t,  
metrics=c(sensitivity=sen,specificity=spec,accuracy=acc,baseline.acc=baseline.accuracy,auc=auc))  
}

## 多个模型结合常常让性能更好

一次多个模型：

fiveStats <- function(...) c(twoClassSummary(...), defaultSummary(...))

## ENSEMBLE

ensCtrl <- trainControl(method="cv",

number=5,

savePredictions=TRUE,

allowParallel=TRUE,

classProbs=TRUE,

index=createResample(train$Popular, 25),

selectionFunction="best",

summaryFunction=fiveStats)

glmGrid <- expand.grid(alpha=c(.4), lambda=2^-8)

rfGrid <- expand.grid(mtry=c(17))

gbmGrid <- expand.grid(n.trees=c(3500), interaction.depth=c(27), shrinkage=c(.001))

svmGrid <- expand.grid(.sigma=c(.0007),.C=c(16,32))

model\_list <- caretList(

Popular ~ .,

data=train,

trControl=ensCtrl,

metric="ROC",

tuneList=list(

rf=caretModelSpec(method="rf", tuneGrid=rfGrid, nodesize=1, ntree=3000),

glmnet=caretModelSpec(method="glmnet", tuneGrid=glmGrid, preProcess=c("center","scale")),

gbm=caretModelSpec(method="gbm", tuneGrid=gbmGrid),

svm=caretModelSpec(method="svmRadial", tuneGrid=svmGrid, preProcess=c("center","scale"))

)

)

xyplot(resamples(model\_list))

modelCor(resamples(model\_list))

greedy\_ensemble <- caretEnsemble(model\_list)

library('caTools')

model\_preds <- lapply(model\_list, predict, newdata=test, type='prob')

model\_preds <- lapply(model\_preds, function(x) x[,'Yes'])

model\_preds <- data.frame(model\_preds)

ens\_preds <- predict(greedy\_ensemble, newdata=test)

model\_preds$ensemble <- ens\_preds

# 在多处理器上并行计算

# Creates a set of copies of R running in parallel and communicating over sockets.

cl <- makeCluster(7)

registerDoParallel(cl)

。。。

stopCluster(cl)