Machine Learning Homework 2

Rob McCulloch

1/22/2018

Contents

1. Tuning the Laplace Parameter in Text Classification

In the end, the way we did the ham/spam text classification boiled down to computing simple 2x2 tables of conditional distributions for the presence of a term give the document is ham or spam.

We saw that some of the estimated tables assigned probability zero to the term being in the document given it is spam. This would mean if the term is in the document you know "for sure" that it is ham. This seems extreme.

The laplace parameter "shrinks" the estimated probabilities away from 0 (or 1).
In class we tried setting the laplace parameter to 1. This did change the estimated probabilites (in particular no more zeros) but the out of sample performance of the classifier was about the same.

Let's see if we can find the optimal (well, a better) choice of the laplace parameter.

We want to:

- repeatly draw train and test subsets of our data
- for a chosen grid (set) of laplace parameters, train using the training data and the laplace parameter, predict on test
- for each train/test laplace parameter combo, measure the performance of the classifier
- report the results with tabls/plots

Here is a simple example of how you might draw a train/test split in R.

Sample Train and Test

```
n=10;p=2
set.seed(14)
x = matrix(rnorm(n*p),ncol=p)
x
```

```
##              [,1]       [,2]
##  [1,] -0.66184983 -0.3828219
##  [2,]  1.71895416  0.2994216
##  [3,]  2.12166699  0.6742398
##  [4,]  1.49715368 -0.2928163
##  [5,] -0.03614058  0.4880534
##  [6,]  1.23194518  0.8828018
##  [7,] -0.06488077  1.8627490
##  [8,]  1.06899373  1.6117253
##  [9,] -0.37696531  0.1354795
## [10,]  1.04318309  1.0880860
```

```
trainfrac=.75; nTrain = floor(n*trainfrac)
set.seed(99)
ii = sample(1:n,nTrain)
print(ii)
```

```
## [1]  6  2 10  7  4  5  3
```

```
xtrain = x[ii,]
xtest = x[-ii,]
print(xtrain)
```

```
##             [,1]       [,2]
## [1,]  1.23194518  0.8828018
## [2,]  1.71895416  0.2994216
## [3,]  1.04318309  1.0880860
## [4,] -0.06488077  1.8627490
## [5,]  1.49715368 -0.2928163
## [6,] -0.03614058  0.4880534
## [7,]  2.12166699  0.6742398
```

```
print(xtest)
```

```
##            [,1]       [,2]
## [1,] -0.6618498 -0.3828219
## [2,]  1.0689937  1.6117253
## [3,] -0.3769653  0.1354795
```

Read in sms Data and Wrangle

Let's review the Naive Bayes analysis of the sms data.

Read in Data

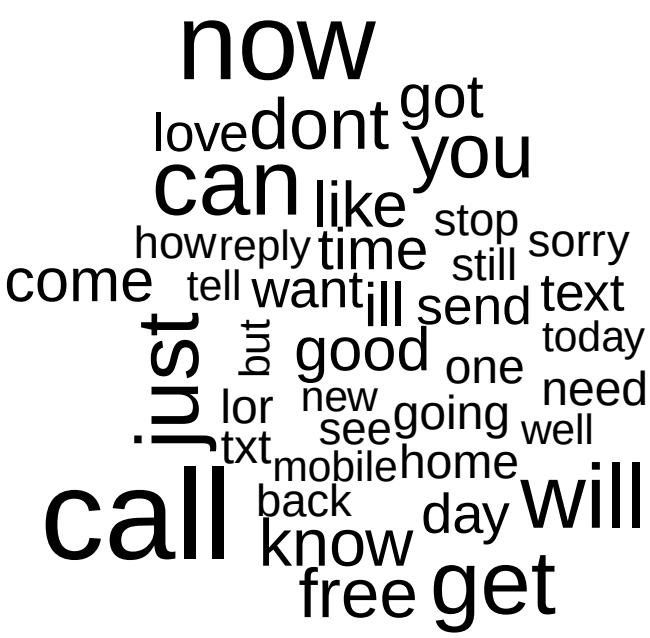
```
# read in data
smsRaw = read.csv("sms_spam.csv", stringsAsFactors = FALSE)
# convert spam/ham to factor.
smsRaw$type = factor(smsRaw$type)

#look at y=type
print(table(smsRaw$type))
```

```
##
##  ham spam
## 4812  747
```

```
#look at x=words
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
wordcloud(smsRaw$text, max.words = 40)
```



Make and Clean Corupus

```
# build a corpus using the text mining (tm) package
library(tm)
library(SnowballC)
#volatile (in memory corpus from vector of text in R
smsC = VCorpus(VectorSource(smsRaw$text))
# clean up the corpus using tm_map()
smsCC = tm_map(smsC, content_transformer(tolower)) #upper -> lower
smsCC = tm_map(smsCC, removeNumbers) # remove numbers
smsCC = tm_map(smsCC, removeWords, stopwords()) # remove stop words
smsCC = tm_map(smsCC, removePunctuation) # remove punctuation
smsCC = tm_map(smsCC, stemDocument) #stemming
smsCC = tm_map(smsCC, stripWhitespace) # eliminate unneeded whitespace
```

Get Document Term Matrix

```
# create Document Term Matrix
smsDtm = DocumentTermMatrix(smsCC)
dim(smsDtm)
```

```
## [1] 5559 6559
```

Out of Sample Missclassification

Train and Test

```
# creating training and test datasets
smsTrain = smsDtm[1:4169, ]
smsTest  = smsDtm[4170:5559, ]
smsTrainy = smsRaw[1:4169, ]$type
smsTesty  = smsRaw[4170:5559, ]$type
cat("training fraction is: ",4169/5559,"\n")
```

```
## training fraction is:  0.749955
```

Freq Words and Convert Counts to Binary

```
smsFreqWords = findFreqTerms(smsTrain, 5) #words that appear at leat 5 times
smsFreqTrain = smsTrain[ , smsFreqWords]
```

```
smsFreqTest = smsTest[ , smsFreqWords]

convertCounts <- function(x) {
  x <- ifelse(x > 0, "Yes", "No")
}
# apply() convert_counts() to columns of train/test data
smsTrain = apply(smsFreqTrain, MARGIN = 2, convertCounts)
smsTest  = apply(smsFreqTest, MARGIN = 2, convertCounts)
```

Naive Bayes and Missclassification

```
library(e1071)
smsNB = naiveBayes(smsTrain, smsTrainy, laplace=1)
yhat = predict(smsNB,smsTest)
ctab = table(yhat,smsTesty)
ctab
```

```
##       smsTesty
## yhat    ham spam
##   ham  1202   28
##   spam    5  155
```

```
misclass = (sum(ctab)-sum(diag(ctab)))/sum(ctab)
perspam = ctab[2,2]/sum(ctab[,2])
cat("misclass,perspam: ", misclass,perspam,"\n")
```

```
## misclass,perspam:  0.02374101 0.8469945
```

Now let's redo if for a random train/test split.

```
# sample train/test
trainfrac=.75
n= length(smsRaw$type)
nTrain = floor(trainfrac*n)
set.seed(99)
ii = sample(1:n,nTrain)
smsTrain = smsDtm[ii, ]
smsTest  = smsDtm[-ii, ]
smsTrainy = smsRaw[ii, ]$type
smsTesty  = smsRaw[-ii, ]$type
# freq words
smsFreqWords = findFreqTerms(smsTrain, 5) #words that appear at leat 5 times
smsFreqTrain = smsTrain[ , smsFreqWords]
smsFreqTest = smsTest[ , smsFreqWords]
# counts -> binary
smsTrain = apply(smsFreqTrain, MARGIN = 2, convertCounts)
smsTest  = apply(smsFreqTest, MARGIN = 2, convertCounts)
smsNB = naiveBayes(smsTrain, smsTrainy, laplace=1)
#pred and misclass
yhat = predict(smsNB,smsTest)
ctab = table(yhat,smsTesty)
ctab
```

```
##       smsTesty
## yhat    ham spam
##   ham  1207   25
##   spam    6  152
```

```
misclass = (sum(ctab)-sum(diag(ctab)))/sum(ctab)
perspam = ctab[2,2]/sum(ctab[,2])
cat("misclass,perspam: ", misclass,perspam,"\n")
```

```
## misclass,perspam:  0.02230216 0.8587571
```

Not too different, *good* !!

Now the idea is to put the above in a loop. We want to loop over train/test splits *and* possible values for the laplace parameter.

2. Classification with Numeric x

In our text classification we ended up have each x_i binary and a y binary.

What would we do if x is numeric?

A common approach is to assume each x_i is conditionally normal.

$$X_i|Y = y \sim N(\mu_{iy}, \sigma_{iy}^2)$$

Recall:

$$X \sim N(\mu, \sigma^2) \Rightarrow f(x) = \frac{1}{\sqrt{2\pi}} \frac{1}{\sigma} \exp(-\frac{1}{2\sigma^2}(x-\mu)^2)$$

Note the the *non naive Bayes* approach to would be to let

$$x|Y = y \sim N(\mu_y, \Sigma_y)$$

where now x is a vector and $N(\mu, \Sigma)$ means the multivariate normal distribution.

A good problem would be to code up these approaches in R and see how they work.

But lets just do a little simple algebra and get expressions for the log odds

$$\log(\frac{p(y=1|x)}{p(y=0|x)}$$

given $X|y \sim N(\mu_y, \sigma_y^2)$.

Note that since

$$p(y|x) \propto p(y) f(x|y)$$

the odds ratio is

$$odds = \frac{p(Y=1)}{p(Y=0)} \frac{f(x|Y=1)}{f(x|Y=0)}$$

the prior odds times the likelihood ratio.

Let's assume $X|Y = 1 \sim N(\mu_1, \sigma^2)$ and $X|Y = 0 \sim N(\mu_0, \sigma^2)$.

How simple can you get the rule for deciding whether or not Y is 0 or 1 given x?