Naïve Bayes Spam Filter

Jenian Tai (kt2694)

10/7/2018

```
readDirectory <- function(dirname) {
	# Store the emails in a list
	emails = list();
	# Get a list of filenames in the directory
	filenames = dir(dirname, full.names=TRUE);

	for (i in 1:length(filenames)){
		emails[[i]] = scan(filenames[i], what="", quiet=TRUE);
	}
	return(emails)
}
```

Loaded the data.

```
ham_test <- readDirectory("/Users/jenian/Documents/APANPS5335/hw4/ham-v-spam/ham-test/")
ham_train <- readDirectory("/Users/jenian/Documents/APANPS5335/hw4/ham-v-spam/ham-train/")
spam_test <-readDirectory("/Users/jenian/Documents/APANPS5335/hw4/ham-v-spam/spam-test/")
spam_train <- readDirectory("/Users/jenian/Documents/APANPS5335/hw4/ham-v-spam/spam-train/")
```

```
print(ham_train[1])
```

```
## [[1]]
##   [1] "gari"      "product"   "high"      "island"    "larger"
##   [6] "block"     "commenc"   "saturday"  "gross"     "carlo"
##  [11] "expect"    "gross"     "tomorrow"  "vastar"    "own"
##  [16] "gross"     "product"   "georg"     "daren"     "farmer"
##  [21] "carlo"     "rodriguez" "hou"       "ect"       "ect"
##  [26] "georg"     "weissman"  "hou"       "ect"       "ect"
##  [31] "melissa"   "grave"     "hou"       "ect"       "ect"
##  [36] "carlo"     "pleas"     "call"      "linda"     "get"
##  [41] "everyth"   "set"       "estim"     "come"      "tomorrow"
##  [46] "increas"   "follow"    "day"       "base"      "convers"
##  [51] "bill"      "fischer"   "bmar"      "enron"     "north"
##  [56] "america"   "corp"      "georg"     "weissman"  "daren"
##  [61] "farmer"    "hou"       "ect"       "ect"       "gari"
##  [66] "bryan"     "hou"       "ect"       "ect"       "melissa"
##  [71] "grave"     "hou"       "ect"       "ect"       "darren"
##  [76] "attach"    "appear"    "nomin"     "vastar"    "resourc"
##  [81] "inc"       "high"      "island"    "larger"    "block"
##  [86] "previous"  "erron"     "refer"     "well"      "vastar"
##  [91] "expect"    "well"      "commenc"   "product"   "sometim"
##  [96] "tomorrow"  "told"      "linda"     "harri"     "get"
## [101] "telephon"  "number"    "gas"       "control"   "provid"
## [106] "notif"     "turn"      "tomorrow"  "linda"     "number"
## [111] "record"    "voic"      "fax"       "would"     "pleas"
```

```
## [116] "see"       "someon"   "contact"  "linda"    "advis"
## [121] "submit"    "futur"    "nomin"    "via"      "mail"
## [126] "fax"       "voic"     "thank"    "georg"    "linda"
## [131] "harri"     "georg"    "weissman" "hou"      "ect"
## [136] "ect"       "effect"   "mscf"     "min"      "ftp"
## [141] "time"      "hour"     "hour"     "hour"     "hour"
## [146] "hour"      "hour"     "hour"     "hour"     "hour"
## [151] "hour"      "hour"     "hour"     "hour"     "hour"
## [156] "hour"      "hour"     "hour"
```

```r
print(spam_train[1])
```

```
## [[1]]
##   [1]"introduc"   "doctor"    "formul"    "hgh"       "human"
##   [6]"growth"     "hormon"    "also"      "call"      "hgh"
## [11]  "refer"      "medic"     "scienc"    "master"    "hormon"
## [16]  "plenti"     "young"     "near"      "age"       "twenti"
## [21]  "one"        "bodi"      "begin"     "produc"    "less"
## [26]  "time"       "forti"     "near"      "everyon"   "defici"
## [31]  "hgh"        "eighti"    "product"   "normal"    "diminish"
## [36]  "least"      "advantag"  "hgh"       "increas"   "muscl"
## [41]  "strength"   "loss"      "bodi"      "fat"       "increas"
## [46]  "bone"       "densiti"   "lower"     "blood"     "pressur"
## [51]  "quicken"    "wound"     "heal"      "reduc"     "cellulit"
## [56]  "improv"     "vision"    "wrinkl"    "disappear" "increas"
## [61]  "skin"       "thick"     "textur"    "increas"   "energi"
## [66]  "level"      "improv"    "sleep"     "emot"      "stabil"
## [71]  "improv"     "memori"    "mental"    "alert"     "increas"
## [76]  "sexual"     "potenc"    "resist"    "common"    "ill"
## [81]  "strengthen" "heart"     "muscl"     "control"   "cholesterol"
## [86]  "control"    "mood"      "swing"     "new"       "hair"
## [91]  "growth"     "color"     "restor"    "read"      "websit"
## [96]  "unsubscrib"
```

```r
makeSortedDictionaryDf <- function(emails){
    # This returns a dataframe that is sorted by the number of times
    # a word appears
    # List of vectors to one big vetor
    dictionaryFull <- unlist(emails)
    # Tabulates the full dictionary
    tabulateDic <- tabulate(factor(dictionaryFull))
    # Find unique values
    dictionary <- unique(dictionaryFull)
    # Sort them alphabetically
    dictionary <- sort(dictionary)
    dictionaryDf <- data.frame(word = dictionary, count = tabulateDic) sortDictionaryDf <-
    dictionaryDf[order(dictionaryDf$count,decreasing=TRUE),]; return(sortDictionaryDf)

}
```

```r
all_emails <- c(ham_test, ham_train, spam_test, spam_train)
dictionary <- makeSortedDictionaryDf(all_emails)
```

```r
makeDocumentTermMatrix <- function(emails, dictionary){
	# This takes the email and dictionary objects from above and outputs a
	# document term matrix
	num_emails <- length(emails);
	num_words <- length(dictionary$word);
	# Instantiate a matrix where rows are documents and columns are words dtm <-
	mat.or.vec(num_emails, num_words); # A matrix filled with zeros
	for (i in 1:num_emails){
		num_words_email <- length(emails[[i]]);
		email_temp <- emails[[i]];
		for (j in 1:num_words_email){
			ind <- which(dictionary$word == email_temp[j]);
			dtm[i, ind] <- dtm[i, ind] + 1;
		}
	}
return(dtm);
}
```

```r
dtm_ham_train <- makeDocumentTermMatrix(ham_train, dictionary)
dtm_spam_train <- makeDocumentTermMatrix(spam_train, dictionary)
dtm_ham_test <- makeDocumentTermMatrix(ham_test, dictionary)
dtm_spam_test <- makeDocumentTermMatrix(spam_test, dictionary)
```

```r
makeLogPvec <- function(dtm, mu){
	# Sum up the number of instances per word
	pvecNoMu <- colSums(dtm)
	# Sum up number of words
	nWords <- sum(pvecNoMu)
	# Get dictionary size
	dicLen <- length(pvecNoMu)
	# Incorporate mu and normalize
	logPvec <- log(pvecNoMu + mu) - log(mu*dicLen + nWords)
	return(logPvec)
}
```

```r
mu <- 1 / length(dictionary$word)
log_pvec_ham <- makeLogPvec(dtm_ham_train, mu)
log_pvec_spam <- makeLogPvec(dtm_spam_train, mu)
```

```
predictNaiveBayes <- function(log_pvec_ham, log_pvec_spam, log_ham_prior, log_spam_prior, dtm_test) {
	n <- nrow(dtm_test)
	yh <- numeric(n)
	for (i in 1:n) {
	   if (sum(dtm_test[i,] * log_pvec_spam) + log_spam_prior > sum(dtm_test[i,] * log_pvec_ham) + log_h yh[i] <- 1
	   }
	   else {
	      yh[i] <- 0
	   }
	}
	return(yh)
}
```

```
ham_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_ham_test)
spam_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_spam_test)

hr <- mean(spam_hat)
fa <- mean(ham_hat)
acc <- (sum(spam_hat) + length(ham_hat) - sum(ham_hat)) / (length(ham_hat) + length(spam_hat))
print(hr)
```

```
## [1] 0.9466667
```

```
print(fa)
```

```
## [1] 0.04
```

```
print(acc)
```

```
## [1] 0.9533333
```

```
fiveFoldCV <- function(dtm_ham_train, dtm_spam_train, log_ham_prior, log_spam_prior, mu){
	errors <- numeric(5)
	#  split up your data into 5 sets n <-
	nrow(dtm_ham_train) fold_size <-n/5
	for (i in 1:5) { full_range
		<- 1:n
		validation_range <- ((i-1) * fold_size + 1):(i * fold_size) train_range <-
		full_range[! full_range %in% validation_range]
		#  train on the train_range using makeLogPvec()
		log_pvec_ham <- makeLogPvec(dtm_ham_train[train_range,], mu) # 1 point
		log_pvec_spam <- makeLogPvec(dtm_spam_train[train_range,], mu) # 1 point

		# validate on the validation_range using predictNaiveBayes()
		ham_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_ham_train[valid
		spam_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_spam_train[val
		# calculate the error rate and store in vector (did you initialize it?)
		errors[i] <- (mean(ham_hat) + mean(1 - spam_hat))/2 # 1 point
	}
	#  return the average error over all folds
	return(mean(errors))
}
```

```r
bm <- 1 / length(dictionary$word)
mus <- c(1/100, 1/10, 1, 10, 100) * bm
errs <- numeric(length(mus))
for (i in 1:length(mus)) {
  errs[i] <- fiveFoldCV(dtm_ham_train, dtm_spam_train, log(0.5), log(0.5), mus[i])
best_mu <- mus[which.min(errs)]
print(best_mu)
```

```
## [1] 0.0004389623
```

```r
log_pvec_ham <- makeLogPvec(dtm_ham_train, best_mu)
log_pvec_spam <- makeLogPvec(dtm_spam_train, best_mu)

ham_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_ham_test)
spam_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_spam_test)

hr <- mean(spam_hat)
fa <- mean(ham_hat)
acc <- (sum(spam_hat) + length(ham_hat) - sum(ham_hat)) / (length(ham_hat) + length(spam_hat)) print(hr)
```

```
## [1] 0.94
```

```r
print(fa)
```

```
## [1] 0.04
```

```r
print(acc)
```

```
## [1] 0.95
```

The accuracy, specificity and sensitivity turn out lower using the new mu, but we shall stay with new mu = 1/|D| to avoid overfit.

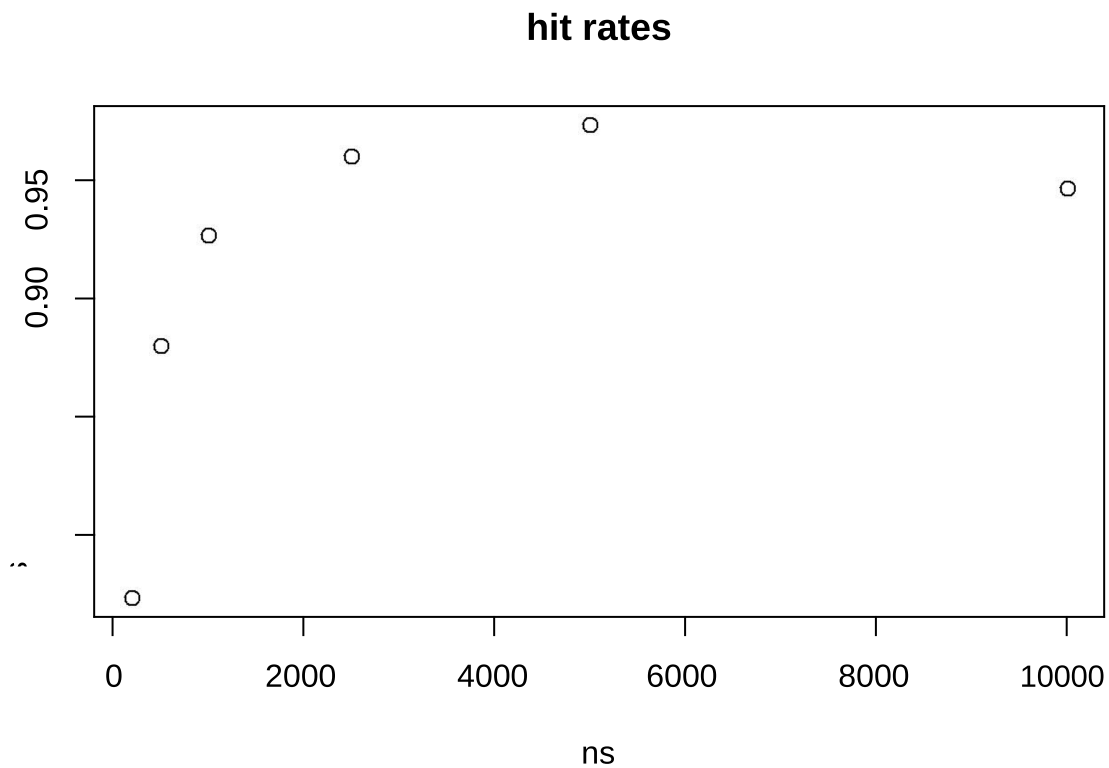
```r
calculateMI <- function(dtm_ham_train, dtm_spam_train) {
  # calculates vector of mutual information for each word.
  ham_sums <- colSums(dtm_ham_train)
  ham_probs <- ham_sums / sum(ham_sums) # vector of probabilities for each word in ham
  spam_sums <- colSums(dtm_spam_train)
  spam_probs <- spam_sums / sum(spam_sums) # vector of probabilities for each word in spam
  all_sums <- ham_sums + spam_sums
  all_probs <- all_sums / sum(all_sums) # vector of probabilites for word in entire set mi <-
  c(length(all_probs))
  for (i in 1:length(all_probs)) {
    if (all_probs[i] == 0 || ham_probs[i] == 0 || spam_probs[i] == 0) { mi[i] <- 0
      # mutual information -> 0 when p(X=x) = 0
    }
    else {
      mi[i] <- .5 * ham_probs[i] * log(ham_probs[i] / all_probs[i]) +
                 .5 * (1 - ham_probs[i]) * log((1 - ham_probs[i])/(1 - all_probs[i])) +
                 .5 * spam_probs[i] * log(spam_probs[i] / all_probs[i]) +
                 .5 * (1 - spam_probs[i]) * log((1 - spam_probs[i])/(1 - all_probs[i]))
    }
  }
  return(mi)
}
```

```r
mivec <- calculateMI(dtm_ham_train, dtm_spam_train)
```

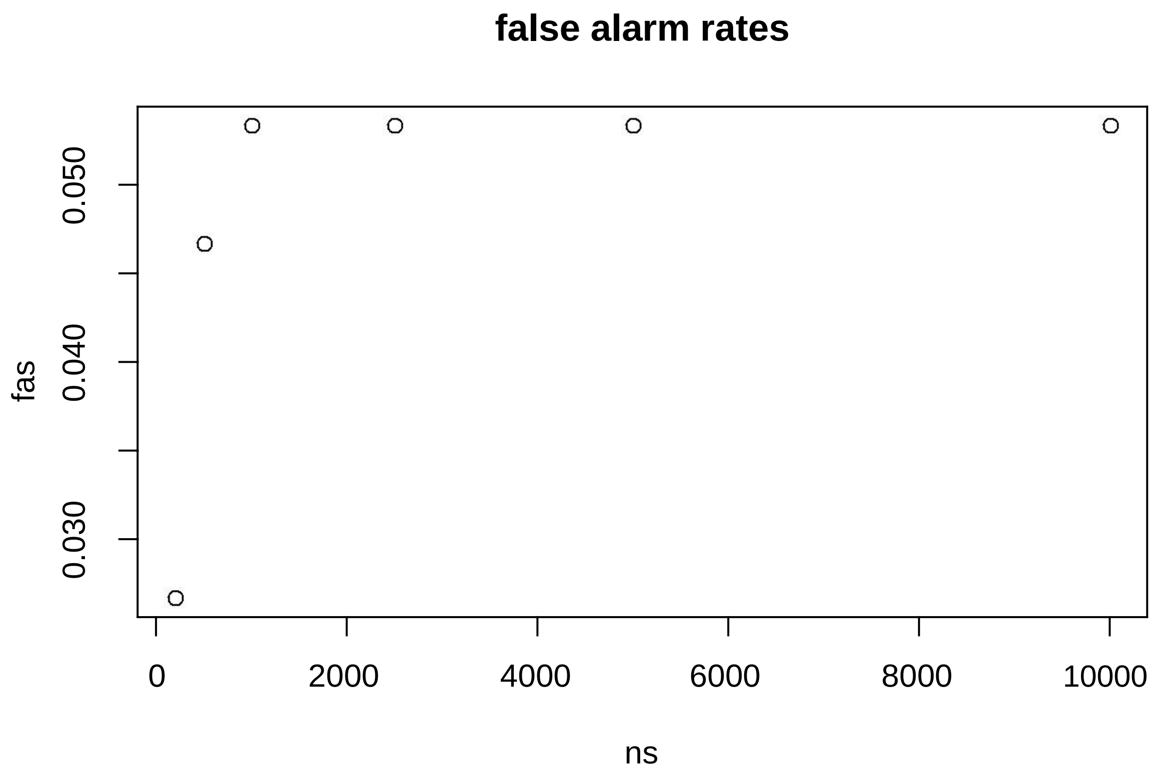
```r
fitAndPredict <- function(dtm_ham_train, dtm_spam_train, dtm_ham_test, dtm_spam_test, mu)
  { log_pvec_ham <- makeLogPvec(dtm_ham_train, mu)
  log_pvec_spam <- makeLogPvec(dtm_spam_train, mu)
  ham_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_ham_test)
  spam_hat <- predictNaiveBayes(log_pvec_ham, log_pvec_spam, log(.5), log(.5), dtm_spam_test)
  hr <- mean(spam_hat)
  fa <- mean(ham_hat)
  acc <- (sum(spam_hat) + length(ham_hat) - sum(ham_hat)) / (length(ham_hat) + length(spam_hat))
  return(c(hr, fa, acc))
}

ns <- c(200,500,1000,2500,5000,10000)
hrs <- c(length(ns))
fas <- c(length(ns))
accs <- c(length(ns))
for(i in 1:length(ns)) {
  n <- ns[i]
```

```
    sub_dtm_ham_train <- dtm_ham_train[,order(mivec)[1:n]]
    sub_dtm_spam_train <- dtm_spam_train[,order(mivec)[1:n]]
    sub_dtm_ham_test <- dtm_ham_test[,order(mivec)[1:n]]
    sub_dtm_spam_test <- dtm_spam_test[,order(mivec)[1:n]]
    o <- fitAndPredict(sub_dtm_ham_train, sub_dtm_spam_train,
                       sub_dtm_ham_test, sub_dtm_spam_test, 1/n)
    hrs[i] <- o[1]
    fas[i] <- o[2]
    accs[i] <- o[3]
}
plot(ns, hrs, main="hit rates")
```



```r
plot(ns, fas, main="false alarm rates")
```




```r
plot(ns, accs, main="accuracy")
```