'JNaiveBayes' package

Tyler Olson Alex Zajichek

May 2, 2017

Abstract

The `JNaiveBayes` package implements a Naive Bayes text classifier with a very direct approach of allowing the user to supply directories of documents containing words, and receive predicted classifications of unclassified test documents. The user has the option to use a parametric approach, which uses a Poisson distribution on word counts, or a nonparametric approach, which calcaluates probabilities based on observed proportions. An additional tuning parameter is also available for each formulation, giving the user the option to weight unobserved words as seen fit. The heavy computation is done in a collection of Java classes by using the `rJava` package as an interface within the `JNaiveBayes` function. The goal of this project was to explore R's capabilities' to connect with Java, and to successfully build an R package using the connection.

1 Introduction

Classification in the field of machine learning is used to predict some type of categorial response for individual observations. The supervised classification of text, where labeled data is used to train the model, has a wide range of applications. Spam detection, word comprehension, and genre identification are just a few examples of the utility of text classification. This paper will develop a Naive Bayes text classifier in two model formulations-parametric, and nonparametric. The models will then be implemented in Java, interfaced by R with the `rJava` package, and combined into a single R package for user accessibility. The user will be able to supply documents with the purpose of categorizing new ones.

2 Model Formulation

Let

- C_i be the i^{th} document category where $i = 1, 2, ..., K$
- D_j be the j^{th} document where $j = 1, 2, ..., M_i$
- W_{jk} be the k^{th} *unique* word in D_j where $k = 1, 2, ..., N_j$
- O_{jk} be the *occurrence* of W_{jk} in D_j

The Naïve Bayes' model can then be defined in two formulations:

Parametric

Let

$$O_{jk}|C_i \sim Poisson(\lambda_{ik} = \frac{\sum_{j=1}^{M_i} O_{jk}}{M_i})$$

where λ_{ik} is the average occurrence of word W_{jk} in M_i documents of a given category, and

$$C_i \sim Bernoulli(p_i = \frac{M_i}{\sum_{i'=1}^{K} M_{i'}})$$

where p_i is the proportion of total documents belonging to category i. Then,

$$\begin{aligned} P(C_i|D_j) &\propto P(D_j|C_i)P(C_i) \\ &= P(O^*_{j1} \cap O^*_{j2} \cap ... \cap O^*_{jN_j}|C_i)P(C_i) \\ &= P(O^*_{j1}|C_i)P(O^*_{j2}|C_i) \times ... \times P(O^*_{jN_j}|C_i)P(C_i) \\ &= \prod_{k=1}^{N_j} P(O^*_{jk}|C_i)P(C_i) \end{aligned}$$

where

$$P(O_{jk}^*|C_i) = (1-\delta)\frac{P(O_{jk}|C_i)}{P(O_{jk}>0|C_i)} + \frac{\delta}{\sum_{j=1}^{M_i}\sum_{k=1}^{N_j} O_{jk}}$$
$$= (1-\delta)\frac{e^{-\lambda_{ik}}\lambda_{ik}^{O_{jk}}}{O_{jk}!(1-e^{-\lambda_{ik}})} + \frac{\delta}{\sum_{j=1}^{M_i}\sum_{k=1}^{N_j} O_{jk}}$$

and δ is a smoothing parameter that allows a small-weighted positive probability to be assigned to unobserved words in a category.

Nonparametric

The nonparametric approach calculates probabilities based on proportions of word occurences. Let

$$P(C_i) = \frac{M_i}{\sum_{i'=1}^{K} M_{i'}}$$

and

$$P(W_{jk}|C_i) = (1-\delta)\frac{\sum_{j=1}^{M_i} O_{jk}}{\sum_{j=1}^{M_i}\sum_{k=1}^{N_j} O_{jk}} + \frac{\delta}{\sum_{j=1}^{M_i}\sum_{k=1}^{N_j} O_{jk}}$$

where M_i is the number of documents in category i. Then,

$$\begin{aligned} P(C_i|D_j) &\propto P(D_j|C_i)P(C_i) \\ &= P(W_{j1}|C_i)^{O_{j1}} P(W_{j2}|C_i)^{O_{j2}} \times ... \times P(W_{iN_j}|C_i)^{O_{jN_j}} P(C_i) \\ &= \prod_{k=1}^{N_j} P(W_{jk}|C_i)^{O_{jk}} P(C_i) \end{aligned}$$

Therefore, given a new document,

$$P(C_i|D_j) = \frac{P(D_j|C_i)P(C_i)}{\sum_{i=1}^{K} P(D_j|C_i)P(C_i)}$$

can be computed in both model specifications.

2.1 More on δ

The difference in probability calculation between the parametric and nonparametric approaches is only present for words in a document that have been previously observed by a category. If the word has not been observed, a probability of one over the total number of words in a category is given. Further, the δ parameter allows the user the freedom to weight these unobserved words as seen fit. Typically it should be a small value. By default, $\delta = 0.01$.

3 Requirements

3.1 Data

3.2 Directory

4 Example

```
> library(JNaiveBayes)
> library(xtable)
> train <- system.file("Data", "Training", package = "JNaiveBayes")
> test <- system.file("Data", "Testing", package = "JNaiveBayes")
> model1 <- JNaiveBayes(trainDir = train, testDir = test, parametric = FALSE, delta = 0.01)
> model2 <- JNaiveBayes(trainDir = train, testDir = test, parametric = TRUE, delta = 0.01)
> model3 <- JNaiveBayes(trainDir = train, testDir = test, parametric = FALSE, delta = 0.15)
> xtable(model1$Probabilities);xtable(model2$Probabilities);
```

	Personal	Public
unknown1.txt	0.10	0.90
unknown2.txt	0.97	0.03
unknown3.txt	1.00	0.00

	Personal	Public
unknown1.txt	0.75	0.25
unknown2.txt	1.00	0.00
unknown3.txt	1.00	0.00

5 Application

This section gives a demonstration of the `JNaiveBayes` package for a publicly available dataset found at the following website:

`https://www.kaggle.com/crowdflower/twitter-airline-sentiment`

The goal of this application will be to test the predictive ability of this methodology in regards to *sentiment analysis*. Given a number of sentiments (categories) and their corresponding documents, a large test set will be classified. The test set will be held out of the original data so the number of correct classifications can obtained. The R code used will be included.

5.1 Data description

The archive from the webpage contains Twitter information of many users' "tweets" towards a number of US airlines from 02/16/2015-02/24/2015. There are a total of 14640 recorded tweets in the dataset, each with one of the corresponding sentiments: *negative, neutral, positive*. Below shows the first 3 rows of the dataset, after removing everything but the tweets and labels, including punctuation:

```
> tweets <- read.csv("Tweets.csv", header = T, stringsAsFactors = FALSE)
> tweets <- data.frame("Sentiment" = as.character(tweets$airline_sentiment),
+ "Tweet" = gsub("[[:punct:]]",'',as.character(tweets$text)))
> head(tweets)[1:3,]
  Sentiment                                                       Tweet
1   neutral                                VirginAmerica What dhepburn said
2  positive VirginAmerica plus youve added commercials to the experience tacky
3   neutral  VirginAmerica I didnt today Must mean I need to take another trip
```

We will randomly select 1000 tweets to be held out for classification, leaving 13640 to build the model.

```
> set.seed(10) #For reproducibility
> test_inds <- sample(1:nrow(tweets),1000, replace = F)
> train <- tweets[-test_inds,]
> test <- tweets[test_inds,]
```

5.2 Data preprocessing

Since all of this data is currently in a data frame, there needs to be some manipulation done to get it into the correct form for `JNaiveBayes`, as described in the previous section. Each category must have its own directory, containing text files that have a single word on each line. Instead of creating a document for each tweet in the training data, all words from a category can be put into a single document because our model uses class-conditional independence. The following code was used to do this:

```
> #separating out the three sentiments
> neut <- subset(train, Sentiment == 'neutral')
> pos <- subset(train, Sentiment == 'positive')
> neg <- subset(train, Sentiment == 'negative')
>
> #creating vectors where each element is a single word
> negs <- unlist(strsplit(neg$Tweet, ' ')); poses <- unlist(strsplit(pos$Tweet, ' '))
> neuts <- unlist(strsplit(neut$Tweet, ' '))
> #writing each vector out to file where words are placed line by line
> write.table(tolower(negs[negs != ""]), file = "Negatives.txt", row.names = F, quote = F)
> write.table(tolower(poses[poses != ""]), file = "Positives.txt", row.names = F, quote = F)
> write.table(tolower(neuts[neuts != ""]), file = "Neutrals.txt", row.names = F, quote = F)
```

There are three training documents containing all of the words, which can each be placed in their own subdirectories. Note that each tweet could have been given its own file and would achieve the same result when running the model.

The testing tweets need a little more care. Since the goal is to classify each tweet into one of the three sentiments, there needs to be a separate text file written out for all 1000 of them.

```
> #storing the true labels
> test_labs <- test$Sentiment
>
> #obtaining a list where each element contains a tweet
>  #separated by word into a vector
> test_tweets <- strsplit(as.character(test$Tweet), ' ')
>
> #looping through the list and writing each tweet to a file
>  #each with a different name
> for(i in 1:length(test_tweets)) {
+    write.table(tolower(test_tweets[[i]][test_tweets[[i]] != ""]),
+                file = paste0("tweet",i,".txt"), row.names = F, quote = F, col.names = F)
+ }
```

All of the testing documents can be placed in a single directory.

5.3 Obtaining predictions

Once all preliminary work is finished, the model can be run.

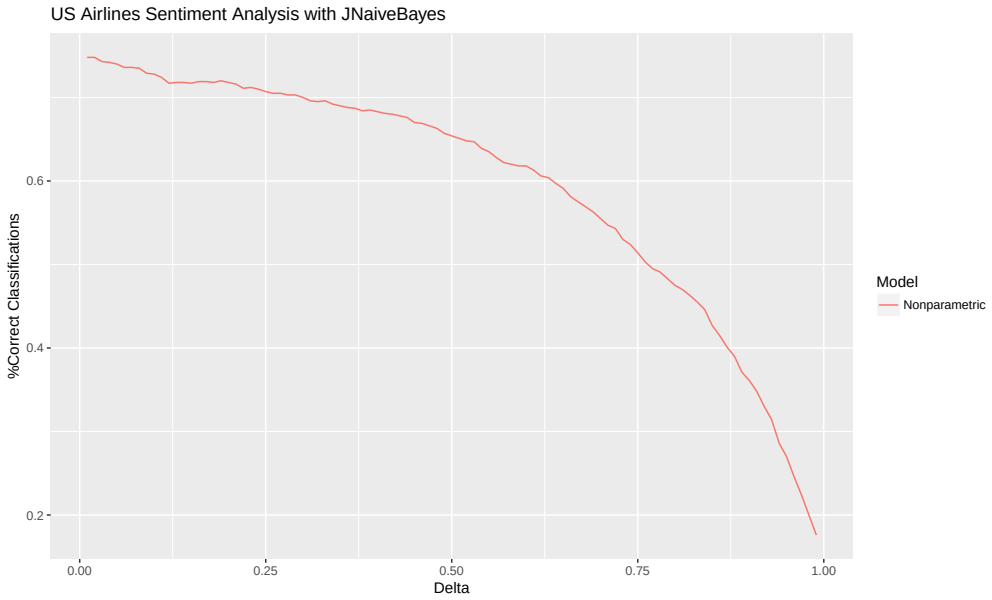
```
> library(JNaiveBayes}
> model <- JNaiveBayes(trainDir = "Sentiments", testDir = "TestTweets", delta = 0.01, parametric = FALSE)
> head(model$Probabilities)
               neutral     positive     negative
tweet1.txt 0.9341720097 2.840133e-02 0.0374266618
tweet2.txt 0.2084022564 1.710277e-02 0.7744949746
tweet3.txt 0.0941010026 8.214944e-01 0.0844046368
tweet4.txt 0.0001182587 4.640731e-03 0.9952410105
tweet5.txt 0.0001059282 9.994558e-01 0.0004382224
tweet6.txt 0.0002144262 2.305706e-05 0.9997625167
> preds <- apply(model$Probabilities, 1, which.max)
> labels <- colnames(model$Probabilities)[preds]
> correct <- as.numeric(labels == test_labs)
> mean(correct) #0.748
>
> #checking how many words are being taken in by the model
> length(negs); length(poses); length(neuts)
[1] 169974 [1] 30849 [1] 41439
> sum(c(length(negs),length(poses),length(neuts)))
[1] 242262
> length(unique(c(negs,poses,neuts)))
[1] 18919
>
> #checking time it takes to carry out analysis
> system.time(JNaiveBayes(trainDir = "Sentiments", testDir = "TestTweets", delta = 0.01, parametric = FALSE))
   user  system elapsed
  1.368   0.055   1.558
```

When using the default settings of the function, the model correctly classified 748 of the 1000 tweets.

5.4 Exploring the tuning parameter

The effect of the the choice of δ can be assessed by examing the classification rate at a range of values.

```
> correct_classifications <- function(delta, parametric) {
+    model <- JNaiveBayes("Sentiments","TestTweets", delta = delta, parametric = parametric)
+    preds <- apply(model$Probabilities, 1, which.max)
+    labels <- colnames(model$Probabilities)[preds]
+    correct <- as.numeric(labels == test_labs)
+    mean(correct)
+ }
> deltas <- seq(0.01, .99, .01)
> results <- sapply(deltas, correct_classifications, parametric = FALSE)
```



It is observed that the choice of δ plays a significant role in the model's accuracy. The plot for the parametric approach was omitted as the results were unusual.