Mining Loan Descriptions with Naive Bayes

Stuart Liu-Mayo

Naive Bayes classifiers are a family of probabilistic classifiers that make use of Bayes rule and the naive assumption of feature independence. While very limited compared to modern exotic models, they have shown remarkable value in text mining, notably for recognizing spam emails, and are attractive for their simplicity and interpretability. Here, I will demonstrate how to train a naive Bayes classifier in R using the `e1071` package and attempt to extract information from the description fields of Lending Club loans.

Data - Lending Club Loans

Lending Club is a peer-to-peer loan service that publishes the data on loans they have originated. Information on 115 different features is provided, which range from FICO score to debt-to-income ratio to the number of derogatory public records on a borrower. Also included is a text description written by the borrower explaining why they want to borrow on the Lending Club platform.

The loan data, including a complete data dictionary, can be found here.

The loan descriptions are diffcult to utilize since they are free-form text. Using a naive Bayes classifier we can try to extract some information connecting the descriptions to the likelihood of negative credit events. At best we might find words that are highly correlated with outcomes and at worst we will produce a numeric value, the posterior probability of default, that can be used as an input to a more complex model.

Exploring and Preparing the Loan Data

First we must read in the data and prepare it for input to naive Bayes. We will have to select only the loans that actually have descriptions (around half during the time period selected have no description at all) and create a new column identifying whether a negative credit event occurred on the loan.

```
# read in the data
loan.data <- read.csv("data\\LoanStats_2012_2013.csv", skip=1, nrows=188181)

# select only rows that include descriptions
has.desc <- which(loan.data$desc != "")
loan.data <- loan.data[has.desc, ]

# define column for having gone bad
bad.statuses <- c("Charged Off", "Default", "Late (16-30 days)",
                  "Late (31-120 days)")
is.bad.loan <- loan.data$loan_status %in% bad.statuses
is.bad.loan <- factor(is.bad.loan, levels=c(TRUE, FALSE),
                      labels=c("Bad", "Good"))

# keep only the descriptions and indicator for badness
loan.data <- data.frame(description=loan.data$desc, is.bad.loan=is.bad.loan)
```

Next we can begin cleaning up the descriptions by removing Lending Club's annotations and residual hypertext tags.

```
# clean up the descriptions a bit
# all descriptions include the leading line:
#    "Borrower added on MM/DD/YY > "
```

```
# let's remove that portion
loan.data$description <- gsub("Borrower added on ../../.. >", "",
                              loan.data$description)
# also remove the <br> tags
loan.data$description <- gsub("<br>", "", loan.data$description)
```

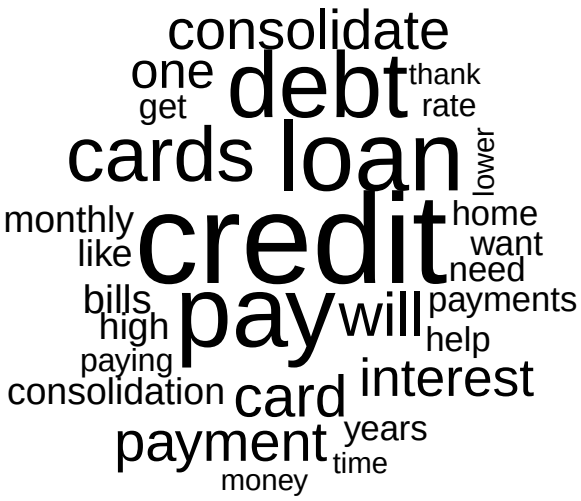
At this point we are ready to transform the text strings into a format that can be interpreted by the naive Bayes algorithm. We will use text mining tools from the `tm` package to build a document-term matrix, a sparse matrix with a row for every loan description and a boolean column for every word.

```
# build corpus of descriptions
desc.corpus <- Corpus(VectorSource(loan.data$description))

# lowercase all the descriptions
cleaned.corpus <- tm_map(desc.corpus, tolower)

# remove stop words, e.g. to, and, but, or
cleaned.corpus <- tm_map(cleaned.corpus, removeWords, stopwords())

# remove punctuation
cleaned.corpus <- tm_map(cleaned.corpus, removePunctuation)

# remove extra whitespace
cleaned.corpus <- tm_map(cleaned.corpus, stripWhitespace)
```

Now, having build up a library of the words that appear in loan descriptions, it is nice to digress briefly to visualize what we have done. Using the `wordcloud` package we can draw word clouds displaying the most common words used.

```
# draw some word clouds
# only good loans
wordcloud(subset(cleaned.corpus, loan.data$is.bad.loan == "Good"),
          min.freq=5000, random.order=FALSE)
```


```
# only bad loans
wordcloud(subset(cleaned.corpus, loan.data$is.bad.loan == "Bad"),
          min.freq=1000, random.order=FALSE)
```



It is hard to see any strong differences between the two groups from their word clouds. Both feature unsurprising words like "credit," "debt," "loan," "pay," and "cards." Clearly we will be hunting in the margins for less common words that may be indicative of trouble, or fitness, as the case may be.

Training and Testing a Naive Bayes Classifier

To be sure of our results, we should use cross-validation to evaluate the classifier. We will start with a 10-fold cv. First, a for loop to wrap the training and testing code:

```
set.seed(1337)
idxs <- sample(1:length(cleaned.corpus))
n.in.fold <- ceiling(length(cleaned.corpus) / 10)
crosstab <- matrix(numeric(4), ncol=2)
for (k in 1:10) {

    # testing and training code

}
```

Within each cv fold, we have to select the testing and training sets and then build the final document-term matrix from the testing set. To speed up the training, we want to include only words that appear in a significant number of loan descriptions in our training set. Therefore we will build the document-term matrix using as features the words that occur in at least 0.1% of training set loan descriptions.

```
    # select the elements in the fold
    first <- (k - 1) * n.in.fold + 1
    last <- min(k * n.in.fold, length(cleaned.corpus))
```

```
    in.fold <- first:last

    # separate into train and test sets
    # unprocessed data frame
    is.bad.train <- loan.data$is.bad.loan[-in.fold]
    is.bad.test <- loan.data$is.bad.loan[in.fold]
    # corpus
    corpus.train <- cleaned.corpus[-in.fold]
    corpus.test <- cleaned.corpus[in.fold]

    # build document-term matrix using only frequently-used words
    desc.dtm.train <- DocumentTermMatrix(corpus.train)
    desc.dict <- findFreqTerms(desc.dtm.train,
                               ceiling(length(corpus.train) * .001))
    desc.dtm.train <- DocumentTermMatrix(corpus.train,
                                         list(dictionary=desc.dict))
    desc.dtm.test <- DocumentTermMatrix(corpus.test,
                                        list(dictionary=desc.dict))
```

We now have a document-term matrix composed of counts of the number of times a word appeared in each loan description. The basic naive Bayes classifier, however, only cares about whether or not a word appears at all, so we must convert our matrix of counts to a matrix of booleans.

```
    # convert the counts in the document-term matrix to booleans
    ConvertCounts <- function (x) {
        x <- ifelse(x > 0, 1, 0)
        x <- factor(x, levels=c(0, 1), labels=c("No", "Yes"))
        return(x)
    }
    desc.dtm.train <- apply(desc.dtm.train, 2, ConvertCounts)
    desc.dtm.test <- apply(desc.dtm.test, 2, ConvertCounts)
```

This leaves us with a training matrix and a testing matrix that can be used as input to a naive Bayes classifier. In R, such classifiers are implemented well by the `e1071` package.

```
    # train the model!
    loans.nbclassifier <- naiveBayes(desc.dtm.train, is.bad.train)
```

Then we can use the model to make predictions on the test set and store them in a combined confusion matrix for all cross-validation folds.

```
    # evaluate it on the test set
    loans.test.pred <- predict(loans.nbclassifier, desc.dtm.test)

    # record results from the fold
    classification <- c("Bad", "Good")
    for (i in 1:2) {
        for (j in 1:2) {
            crosstab[i, j] <- (crosstab[i, j]
                               + sum((is.bad.test == classification[i]
                                      & loans.test.pred == classification[j])))
        }
    }
```

After running the whole 10-fold cross-validation, we can quickly inspect the results in confusion matrix form:

```
# print out the confusion matrix
crosstab <- cbind(crosstab, rowSums(crosstab))
crosstab <- rbind(crosstab, colSums(crosstab))
rownames(crosstab) <- c("actual.bad", "actual.good", "sum")
colnames(crosstab) <- c("predicted.bad", "predicted.good", "sum")
crosstab
```

```
##             predicted.bad predicted.good   sum
## actual.bad             62          12247 12309
## actual.good           412          68757 69169
## sum                   474          81004 81478
```

```
round(crosstab / crosstab[3, 3], digits=4)
```

```
##             predicted.bad predicted.good    sum
## actual.bad         0.0008         0.1503 0.1511
## actual.good        0.0051         0.8439 0.8489
## sum                0.0058         0.9942 1.0000
```

These results can feel pretty disappointing. When the classifier predicts that a loan will go bad, there is only around a 15% chance that it really will. Similarly, when it predicts the loan is good there is only around an 85% chance that is really is. This is basically the same as our starting overall proportions!

However, one shouldn't become too disheartened yet. These predictions came from only one of over 100 features so we shouldn't be surprised that there is very little information available to extract with such a simple model.

The important product of this model is the posterior probability, which is a real number:

```
# predictions in raw posterior probability form
loans.test.pred.raw <- predict(loans.nbclassifier, desc.dtm.test, type="raw")
head(loans.test.pred.raw)
```

```
##             Bad      Good
## [1,] 0.05652436 0.9434756
## [2,] 0.18721979 0.8127802
## [3,] 0.09719732 0.9028027
## [4,] 0.04782134 0.9521787
## [5,] 0.29402216 0.7059778
## [6,] 0.03363672 0.9663633
```

Effectively, we have mined the text descriptions for a single number, which is interpreted as the posterior probability of negative credit event, and which can be used in a more exotic model.

Improving the Naive Bayes Classifier

In this simple example, I have neglected some further refinements that can be made and that usually improve the performance of the classifier. Firstly, in constructing the document-term matrix, I discarded all words that did not appear in more than 0.1% of training examples. When scratching at the margins for diminishing returns of predictive power, including those uncommon terms at the cost of a much wider document-term matrix can help.

Additionally, I did not use any Laplace smoothing in the classifier. Laplace smoothing helps when words that have never been seen before show up in test data. If we include all words in the document-term matrix, including rare words, then smoothing will help performance when new words are encountered.

Finally, I did not make any attempt to identify or correct misspellings or to combine similar words. For example, words like pay, payment, and payments should probably be grouped together instead of counted

separately. This can pay off when grouping rarer words reveals their significance, but is a more labor-intensive task.