CYO Project

Daniel Cash

May 13, 2019

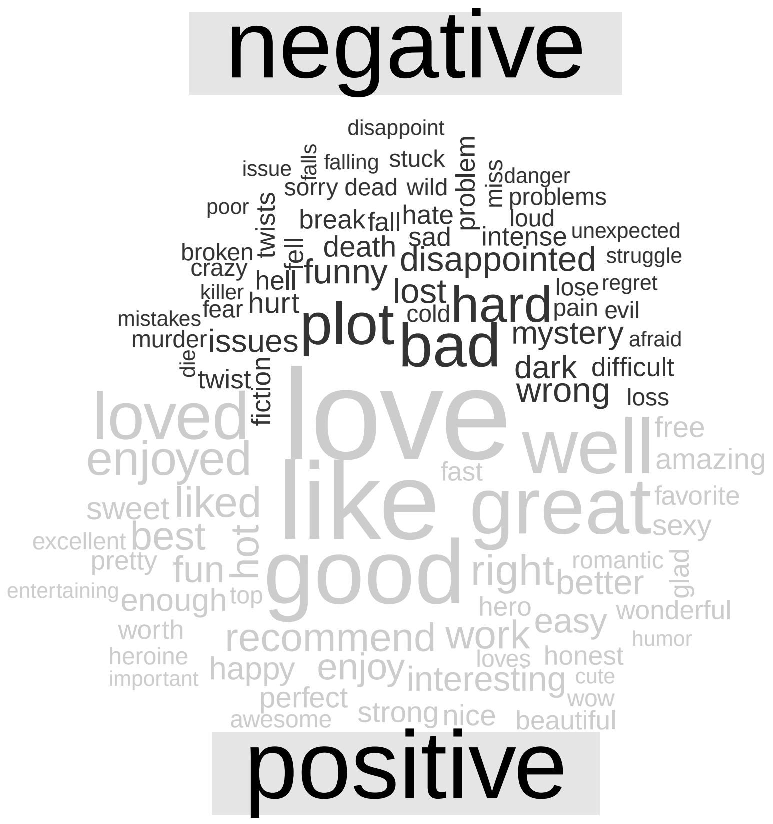
Amazon Kindle rating prediction: Introduction

For my capstone project I will be constructing a predictive model for Amazon Kindle user ratings. Since the goal will be classification by doing sentiment analysis, the only data that will be used to predict ratings will be the review text itself. The original data can be found on http://snap.stanford.edu/data/amazon/productGraph/categoryFiles/reviews_Kindle_Store_5.json.gz and the json file contains millions of records. Only a small subset will be used for this project due to hardware restrictions. This way the code will run in a timely manner. The R code will be included in the report to make it easier to follow along. The metrics that will be used in evaluating the models are accuracy and F1. F1 is a balanced, weighted average of precision and recall, with no preference towards either. F1 is a good metric for this problem since no outcome is better or worse than another.

Methods and Analysis

First we will load the required packages and data into R. The data has a lot of unnecessary data for our analysis and will need to be tidied. Only 1000 records will be used from the dataset. The ratings will split into two groups for this classification problem. Anything less than a three star rating will be labeled a negative rating and anything else a positive.

Before the data is split, some exploratory analysis is conducted. A wordcloud will reveal frequency and sentiment at the same time. The wordcloud below uses the bing sentiment lexicon which assigns words into positive and negative categories.



This doesn't reveal how balanced the dataset is, however. Using the classification rubric, a two dimensional matrix will be constructed using just the text and the rating sentiment. After, a comparison of the count of each label will reveal the disparity between them. This means it will be a difficult task to increase the F1 statistic.

```
##
## negative positive
##       57      943
```

The bag of words approach will be used. Each word is represented as a feature and each document a vector of features. Word order is disregarded. Prior to building the model, the vectors are converted into a corpus, a large and structured set of texts. The corpus will be cleaned for punctuation, numbers, whitespaces, stop words, and will be converted to lowercase for easier comparison. The corpus is then converted into a Document Term Matrix. Each row (document) is a review's text. The words are laid out in matrix with words and the occurrence of the words in the documents. The data is then split into training and testing sets.

```
corpus <- VCorpus(VectorSource(text))
corpus.clean <- corpus %>%
  tm_map(content_transformer(tolower)) %>%
  tm_map(removePunctuation) %>%
  tm_map(removeNumbers) %>%
  tm_map(removeWords, stopwords(kind="en")) %>%
  tm_map(stripWhitespace)

dtm <- DocumentTermMatrix(corpus.clean)

split_val <- floor(0.8 * nrow(reviews))
train_ind <- sample(seq_len(nrow(reviews)), size = split_val)

df.train <- reviews[train_ind, ]
df.test <- reviews[-train_ind, ]

dtm.train <- dtm[train_ind, ]
dtm.test <- dtm[-train_ind, ]

corpus.clean.train <- corpus.clean[train_ind]
corpus.clean.test <- corpus.clean[-train_ind]
```

Three models for comparison are shown below. The first restricts the document term matrix to only terms that appear at least five times. The second uses a normalized term frequency, Tf-Idf, which measures the relative importance of a word to a document. The third uses bigrams and removes sparse bigrams.

```
fivefreq <- findFreqTerms(dtm.train, 5)

dtm.train.nb <- DocumentTermMatrix(corpus.clean.train, control=list(dictionary = fivefreq))
dtm.test.nb <- DocumentTermMatrix(corpus.clean.test, control=list(dictionary = fivefreq))

tfidf.dtm.train.nb <- DocumentTermMatrix(corpus.clean.train, control=list(weighting = weightTfIdf))
tfidf.dtm.test.nb <- DocumentTermMatrix(corpus.clean.test, control=list(weighting = weightTfIdf))

BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
bi.dtm.train.nb <- DocumentTermMatrix(corpus.clean.train, control = list(tokenize = BigramTokenizer))
bi.dtm.train.nb <- removeSparseTerms(bi.dtm.train.nb,0.99)
bi.dtm.test.nb <- DocumentTermMatrix(corpus.clean.test, control = list(tokenize = BigramTokenizer))
bi.dtm.test.nb <- removeSparseTerms(bi.dtm.test.nb,0.99)
```

A binary conversion function is used to label word frequencies as present or absent and is then applied to the document term matrices. The reasoning behind this is that for sentiment classification word occurrence matters more than frequency.

```
convert_count <- function(x) {
  y <- ifelse(x > 0, 1,0)
  y <- factor(y, levels=c(0,1), labels=c("No", "Yes"))
  y
}

trainNB <- apply(dtm.train.nb, 2, convert_count)
testNB <- apply(dtm.test.nb, 2, convert_count)
tfidf.trainNB <- apply(tfidf.dtm.train.nb, 2, convert_count)
tfidf.testNB <- apply(tfidf.dtm.test.nb, 2, convert_count)
bi.trainNB <- apply(bi.dtm.train.nb, 2, convert_count)
bi.testNB <- apply(bi.dtm.test.nb, 2, convert_count)
```

The data is now ready for training and prediction. Naive Bayes evaluates the products of probabilities, which creates problems for the model due to words that do not occur in the sample (0 probability). Therefore, Laplace smoothing is used to assign a small, non-zero probability.

```r
classifier <- naiveBayes(trainNB, as.factor(df.train[,"sc"]), laplace = 1)
tfidf.classifier <- naiveBayes(tfidf.trainNB, as.factor(df.train[,"sc"]), laplace = 1)
bi.classifier <- naiveBayes(bi.trainNB, as.factor(df.train[,"sc"]), laplace = 1)

pred <- predict(classifier, newdata=testNB)
tfidf.pred <- predict(tfidf.classifier, newdata=tfidf.testNB)
bi.pred <- predict(bi.classifier, newdata=bi.testNB)
```

Truth tables will show what was and wasn't correctly classified. Then the confusion matrices will be constructed.

```
## [1] "Five"

##            Actual
## Predictions negative positive
##    negative        1        7
##    positive       12      180

## [1] "Tf-Idf"

##            Actual
## Predictions negative positive
##    negative        4       22
##    positive        9      165

## [1] "Bigrams"

##            Actual
## Predictions negative positive
##    negative        1        2
##    positive       12      185
```

The results of the confusion matrix will help determine which model tested better.

```
## [1] "Five"
```

```
##          Sensitivity          Specificity       Pos Pred Value
##           0.07692308           0.96256684           0.12500000
##       Neg Pred Value            Precision               Recall
##           0.93750000           0.12500000           0.07692308
##                   F1           Prevalence       Detection Rate
##           0.09523810           0.06500000           0.00500000
## Detection Prevalence    Balanced Accuracy
##           0.04000000           0.51974496
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.90500000     0.04809619     0.85562336     0.94183039     0.93500000
## AccuracyPValue  McnemarPValue
##     0.96266961     0.35879536
```

```
## [1] "Tf-Idf"
```

```
##          Sensitivity          Specificity       Pos Pred Value
##            0.3076923            0.8823529            0.1538462
##       Neg Pred Value            Precision               Recall
##            0.9482759            0.1538462            0.3076923
##                   F1           Prevalence       Detection Rate
##            0.2051282            0.0650000            0.0200000
## Detection Prevalence    Balanced Accuracy
##            0.1300000            0.5950226
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.84500000     0.12970241     0.78726236     0.89219060     0.93500000
## AccuracyPValue  McnemarPValue
##     0.99999766     0.03114121
```

```
## [1] "Bigrams"
```

```
##          Sensitivity          Specificity       Pos Pred Value
##           0.07692308           0.98930481           0.33333333
##       Neg Pred Value            Precision               Recall
##           0.93908629           0.33333333           0.07692308
##                   F1           Prevalence       Detection Rate
##           0.12500000           0.06500000           0.00500000
## Detection Prevalence    Balanced Accuracy
##           0.01500000           0.53311394
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##     0.93000000     0.10313901     0.88534027     0.96120443     0.93500000
## AccuracyPValue  McnemarPValue
##     0.67865872     0.01615693
```

The range of the accuracy is not too large, but the F1 score is either N/A or very low. In order to boost this k-fold cross validation will be used. The steps used before will be repeated 10 times in a loop to find the optimal number of folds and the best model.

Results

```
## Five - # of folds:2

## Tf-Idf - # of folds:4

## Bigrams - # of folds:2

## Five - F1:0.4

## Tf-Idf - F1:0.352941176470588

## Bigrams - F1:0.4

## Five - Accuracy:0.94

## Tf-Idf - Accuracy:0.78

## Bigrams - Accuracy:0.97

## Always guess positive:0.943
```

The at least five frequency model improved quite a bit and now beats the Tf-idf model in both F1 and accuracy. Blindly guessing a positive sentiment (3 or higher) will lead to a high accuracy, higher than all but the bigrams model. This shows how imbalanced the dataset is and why the F1 score is important. The Bi-gram model preformed the best in terms of accuracy and F1 score and therefore is the best model.

Conclusion

The initial models that did not use cross validation had little ability to predict true negatives even though the accuracy was high. With cross validation both the accuracy and F1 scores improved. Despite the simplicity of assumptions, the Naive Bayes algorithm does reasonably well. Only a small subset of the data was used due to hardware restrictions. Future analysis could use more data and would produce better models.