Text Classification

Nura Kawa

December 8, 2016

Introduction

This is an independent research project, supervised by Professor David Aldous. My goal is to learn basic concepts of Natural Language Processing and the role that probability plays in language modeling. All code, images, and data for this project can be downloaded online at github.com/nurakawa/text-classification .

Project Description

This project involves five steps:

1. Data Pre-Processing
2. Language Modeling
3. Classification
4. Visualization
5. Evaluation of Results

Data Pre-Processing

Downloading Data

I use the open-source BBC Dataset, available here: http://mlg.ucd.ie/datasets/bbc.html.

The dataset contains 2,225 articles from the BBC news Ibsite corresponding to stories in five topical areas from years 2004-2005. Each article is labeled with one of the following five classes: `business`, `entertainment`, `politics`, `sport`, and `tech`.

I downloaded the dataset and wrote the script `import-data.R` to output the data as a two-column data frame, with one row per article. The first column contained the document text, while the second column contained the class labels.

Splitting into Training and Testing

Next, I split the data frame into training and testing sets, randomly assigning 70% of the data to a "training" set and leaving the remaining 30% for "testing". The purpose of this step is for classification. Classification models are fit to training data, which represents past observations, in order to predict the testing data, which would represent new observations. It is important to perform this step *before* creating a language model in order to avoid the mistake of training our model on data that is supposed to be unseen. Thus, I create two copies for each language model: one with 70% of the dataset, and another with only 30%.

Language Modeling

The goal of language modeling is to extract information from a *corpus*, or set of documents. One can engineer a variety of features from text, such as word length, or frequency of punctuation. The most descriptive and useful is *term frequency*, or the distribution of counts per word in each document.

Next, I discuss two language models that I used to fit my corpus: "Bag-of-Words" and "N-Gram". Because my corpus is quite large, I demonstrate each method with a toy example.

Bag-of-Words Model

The "Bag-of-Words" model treats a corpus as, quite literally, a "bag" of words. Each word in the document is treated as a term whose probability of occuring is independent of preceding or proceeding terms. To fit this model, one splits a corpus into a list of its words and counts the frequency of each term per document.

This is quite doable for large sets of documents using the R packages `{tm}` and `{stringr}`. I demonstrate this below with a quick example from a small portion of *The Tale of Two Cities* by Charles Dickens:

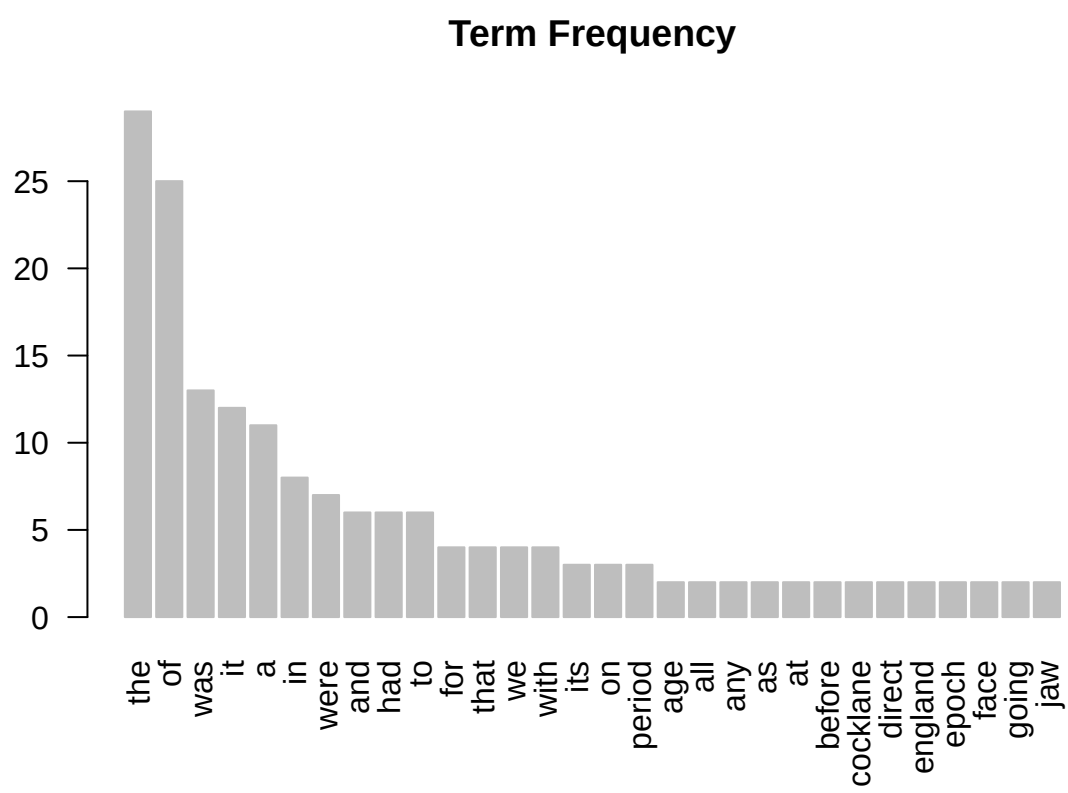
```
## [1] "It was the best of times, it was the worst of times, it was the age of wisdom,"
```

```
## [1] "it was the age of foolishness, it was the epoch of belief,"
```

Here, our corpus is the book. Our "bag of words" becomes a list of all terms:

```
##  [1] "it"          "was"         "the"         "best"        "of"
##  [6] "times"       "worst"       "age"         "wisdom"      "foolishness"
## [11] "epoch"       "belief"      "incredulity" "season"      "light"
## [16] "darkness"    "spring"      "hope"        "winter"      "despair"
## [21] "we"          "had"         "everything"  "before"      "us"
## [26] "nothing"     "were"        "all"         "going"       "direct"
```

To numerically express this bag-of-words, I simply calculate term frequency for each sentence.



Notice that many of the most frequent words are "stop-words", or commonly-used words that give no additional meaning to our work. These include articles, pronouns, and conjuctions, such as "the, to, for, he, and". Removing these stop-words gives more information about the content of the document that you are breaking down. I perform this step when fitting the bag-of-words model to the training and testing data.

N-gram Model

An *N-gram* is a phrase of word length N. Parsing our toy example into N-grams reveals the following terms:

For N=2, the top 15 terms:

```
##  [1] "it was"    "was the"   "with a"    "in the"    "of the"
##  [6] "to the"    "a king"    "a large"   "a queen"   "age of"
## [11] "all going" "and a"     "before us" "epoch of"  "face on"
```

For N=3, the top 15 terms:

```
##  [1] "it was the"       "a king with"      "a large jaw"
##  [4] "a queen with"     "all going direct" "and a queen"
##  [7] "before us we"     "face on the"      "jaw and a"
## [10] "king with a"      "large jaw and"    "of times it"
## [13] "on the throne"    "queen with a"     "the age of"
```

The N-gram model uses conditional probability to model a language or document. A term's probability of occurance in a document is conditioned on the preceding N-1 words.

For example, the probability of seeing the tri-gram "american presidential election" is:

$\mathbf{P}$("american")$\mathbf{P}$("presidential|american")$\mathbf{P}$(election|american presidential)

In our example document, the bi-gram "of times" exists twice in the first sentence: "It was the best **of times**, it was the worst **of times**." In the bag-of-words model, the words "of" and "times" would be counted as occuring twice; yet, their functionality as a bi-gram gives a different meaning to the writing style of Charles Dickens. More relavant examples are N-grams seen in our data, such as "stock exchange", "labour party", and "french open championship", reveal a different meaning than from each word on its own.

In fact, one can derive easily that the Bag-of-Words model is equivalently the N-gram model with N=1.

Term Stemming

While not itself a model, it is important to mention the concept of term-stemming, where endings are removed from words to avoid highly-correlated columns and retain information. An example of stemming is shown below using a basic function that I wrote called "stem_word":

```
stem_word("stationary")
```

```
## [1] "station"
```

```
stem_word("player")
```

```
## [1] "play"
```

```
stem_word("played")
```

```
## [1] "play"
```

```
stem_word("running")
```

```
## [1] "runn"
```

The example of "player" and "played" shows that stemming can reduce the number of terms in our language model and eliminate the problem of high correlation. However, stemming causes the problem of loss of information. For example, the words "stationary" and "station" have different meanings, but stemming reduces them to the same term.

Classification

After fitting a language model to the training and testing sets, I fit a multi-nomial classificaiton model. *Classification* is the process of assigning a class to new data given a set of past entries. Our goal is to be able to read in a new document and, using a classification algorithm, predict what type of article the document is (business, sports, etc.). There are two methods: one is unsupervised, where prior class labels are not used, and the other is supervised, where class labels of known cases are used in validating the model.

Unsupervised: K-Nearest Neighbor Classification

I ran KNN classification testing various K, from 1 to 20. I found no success in this method. K-Nearest Neighbor looks to the "nearest neighbors", or most similar documents, of a new document, and classifies the new document based on those most similar to it. This method has the possibility of working well if the term frequencies are unique enough such that clusters are clearly found within the corpus. Running clustering algorithms to check, I found many individual clusters appeared, with high within-cluster variances. This shows that the algorithm found several classes within the documents, splitting them beyond simply five categories. Perhaps withing "business" there are distinctions betIen articles written about the stock market and articles written about start-up companies; perhaps "sports" reveals quite a distinction betIen football and rugby. More exploration is necessary to see if this hypothesis is true; regardless, a high variance in the

information and a large number of terms makes K-nearest neighbor an unsuitable classificaiton method for our purposes.

Supervised

Next I discuss supervised learning, using two highly successful models: Logistic Regression and Support-Vector Machines

Logistic Regression with {glmnet}

I fit a penalized multinomial logisitc regression model using the L-1 penalty (this is the same penalty used in the LASSO algorithm). I played with Stemming (mentioned above) and tested for a change in results.

Logsitic Regression model fits a regression line to data with categorical response (in our case, type of article) from binary predictors (in our case, a term-frequency matrix). The L-1 penalty is added as a constraint to optimization in order to shrink non-important coefficients (predictors) to 0.

Support-Vector Machines

A support vector machine for multiple classes fits several hyperplanes in a high-dimensional space to separate data points and assigns classes to each area of the plane. Thus, no "regression line" is built - rather, this model separates data points with highest margins.

This model worked nearly equally well as logistic regression.

Results

The table below shows the prediction accuracy of each language/classification model attempted. I do not include KNN classification due to its extremely poor performance (less than 50% accuracy).

GLMNET	GLMNET with Stemming	SVM	SVM with Stemming
95.51	95.21	95.96	95.21
88.17		91.47	
70.36		72.01	
43.86		36.23	

Table 1: Results

Notice that N-gram models perform poorly for N=3 and N=4. It is important to note that the higher the N value, the more data is needed to train a model. That is because the longer the sequence of words, the less likely I are to see it our data. It is therefore easy to see why I get such poor results fitting models with N=3 and N=4 - I would need a much larger corpus of documents to increase prediction accuracy.

Conclusion

I concluded that, with a 95.96% accuracy, the best way to classify the BBC dataset is to model it with bag-of-words, stem the terms, and fit a Support Vector Machine. Logistic Regression performs nearly equally well. Interestingly, a Support Vector Machine improves the fit for an N-gram of N=2 model by over 3%. Thus, we see that a Natural Langauge Processing for classification can be improved with fitting different machine-learning algorithms, but that simply using an algorithm without paying attention to the quality of the language model proves futile.

Further Exploration

Given more time, I would test different algorithms and play with more NLP methods to ensure a higher-quality data set. I would also repeat this project with different data sets (including some of larger dimesion) and see if my findings are reproducible.

Appendix: Word Clouds

Figure 1: Wordcloud: bag-of-words model of 'business'

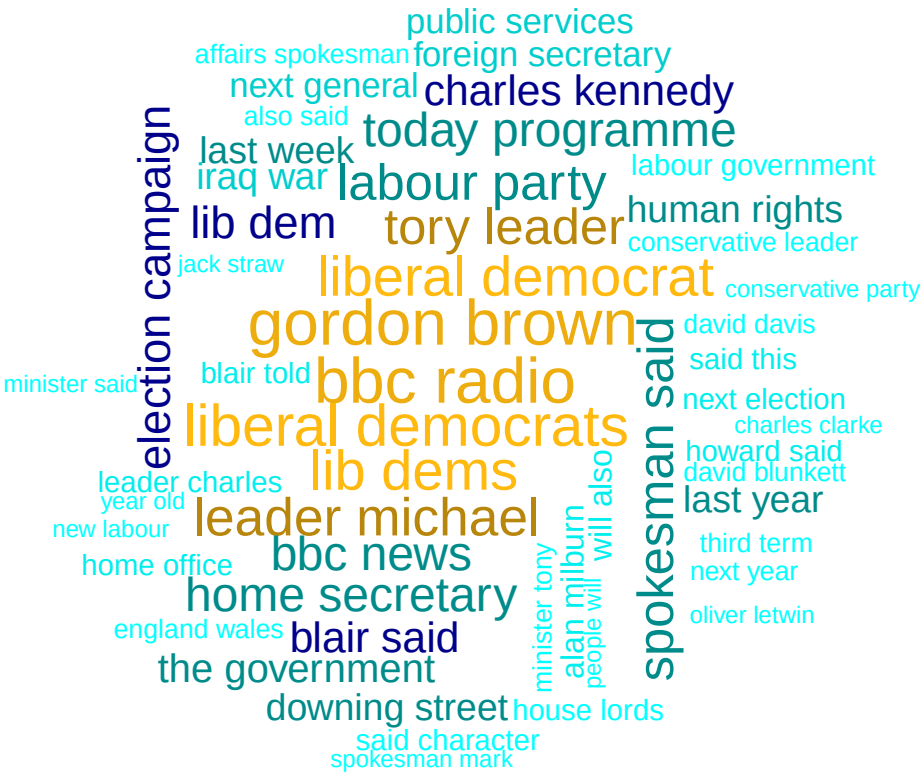


Figure 2: Wordcloud: bi-gram model of 'politics'

Figure 3: Wordcloud: tri-gram model of ‘sports’