Data Science Capstone Project: NLP model

Miao YU

2014/10/10

Introduction

Portable office actually means the works done on the cellphones or tablets and we need input system to saving our time on typing on those device. So a smart and efficient keyboard is required. The core of this input system is a natural language processing model. This report is focused on this, covering from the very beginning, namely data collection, to the final model. The first parts came from the milestone report with a revision.

Data Acquisition and Cleaning

The data were downloaded from the course website (from HC Corpora) and unzipped to extract the English database as a corpus. Three text documents from the twitter, blog and news were found with each line standing for a document.

Data Pre-Summary

After scan the three documents with `bash`, the basic summary of the data set is shown as follows:

	line counts	word counts	document size
twitter	2360148	30373603	166816544
news	1010242	34372530	205243643
blogs	899288	37334147	208623081

Table 1: Summary of the datasets

I found twitter is short(of course less than 140 words) with a lot of informal characters and less grammar, which means more noise; news is written in a formal manner but the topics is focused; blog's pattern is between the twitter and news with less noise and more topics; the average length of each lines in the three database: blog > news > twitter, which means blog is the longest document class and longer document will help to build a better model for prediction in certain context.

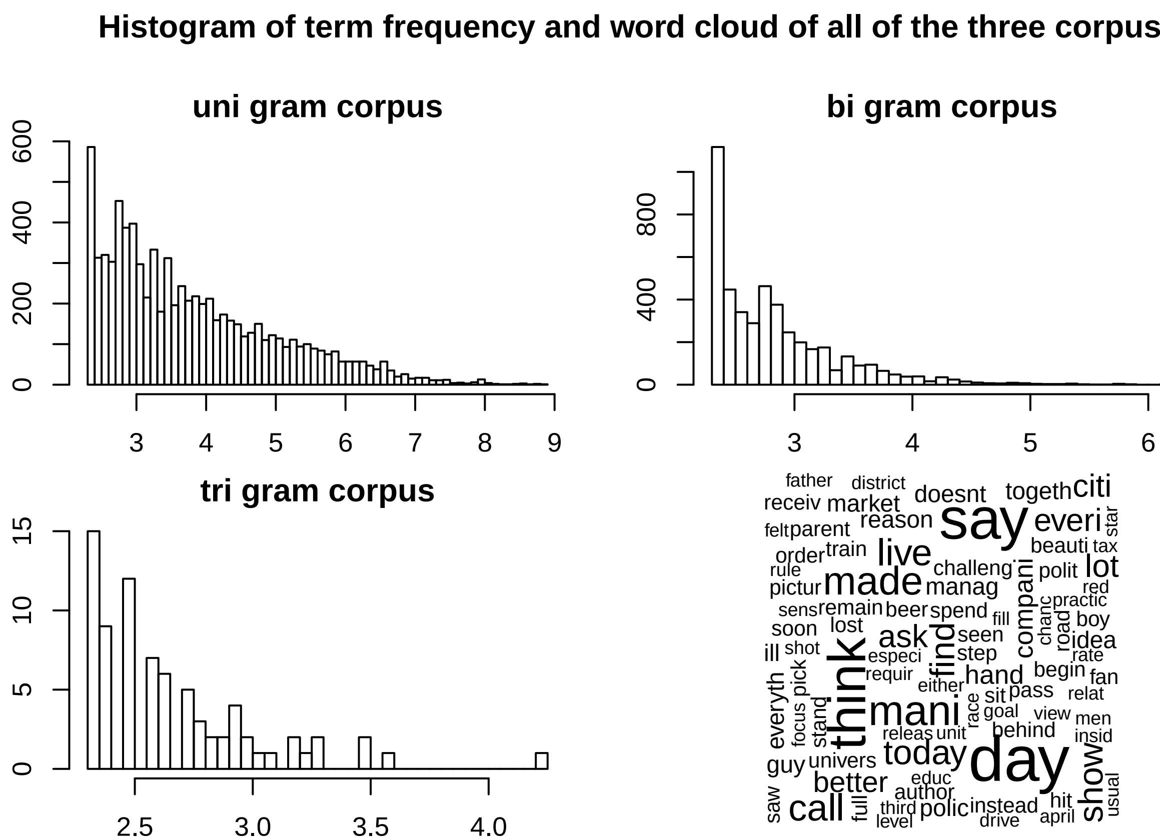
So, the blog data will be good for us to build a model if those three document is too large to be loaded for exploring. However, using sampling will ease the burden on the calculation and finally I sampled 30,000 20,000 and 10,000 lines with seed from the blogs, news and twitter database for exploring and the left data will be sampled to make the test data sets.

Tokenization

The whole tokenization is aiming at removing meaningless characters and the words with low frequency to avoid overfit in the corpus. The final corpus will show the words or terms with a high frequency which will be helpful for exploring the relationship between the words and building a meaningful statistical model.

So, I extracted 1)the ASCII characters, 2)changed the capital characters to lower case, 3)removed the punctuation, 4)numbers and 5)stop words and 6)stemmed the left words to get the corpus. The dirty words were not removed because I decided to remove them from the model predicted words from a web dictionary.

Exploratory analysis



I choose a n-gram model scheme for the exploratory analysis. I extracted n-gram corpus with the help of `RWeka` package. The uni-gram terms corpus has 8063 words, the bi-gram corpus has 4551 terms and the tri-gram corpus has 78 terms.To decrease the spares of the term frequency, I removed the terms occurred less than ten times in the whole documents. Then I explored three corpus(uni-gram, bi-gram and tri-gram) and made a histogram to show the distribution of the terms in them.

As shown in Figure 1, the logged frequencies in all of the three corpus were still skewed to the left, which mean the sparse of the terms data. So it will be hard to build a good global regression model but local model would be OK.

Also I found only 8063 words occurred more than ten times in the sampled documents compared with nearly 70 thousand words in an online dictionary, which mean focused on little words would work in most of the prediction. The word cloud showed the terms occurred more than 400 and those terms would be good to build a classification filter models before using a n-gram model to speed up the whole prediction.

From the exploratory analysis I summaries the following items to build the model:

- tha data obeied Zipf's law: human language has many low frequency word types and relatively few high frequency word types.
- using current laptop to process the whole data set will be time-consuming

- each data set will be overfit on the most common words while underfitted on the least common words
- a pre-classification of the words before will be helpful to narrow down the scales
- using different sources of documents will be helpful for a supervised learning while a PCA analysis of the documents will also be helpful for a unsupervised learning to group the source

Modeling

Pre-classification

I sampled the origin train data to get a corpus without capital characters, punctuation, number and stop words. I didn't stem the words because I found stemming would make the prediction unclear. The most frequency words were used to classify the original data into different sources. Here I use news, blogs and twitters as three different sources. However, only three sources made the model building very slow. So I used the most frequency 115 words occurred more than 1000 times in 60000 samples were used to class the sentences by a support vector machines. Results were bad and it seemed all of the sentences will be grouped into the twitter group. This is a proof for that twitter will cover most kinds of words. I dropped this idea but I think classification will be helpful to get many sub-models to speed up the whole models. Maybe some unsupervised learning models will be useful to get some topic groups for the classification and local prediction under a topic might be better for global prediction.

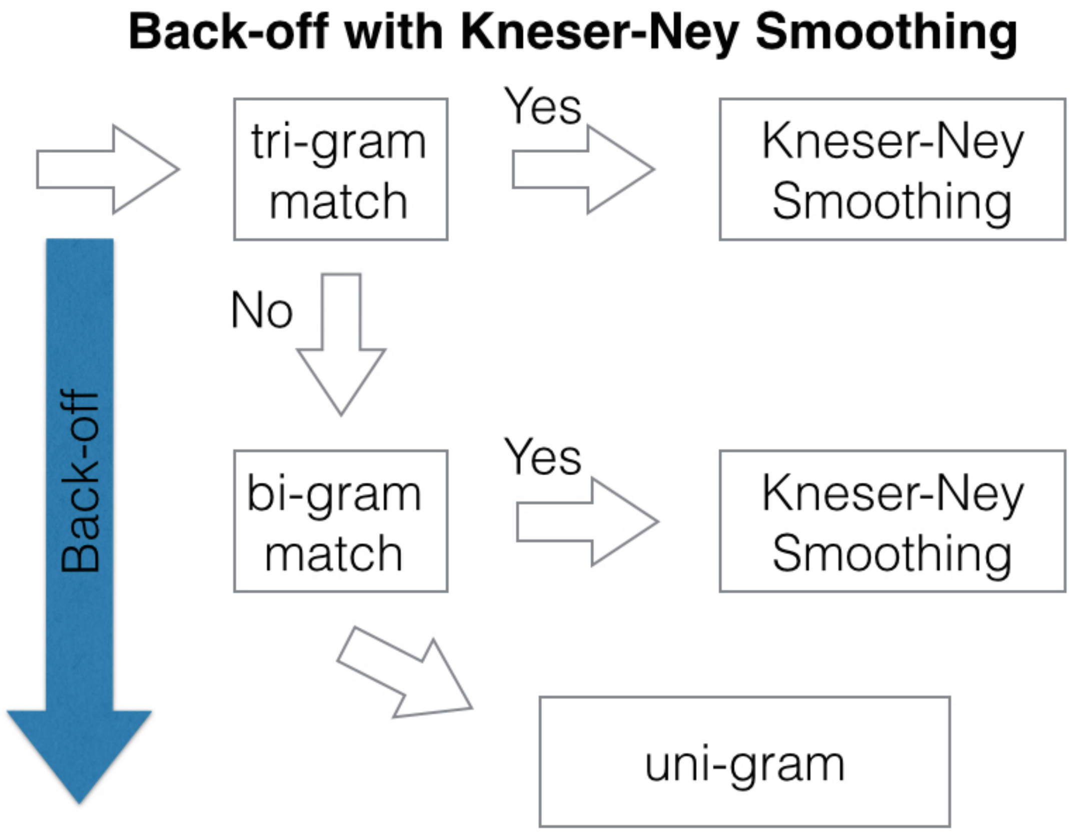
N-grams Extraction

The extraction of n-gram is the base of the model. After I get the n-gram corpus with `RWeka` package, the term-document matrix were used to get the counts for each n-grams by a row sum. It will really take a long time to get the n-gram(uni-gram, bi-gram and tri-gram in this study). To speed up I only extract the terms with a at least occurrences of six times in the whole corpus. The idea is that low frequency words will cause overfit for certain corpus and increase the burden on the calculation or corpus building; more data at certain frequency threshold will subset the meaningful terms from noisy terms. For example, "aaaaaaaaaaa" and "capstone" might both occur two times in a small corpus. When we get a ten times bigger corpus, "aaaaaaaaaaa" might still occur two or at most twenty times. But a much more meaningful word such as "capstone" might at least occur twenty times. Considering a bigger corpus will induce more unigrams, the threshold might set lower than ten.

In this study, I explored the data with 60,000 sentences but my final corpus is twenty times bigger than exploratory analysis and I only use a threshold of six to get a smaller but much meaningful corpus for next step modeling. For different corpus, the threshold is actually a parameter to be optimized. On my PC, less than five will make the n-gram extraction very slow and six was my last choice and I got a uni-gram with 81,456 words, bi-gram with 413,605 words and tri-gram with 49,670 words. Using a higher probability of the last words in terms will predict next words for certain sentence and this is the core of N-gram model.

Backoff and Smoothing

The n-gram model worked well if the terms were huge enough to cover any cases. However, such model will cost a lot of time training the data. Another way is just using a back-off model to change n-gram model into (n-1)-gram model. The simplest back-off model will first get the probability of every (n-1) terms, order them and show the first few words as prediction. When no words were shown, a (n-1)-gram model will be used until uni-gram model, which will show the most common words in the corpus. A simple workflow is shown in the following picture.



However, such case that there were only one terms in a tri-gram while many terms in a bi-gram for certain words will make a simple back-off model hard to distribute the probability to the candidates. A common way is that smoothing the counts on the n-gram. Good-Turing Estimate show a good idea to get the probability space for (n-1)-gram model. The probability of unseen n-gram p is calculated by

$$p = \frac{N_1}{N}$$

N is the total number of terms observed in the corpus. This probability could be saved by a discount on the observed terms' counts. I use an absolute discounting on each counts based on Ney et al.'s study:

$$D = \frac{n_1}{n_1 + 2 * n_2}$$

n_1 and n_2 were terms occurred exactly once and twice.

After I get the free space for the (n-1)-gram model, the Kneser-Ney Smoothing were employed to get the probability with a combination of (n-1)-gram. I actually combined a Kneser-Ney Smoothing with a back-off model: When the model could find terms in the tri-gram, a tri-gram Kneser-Ney model will be used. While the model can't find a hit in tri-gram, a bi-gram Kneser-Ney Smoothing were run. Those two Kneser-Ney Smoothing have different discountings. The detailed formula could be found in Körner's paper in the reference and the code for the whole process is shown in the RMD document's code chunk.

Stupid Backoff Implementation

The model above were slow to show predicted words and I speed up the model with less code and a relative small loss of prediction accuracy. The core of the code called Stupid Backoff implementation, which is often used in web-based corpus. The core of this backoff implementation is that using a fixed discount for (n-1)-gram's possibiliy. With a huge corpus, the performance of Stupid Backoff implementation will show a similar prediction accuracy with the Kneser-Ney Smoothing. The main function code is shown in the RMD document's code chunk.

Prediction

Well, though I spent a lot of time understanding the algorithms in the model, the performance of the final models were still poor in the course quiz. I tried to write a function to test the model on the test set but I can't find a fair way to show the results. The model works fine with common terms while bad on unusual terms. I thought the main reason is that I only use a small part of the documents to build my model and the model is underfit. However, I suggest the reader to try it on the web app and a subjective feeling might be "objective" in this kind of model.

Summary

- The data were really BIG for PC
- Intuition could help exploratory analysis
- Data clean should not remove the signals from noises
- N-grams could be get by a threshold using big data
- Back-off and smoothing is necessary
- Kneser-Ney Smoothing is slower than Stupid Backoff Implementation
- Next word prediction is a more subjective model relying on the corpus
- Pre-classification might help to get a local best prediction
- Google is REALLY an important tool for data scientist
- this capstone project's topic is totally new to me and I am looking foreword to hearing from you

References

- Körner, M. C. (n.d.). Implementation of Modified Kneser-Ney Smoothing on Top of Generalized Language Models for Next Word Prediction Bachelorarbeit, (September 2013).
- Williams, G. (2014). Data Science with R Text Mining.
- Coursera Discussion Board