CS 4740 Project 1 Part 1 report

Rui Liang, Gi Yoon Han, Xiaoyun Quan

September 11, 2016

1 Introduction

In the first part of Project 1, we investigated into the mechanism of generating random sentences from datasets. Specifically, we started with preprocessing the dataset to keep only relevant information in our corpus, based on which we build up our unigram and bigram models to generate random sentences.

2 Unsmoothed n-grams

To get rid of all irrelevant information in the training dataset, we first processed the data following such steps:

- remove the sender's email addresses in each single file under the topic folder by deleting any segement of sentence starting with "From" and ending up with a space follwing an email address
- remove the first "Subject :" in each single file
- remove all linebreakers " >"

By doing so, we clean up our data so that irrelevant info such as line breakers in the text files will not take up tokens. Note that we did not remove the contents of subject because they might be relevant to the topic. Also, the email addresses other than sender's ones are kept to retain information integrity. The next thing to do is merge all files to obtain the corpus and tokenize the corpus. Specifically, the merged file is scanned for sentence ending indicators including "." "!" "?" or multiples of them, and at the beginning and ending of each sentence we add "<s>" and "</s>" respectively, which is stored as `corpus_sentence`. However, we did not consider "<s>" and "</s>" for tokenization, where results were stored in `corpus_word`.

Now that the corpus is built up, we are able to compute our unigram and bigram probablities. The probabilities are stored in two dictionary named `unigram_probabilities` and `bigram_probabilities` corresponding to unigram and bigram models respectively.

3 Random sentence generation

With our unigram and bigram language models ready, we are now able to develop a random sentence generator. The results are as expected: some are comprehensible and some are not. For example, IN the 'religion' topic corpus, a sentence " So you do wrong with gunfire . " was produced with bigram model and "Lafayette H well perhaps ____ Rosicrucians ." was generated from unigram model. While the bigram model is giving a more comprehensible sentence because it considers the preceeding word, the unigram model is less reliable by its nature. Also note that even though symbols such as "_" make the sentence even more confusing, we insisted on keeping all special symbols because they might be meaningful in certain context such as 'graphics' topic. Another noteworthy example is that in the 'atheism' model, we obtained a random sentence "." which consists of only one period and one space. This is not surprising since most periods "." are followed by a space, and in our case since we randomly selected "." as the first word, it is hence expected to give such a sentence. A similar sentence is generated in 'space' topic under unigram model : "moon." Incomplete sentence fragments can also be generated. For example, we get " On the Reagan administration . " in the 'space' topic. It is not surprising if we consider the construction of bigram model: the probability $P(\text{"."}|\text{"administration"})$ is found to be as high as 0.5 which explains why this sentence is very likely to happen.

We can have a better comparison between unigram and bigram models by looking at the results from seeding. For example in the 'atheism' topic, we set the beginning of sentences as 'I', and what we got from unigram model is 'I keith my strong] the the ?', compared to the one from bigram model ' I guarantee that was the theists be aware of the motto to defend , what that it must be a fallacy . '. Apparently the bigram result makes more sense because in the corpus 'I' is very likely to be followed by 'guarantee' and likewise 'guarantee' will be followed by 'that' with a high probability. However in unigram model, the preceeding word 'I' is not making a difference, so it will continue to generate succeeding words independently and that's why the unigram sentence is less readable.

4 Workflow

Throughout this project, work was distributed to each member fairly and voluntarily. For each section, before coding it up, we would have a thorough discussion about what to do and how to implement. Rui offered to do the preprocessing part. Upon completion of preprocessing, Gi Yoon offered to do the n-gram probabilities calculation and random sentence generator part. Shortly after, we together saw the experiments with random sentence generator working well, and Xiaoyun undertook to summarize all of our work in this report. We are glad that communication among members has been efficient and work has been done in a timely manner.

5 Smoothing and unknown words

Now let's consider testing our training dataset on some test files. The first question arose here is that what if we encounter some new words in the test file that are not in corpus? One way to solve this is by including unknown words in our corpus. By doing so, the probability of new words in test file will be nonzero. We learned two ways to realize this in class: one is to replace the first occurrence of each word with "<unk>", and the other is by replacing all words that only appear once in the corpus with "<unk>". We believe the latter will outperform for the reason that the gap between word types and corpus length is not that wide for all topics otherwise we would get a fairly large amount of pseudo-words which is not desired (see table below). For example, under 'Autos' topic, we have 9555 word types and 74288 tokens, and the first method will make $\frac{1}{8}$ of the corpus to be "<unk>", which will mislead the unigram and bigram probabilities.

	Autos	Atheism	Graphics	Medicine	Motorcycles	Religion	Space
Token	9555	10917	11583	12178	9537	11866	12427
Word Type	74288	120085	89140	93990	69167	117811	96024

Now that we have set up an open vocabulary, it is likely that this new corpus still has many zero bigram probabilities which can be problematic for training on test set. The strategy here will be smoothing for our language models. In fact, Good-Turing is implemented for both unigram and bigram models to get a smoothed count for each word using this formula:

$$c^* = (c+1)\frac{N_{c+1}}{N_c}$$

where c is the original count and $N_c = \sum_{\text{x: count(x)}=c} 1$