Language Modeling Assignment Report

Noah Sterling, Lucas Reisch

Preprocessing

We decided to code our project using Python 2.7.12. There were a couple of things we decided to do right away. For one we decided that the we did not want to include the From, Subject or any email addresses in the training model so we removed any references to them as best we could. We also decided we wanted to removed any weird email signature as best as we could since these usually consisted of symbols rather than actual words. The main rationale for removing these things was because we did not want them to appear in our random generated sentences. Removing most of the punctuation and the email markers from the training model just made out random sentences look much cleaner. We also decided that capitalization did not matter. We made everything lower case in order to make the calculation easier.

Random Sentence Generator

Before we could actually generate any sentences from the training model, we had to first code the unigram and bigram models. In order to do this we decided to loop through all of the text files in the data set and use a preexisting program to parse the text file into individual tokens. The program we used is called spaCy (**https://spacy.io/docs/#install-source-osx**). Once we had all of the tokens from the training set, we created two dictionaries, one for the unigram model and one for the bigram model. These dictionaries contained either the unigram or bigram and its associated number of token occurrences in the training set. When iterating through the tokens, we kept track of the total number of tokens as well as the number of occurrences for each unigram or bigram element.

After all of the occurrences were counted we used the n-gram equations from class in order to calculate the unigram and bigram probabilities, which is the probability that that word or word pair appears in a sentence based on the given training data. Once the probabilities were calculated, we used them in order to generate our sentences. We created a program which picks a word based on its probability for the unigram model and picks the successive word given a starting word for the bigram model based on the probability that that particular bigram shows up.

Random Sentences

Atheism Unigram

```
<s> set why not states is as cruel invited messiah the say so or </s>
<s> net upon of us me argument conquered this things it banking </s>
<s> religious kelley and believe to here ideals bronx to </s>
```

Atheism Bigram

```
<s> think i will also i've seen the help </s>
<s> then one can you get an islamic law </s>
<s> mean that there have in such inconsequential characteristics </s>
```

Auto Unigram

```
<s> all dealer fairly out mail keep </s>
<s> crime forget not doesn't on it heard </s>
<s> at ban is explorer mechanic </s>
```

Auto Bigram

```
<s> thanks to clear coated </s>
<s> if you're not reproduce the beach and stay with a patent on him a c + very
ugly station wagon has passed over the problem with gaia </s>
<s> 357 magnum revolver sitting in the inside </s>
```

Graphics Unigram

```
<s> with paste for harddisk ! a low-level client it's </s>
<s> is several 24 anti-aliasing unless of the there image post on unix z morph
users functions </s>
<s> trigger available client by problem to well which status this vesa d able
publish and sale anything anyone </s>
```

Graphics Bigram

```
<s> you are interested send a 24 bit has a dizzying number of the nearest
archive location graphics graphics </s>
<s> were saying help in jpeg it eventually degrade to see it until i
spend days </s>
<s> this to poke me on </s>
```

Medicine Unigram

```
<s> a vitamin all handedness that then the side thorough the
administration </s>
<s> re-entry if none reports it wants crew-cut set-up </s>
<s> discovered available dollars inhibitors </s>
```

Medicine Bigram

```
<s> i simply move the other licensed physician enough for cesarean rate in
investigations of these are usually have stabilized over 100 % of a
```

philosopher must a different so her gynocologist even look into what i can't she hasn't shown to come to eat pasta rice </s>
<s> upon the galvanocautery describes general procedure it doesn't know of some recreational </s>
<s> both wash hands he was the 19th century that supposedly enormous benefits </s>

Motorcycles Unigram

<s> meeting in & #007 was the francisco </s>
<s> circuitous worst back time ama toronto bikes in sitting just </s>
<s> the turbo no is </s>

Motorcycles Bigram

<s> both bikes with stupid act </s>
<s> the gas tank is sponsored by one rational to do </s>
<s> speedy mercer writes no way out of thinking about them out of questions but it was a cop doesn't like handcuffs </s>

Religion Unigram

<s> pride to denying centuries mention everything love is
blessed the use </s>
<s> rob long other not is bottles peoples interesting </s>
<s> there was speaking comments more the hurt they you there jesus a murray styles john peace </s>

Religion Bigram

<s> i found them </s>
<s> right and that calls it is the entire physical presence of the text you want to show more than a bit </s>
<s> i tell you an uncompromising standard axiom that has it </s>

Space Unigram

<s> upcoming the an atmosphere on 200mph was measures giant budget walking various the 602 a since questions </s>
<s> some 995 _ a mysteries 2016 despite release </s>
<s> thing you suprised catch other </s>

Space Bigram

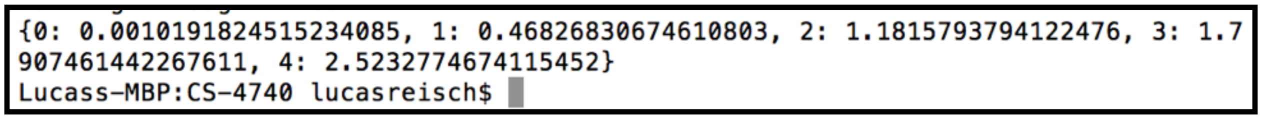
<s> usually induced partly because say nasa fact that i really
depressed </s>
<s> upon the way of studies in article </s>
<s> the class work and fictional contains formulae for apollo but it's not _the gods must model which philosopher would only a valid abort
maneuvers to go </s>

Random Sentence Analysis

Overall the generated sentences were pretty good. Obviously the bigram sentences are much better than the unigram sentences because the Language Model is more constricted. In the unigram model, the next word will just be something from the corpus, but in the bigram model, the next word has to be a valid bigram from the corpus in order for that pair to have a non-zero probability. For this reason, things like “19th Century”, “24 bit”, and “357 magnum revolver” all make sense and work small scale within the sentence. Even still however, the sentences remain largely ungrammatical and very rarely turn out meaningful. This is to be partially expected though because it is only a bigram model, it is unsmoothed, and the training set was a collection of emails, which, aside from tweets or texts, are one of the worst things to use if you want to create a grammatical meaningful sentence generator. One thing that the generator does well is it preserves the topic of the corpus. It is relatively easy to guess the topic of the training set based on the sentence s the program outputs.

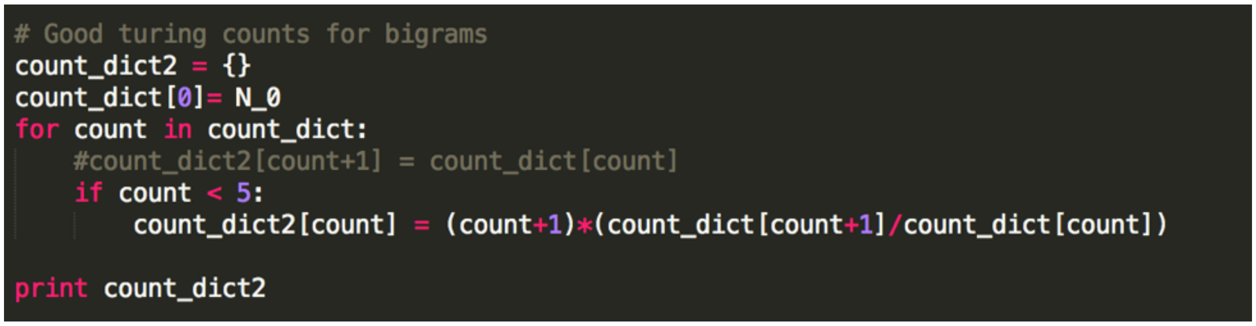
Smoothing and Unknown words

In order to accomplish Good-Turing Smoothing and unknown word tagging we began with altering our original corpus. We removed each word that only appeared once in the corpus and replaced it with the tag “<unk>”. After doing this we gathered the counts for both unigrams and bigrams in the corpus. Once we finished this we got the Good-Turing Smoothing counts for bigrams that appeared less than 5 times in the corpus. The c* counts for the space corpus are as follows:



```
{0: 0.0010191824515234085, 1: 0.46826830674610803, 2: 1.1815793794122476, 3: 1.7
907461442267611, 4: 2.5232774674115452}
Lucass-MBP:CS-4740 lucasreisch$ 
```

We calculated the c*s with the following piece of code



```
# Good turing counts for bigrams
count_dict2 = {}
count_dict[0]= N_0
for count in count_dict:
    #count_dict2[count+1] = count_dict[count]
    if count < 5:
        count_dict2[count] = (count+1)*(count_dict[count+1]/count_dict[count])

print count_dict2
```

count_dict2 is the dictionary to store the c* values in. count_dict is the dictionary of bigram counts for all occurrences of every bigram.

The role of the unknown words and smoothing in the following parts is to increase accuracy by decreasing the most infrequent uses of words. These words that are rarely used in a corpus make it difficult to get accurate probabilities.

Perplexity

In order to calculate perplexity we made a function calculate_perplexity. The function takes in the bigram probabilities, the tokens from our corpus, and the test_file that we wish to calculate the perplexity of. We start off by taking the test_file and removing all unknown words from it to keep our design consistent. We do this as we iterate through the tokens to get all the possible bigrams. Once we get the bigrams from our test file we begin to iterate through them to get the probabilities from our actual bigram probabilities for our language model. If the bigram doesn't appear in our model we assign it the minimum probability in order to make sure we aren't adding any zero probabilities to our perplexity calculation. As we iterate through these bigrams we take the -log of probability, sum them, the apply the formula:

$$PP = \Big(\prod_{i}^{N} \frac{1}{P(W_i | W_{i-1}, \ldots, W_{i-n+1})}\Big)^{-1/N} = \exp\big(1/N \cdot \sum_{i}^{N} \log(P(W_i \mid W_{i-1}, \ldots, W_{i-n+1}))\big)$$

Once we apply this formula we end up with our final perplexity value and output it.

Topic Classification

We approached the topic classification by taking the minimum value of all language models perplexities for a test file. This should give us an accurate portrayal of what topic the test_file is. We first gathered a csv of all the perplexities for each test_file for each language model. Then we ran through these to output a table of what each document should be classified as.

Context Aware Spell Checker

Since we already have the bigram probabilities and we know which words we have to check the spelling of, my partner and I decided that the best way to do this would be to search through test files in search of a word in the confusion set. Once one was found, we got the preceding word and ran a bigram probability check to see which word was more probable in the situation. If it was found that the confusion set pair was more likely in that particular environment then it was corrected and the next confusion set word was searched for.

Extension - Trigram model

For our extension we implemented a trigram model. All the code is in the file trigram_model.py. First we went through the processes to calculate for unigram and bigram models in order to get the necessary counts for the trigram model. Once we did this we removed the unknown words from the counts. Once we did that we implemented a random sentence generator similar to the bigram model. This outputted very good sentences from the corpus. We made it select the first and second word from the bigram model then use the trigram model to implement the rest of the random word decisions. The function that did this is make_trigram_sentence(tri_probs,bi_probs). An example of an output sentence is:

```
<s> they were not contemplating suicide it is known at the second more
serious one appeared along the fc lines of hebrews </s>
```

After this we began implementing the perplexity function. We did it in a very similar manner as the bigram perplexity function. Our perplexity dropped significantly from bigram to trigram which was expected. Our bigram perplexity for the file_0.txt from the motorcycles corpus was 16.0510798 and for the trigram perplexity it was 8.26490383243. This was what we hoped to have happen as the model becomes more complex the perplexity should drop.

Workflow

For the most part my partner and I worked together on the entire assignment. We met together a lot to go over the assignment and understand what we had and wanted to do. Since one of us is a better coder and the other has more linguistic experience, we divided up the assignment some this way. One person thinking about what we needed to do and why and the other person actually coding it. With this being said we both conferred about the algorithms we wanted to use and what kind of language model we wanted to build. We both contributed to different parts of this document.