Report on Language Model for Authorship

Yanshan Guo and Lucy Lu

February 25, 2016

1 Building Language Model

1.1 Data Cleaning Method

1.2 tokenize method

We first converted each author's entire text into a string without newline characters. Then we removed all punctuations and attached startline and endline characters to sentences. Every word is lower case for the convenience of comparison. The tokenize function, which is called clean_Data_And_Get_Dev_Training_Set will return a list of tokens including every word in the author's text and startline and endline characters.

1.2.1 Development Set

We select the first N number of sentences of each author's text to be their development set. We were worried about the possible bias caused by selecting the beginning of the text. However, tests with the development set being the beginning part of the text and tests with the development set being the middle part of the text did not yield significant different results that will eventually affect our choice of language models. Therefore, we decide to let each author's development set to be the beginning N sentences of each author's text.

Another thing to mention is that every sentence in our development set has length of at least five. This is due to the implementation of the N-gram count function that stores information about the occurrences of a given N-gram. This function calculates N-gram for N going from 1 to 5. Therefore, any input sentence should contain at least five words in it.

1.2.2 Training Set

The training set is the rest of the text that has not been included in the development set.

1.2.3 Unknown Words

Our initial attempt is to clean data by adding only the words in a dictionary. However, the result is not satisfactory because the dictionary is not big enough. Even if a stemming method is used before dictionary lookup, not all words can be recognized because the

stemmer does not work for all words. It is possible, due to the limitation of the tokenizing method, that some sentences may be incorrectly splitted: some abbreviations are not defined in the abbreviation list and making the program incorrectly treat the abbreviations "." as an end-of-sentence indicator. This error will not be a serious problem. The sentence length is set to be greater than or equal to 5 and we believe that even if the sentence is incomplete, a length of 5 or greater segment can provide adequate information.

1.3 N-gram Size Selection

N-grams from unigram to quintagram are tested.

1.4 Smoothing Method

This program provides two main smoothing methods. One method is Laplace smoothing and the other method is Good Turing smoothing in combination with linear interpolation.

1.4.1 Laplace

The Laplace smoothing method adds 1 to numerator and the size of the vocabulary set to the denominator. It is the same as the Laplace smoothing we talked about in the class.

1.4.2 Good Turing

For the Good Turing method, we use the following equation to calculate a word sequence's probability.
Let c be the count of a given N gram $w_1...w_n$ and c_{n-1} be the count of $w_1...w_{n-1}$, then

$$c^* = \frac{(c+1)\frac{N_{c+1}}{N_c} - c\frac{c(k+1)N_{k+1}}{N_1}}{1-\frac{(k+1)N_{k+1}}{N_1}}, \text{ when } 1 \leq c \leq k,$$

$c^* = \frac{N_1}{N^2}$, when $c = 0$,

$c^* = 1$, when $c > k$

Thus, $P(w_1...w_n) = c^* \frac{c}{c_{n-1}}$.

This Good Turing method is adapted from the textbook. We modify the case when $c = 0$. Even though, theoretically, setting c^* to be $\frac{N_1}{N^2}$ will cause some loss of probability. Empirically, we find that did not affect the result much and it has the benefit of ensuring a stable low probability of unknown words. In our program, we let $k = 5$.

1.4.3 Linear Interpolation

The technique we used in combination with Good Turing is linear interpolation, which calculates the probability of a given N-gram by combining different order N-grams by linearly interpolating all the models. To get the optimal lambda values for our N-gram language model, we train lambda values on the development set as the held-out corpus. For a given N-gram model, we repeatedly sample $\{\lambda_1, \lambda_2, ..., \lambda_n\}$, where $\sum_{i=1}^{i=n} \lambda_i = 1$, uniformly from zero to one and calculate the probability of the held-out corpus. For a given N-gram, the set of lambda values that yield the highest probability for the held-out corpus will be the lambda parameters we use later in the test option.

2 Result

2.1 Language Model Perplexity

The following chart shows the perplexity for each N-grams. For Dickens, Christie, Doyle, Thoureau and and Whitman, unigram yields the smallest perplexity. Therefore, for these authors, unigram is the best language model. For Austen and Wilde, bigram yields the smallest perplexity, thus for these authors, bigram is considered the best language model.

Perplexity

	1-gram	2-gram	3-gram	4-gram	5-gram
Dickens	**812**	2280	18434	92407	110537
Christie	**3383**	29857	109157	67842	39900
Doyle	**1320**	4843	16902	63226	44179
Whitman	**1217**	2647	inf	inf	inf
Wilde	1722	**1663**	16178	27124	40999
Austen	299	**97**	2488	10813	37341
Thoureau	**779**	871	inf	inf	inf

2.2 Linear Interpolation λ Value For Bigram

Considering that unigram and bigram perform the best from the perplexity chart above, we only compute λ for bigram (unigram does not need λ). The following chart shows the best λ_1 and λ_2 values for a bigram language model using linear interpolation. The result shows that an almost even distribution between unigram and bigram yield the best probability. The two λ values are obtained by selecting the two values that yield the highest probability of the held-out corpus from 20 number of potential (λ_1, λ_2) values, repeatedly sampled by a random float number generator.

Linear Interpolation Lambda Value For Bigram

	Christie	Dickens	Wilde	Austen	Thoureau	Whitman	Doyle
λ1	0.312	0.480	0.446	0.324	0.480	0.561	0.408
λ2	0.688	0.520	0.554	0.676	0.520	0.439	0.591

2.3 Accuracy

Looking at the accuracy chart, we find that the accuracy for predicting Christie's work decreases as N becomes small. Bigram of Whitman yields the highest probability. Quadgram yields the best result for Wilde. Bigram again, predicts Dickens the best. Same for Austen, Doyle, and Thoureau. In general, we see that bigram is better at predicting most of authors' works.

Accuracy

	Christie	Whitman	Wilde	Dickens	Austen	Thoureau	Doyle
Unigram	66%	87%	64%	75%	64%	67%	55%
Bigram	46%	92%	58%	78%	80%	76%	64%
Trigram	40%	68%	72%	44%	56%	76%	60%
Quadrigram	30%	20%	85%	0%	40%	30%	20%

2.4 Running Time

The expected running time is between 10-15 min without linear interpolation and between 20-25 min with linear interpolation. The linear interpolation slows down the program significantly due to the complexity of the calculation.

3 Conclusion

From the data we collected, we find that we can get best predication by using bigram language models with λ parameters we get in the chart in section 2.2. Even though unigram yields the smallest perplexity for some authors, when combined with linear interpolation, bigram shows better perforance than unigram.

Due to the complexity of the program, we do not have enough time to calculate λ values for all N-grams. This is a potential future improvement of the algorithm since there might be more suitable N-grams with $N > 2$ that predict result better with linear interpolation.