Predicting Next Word Using Katz Back-Off

Part 3 - Understanding the Katz Back-Off Model

Michael Szczepaniak

September 3, 2016 (last updated)

Introduction

Before one can implement any kind of mathematical model, machine learning or otherwise, they need a solid understanding of how it works. This article describes how the Katz Back-Off (KBO) language model works by way of simple examples keeping the math to a minimum. The ideas presented here are implemented in the git project **PredictNextKBO** as a web app deployed at:

https://michael-szczepaniak.shinyapps.io/predictnextkbo/.

Familiarity with basic concepts such as conditional probabilities, the chain rule of probabilities, Markov assumptions, probability density functions, et. al. are helpful, but not required. I will describe these concepts in an easy to understand manner as they arise throughout this article and/or provide references for those who want to dive deeper into a particular concept.

What is the Katz Back-Off (KBO) Model?

Wikipedia provides the following definition:

> Katz back-off is a generative n-gram language model that estimates the conditional probability of a word given its history in the n-gram. It accomplishes this estimation by "backing-off" to models with smaller histories under certain conditions. By doing so, the model with the most reliable information about a given history is used to provide the better results. [1].

Language Models

The term *generative n-gram language model* in the above definition is a loaded term which deserves explaination. Let's start with the term *language model* (LM). In the context of Natural Language Processing (NLP), a language model, which is also referred to as a **grammar**, is a mathematical model which is constructed for the purpose of assigning a probability to either a series of words $w_{i-n}, w_{i-n+1}, ...w_i$ which we can denote as $P(w_{i-n}, w_{i-n+1}, ...w_i)$ or the probability of the last word given the previous words as denoted by $P(w_i|w_{i-n}, w_{i-n+1}, ...w_{i-1})$. This later definition is what we'll be estimating in order to predict the next word of a phrase.

Why would we want to assign a probability to a word or series of words? The simple answer is that we want to make word predictions based on the highest probability of what we might actually observe.

Generative N-gram Language Model

A *generative* grammar or LM, refers to the quality of languages related to the rules governing its structure or syntax [2]. Because of these rules, not all combinations of words form valid sentences and therefore combinations that don't conform to these rules should be assigned lower probabilities of occurring. In such a model, we are aware of the rules governing the structure, but this structure is not explicitly accounted for, but rather inferred from the model. For example, the model should assign a lower probability to the sentence *I red a book* than to the sentence *I read a book* based on joint probability distributions of various word groupings known as **n-grams**.

The term **n-gram** refers to the number of consecutive words utilized in a LM. For example, say we see the terms "*I want to eat Chinese*" 7 times and "*I want to eat Italian*" 3 times in a body of text (aka *corpus*), we might want to match the highest probability 5-gram to complete the 4-gram "*I want to eat*" which in this little example would be *Chinese*.

Dan Jurafky describes the motivations and basic ideas behind probablistic n-gram models in this video [3] and how to estimate these probabilities in this video [4].

Assigning Probabilities Using the Maximum Likelihood Estimate

The most intuitive way to assign probabilities to a series of events is to count up the number of occurances of each event type (like seeing a particular word or combination of words) and divide by the total number of occurances. Assigning probilities in this manner is referred to as the **Maximum Likelihood Estimate** (MLE) because such estimates will be higher than the actual or true probability because unobserved events (e.g. combinations of words) have not been accounted for. These unobserved events take up some of the probability mass in the true probability density function which is not accounted for in the observed distribution.

Accounting for Probability Mass of Unseen N-grams: Discounting

In discounting, some of the probability mass is taken from observed n-grams and distributed to unobserved n-grams in order to allow us to estimate probabilities of these unseen n-grams. In the KBO trigram LM, as will be discussed in more detail later in the article, we select an absolute discount which artificially lowers the counts of observed trigrams and distributes this "stolen" probabilty mass to unobserved trigrams.

The basic concept of discounting is most easily understood by way of example as shown in the two lectures given by Michael Collins referenced in **Appendix 1** at the end of this work.

Discounting is a key concept integral to KBO and other advanced LMs like Good-Turing [5] and Kneser-Ney [6] as described in the lectures by Dan Jurafasky provided in links [5] and [6].

Using N-gram Tables for Prediction - A Little Math

The theory behind the use of n-gram probability estimates starts with the Chain rule for conditional probabilities [7] applied to compute joint probabilities which states:

1. $P(w_1, w_2, w_3, ...w_n) = \prod_{i=1}^{n} P(w_i|w_1, w_2...w_{i-1})$

Jurafsky [3] applies equation 1. in the following example to illustrate:

2. $P(its\ water\ is\ so\ transparent) =$
   $P(its) \times P(water \mid its) \times P(is \mid its\ water) \times P(so \mid its\ water\ is) \times P(transparent \mid its\ water\ is\ so)$

Simplifying N-gram Probability Estimates: The Markov Assumption

The example in equation 2. is the full expansion of the Chain Rule using equation 1. This can be simplified by assuming that the probability of the left side of equation 1. can be approximated by less than the prior $(n-1)$ terms which can be expressed more formally as:

3. $P(w_1, w_2, w_3, ...w_n) \approx \prod_{i=1}^{n} P(w_i|w_1, w_{i-k}...w_{i-1})$

Making the above approximation is referred to as applying the Markov Assumption. From the previous example, this is like saying:

4. $P(the \mid its\ water\ is\ so\ transparent\ that) \approx P(the \mid that)$ or

5. $P(the \mid its\ water\ is\ so\ transparent\ that) \approx P(the \mid transparent\ that)$

In equation 4., the probability of the bigram *the* given *that* is used to estimate the probability of *the* given *its water is so transparent that*. In equation 5., the probability of the trigram *the* given *transparent that* is used to make the same estimate.

Estimating Probabilities

Observed N-grams

Using the bigram MLE to estimate the probability of the last word given the previous word can be written as

6. $P_{ML}(w_i \mid w_{i-1}) = \frac{count(w_{i-1}, w_i)}{count(w_i)} = \frac{c(w_{i-1}, w_i)}{c(w_i)}$

where w_i is the last word (the one we are trying to predict), w_{i-1} is the word just prior to w_i and *count* has been abbreviated to *c*. The trigram MLE can be written as:

7. $P_{ML}(w_i \mid w_{i-1},\ w_{i-2}) = \frac{c(w_{i-2}, w_{i-1}, w_i)}{c(w_{i-1},\ w_i)}$

where w_{i-2} is the word prior to w_{i-1}.

As long as we have seen one or more bigrams or trigrams, we can use 6. or 7. to estimate the probability of the last word w_i given the previous one (w_{i-1}) or two $(w_{i-2},\ w_{i-1})$. If we want to estimate probabilities using trigrams, but no trigrams are observed, we could "back-off" and estimate the probability using the bigram. If no bigrams exist, we could estimate using the unigram probability. The *Stupid Back-off* (SBO) model [8] essentially does this without consideration of unseen n-grams, but has been reported to perform quite well with large scale n-gram tables. Further details regarding the SBO can be found in the paper written by the creators of this alogrithm. [8]

Accounting for Unobserved N-grams

As mentioned earlier, the KBO algorithm estimates probabilities of unseen n-grams by redistributing some of the probability mass from observed trigrams to those that are unobserved through discounting. So the natural first step in this process is to determine the amount of probability mass that gets taken from the observed n-grams. The second step will be to determine how that taken probability mass is redistributed.

The total probability mass of the observer bigrams and trigrams can be written as:

8. $\textit{bigram probability mass} = \sum_{w \in \mathcal{A}(w_{i-1})} \frac{c(w_{i-1}, w)}{c(w_{i-1})}$

9. $\textit{trigram probability mass} = \sum_{w \in \mathcal{A}(w_{i-2},\ w_{i-1})} \frac{c(w_{i-2}, w_{i-1}, w)}{c(w_{i-2}, w_{i-1})}$

where the terms $w \in \mathcal{A}(w_{i-1})$ and $w \in \mathcal{A}(w_{i-2},\ w_{i-1})$ refer to all the words w in the set $\mathcal{A}$ which are all words that terminate an observed bigram starting with (w_{i-1}) or trigram starting with $(w_{i-2},\ w_{i-1})$ respectively. If we define γ_2 to be the amount of discount taken from observed bigram counts, γ_3 the amount of discount taken from observed trigram counts, and c^* to be the new discounted counts for observed bigrams and trigrams, then the backed off probability estimates would be written as:

10. $q_{BO}(w_i \mid w_{i-1}) = \frac{c^*(w_{i-1}, w)}{c(w_{i-1})}$ for observed bigrams, where

11. $c^*(w_{i-1},\, w) = c(w_{i-1},\, w) - \gamma_2$ and

12. $q_{BO}(w_i \mid w_{i-2},\, w_{i-1}) = \frac{c^*(w_{i-2},\, w_{i-1},\, w)}{c(w_{i-2},\, w_{i-1})}$ for observed trigrams, where

13. $c^*(w_{i-2},\, w_{i-1},\, w) = c(w_{i-2},\, w_{i-1},\, w) - \gamma_3$

From here it follows that the amount of discounted probability mass taken from observed bigrams and trigrams can then be defined as:

14. $\alpha(w_{i-1}) = 1 - \sum_{w \,\in\, \mathcal{A}(w_{i-1})} \frac{c^*(w_{i-1},\, w)}{c(w_{i-1})}$ from observed bigrams, and

15. $\alpha(w_{i-2},\, w_{i-1}) = 1 - \sum_{w \,\in\, \mathcal{A}(w_{i-2},\, w_{i-1})} \frac{c^*(w_{i-2},\, w_{i-1},\, w)}{c(w_{i-2},\, w_{i-1})}$ from observed trigrams.

At this point, we've finished the first step by defining the amount of probability mass taken from the observed bigrams and trigrams as a function of some discounted amount γ_n. Now we need to know how to assign this discounted probability mass to unseen bigrams and trigrams. Katz proposed that this be done proportionally to the backed-off (n-1)-gram probability parameters. In other words, an unobserved trigram probability would be estimated in proportion to all the possible bigram tails and unobserved bigram probabilities would be estimated in proportion to all the possible unigram tails.

More formally, if the estimate for this backed-off unigram probability is $q_{ML}(w_i)$ and the backed off bigram probability is $q_{BO}(w_i \mid w_{i-1})$, then the amount of discounted probability mass $\alpha(w_{i-1})$ assigned to unknown bigrams $q_{BO}(w_i \mid w_{i-1})$ and $\alpha(w_{i-2},\, w_{i-1})$ assigned to unknown trigrams $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$ would be:

16. $q_{BO}(w_i \mid w_{i-1}) = \alpha(w_{i-1}) \frac{q_{ML}(w_i)}{\sum_{w \,\in\, \mathcal{B}(w_{i-1})} q_{ML}(w)} = \alpha(w_{i-1}) \frac{c(w_i)}{\sum_{w \,\in\, \mathcal{B}(w_{i-1})} c(w)}$

17. $q_{BO}(w_i \mid w_{i-2},\, w_{i-1}) = \alpha(w_{i-2},\, w_{i-1}) \frac{q_{BO}(w_i \mid w_{i-1})}{\sum_{w \,\in\, \mathcal{B}(w_{i-2},\, w_{i-1})} q_{BO}(w \mid w_{i-1})}$

respectively, where the terms $w \in \mathcal{B}(w_{i-1})$ and $w \in \mathcal{B}(w_{i-2},\, w_{i-1})$ refer to all the words w in the set $\mathcal{B}$ which are all words that terminate an **unobserved** bigram starting with (w_{i-1}) or **unobserved** trigram starting with $(w_{i-2},\, w_{i-1})$ respectively. Notice that the subscripts in the q terms on the middle terms of equation 16. are **ML**. This is because at this point, we are using the Maximum Likelihood estimate for the unigrams which allows for the simplification to the counts shown in the rightmost term.

Steps in Applying the KBO Trigram Alogrithm

1) Organize the data needed to use the equations defined above:
   i. Create tables of unigram, bigram, and trigram counts.
   ii. Select the values for discounts at the bigram and trigram levels: γ_2 and γ_3.
2) Select a bigram that precedes the word you want to predict: (w_{i-2}, w_{i-1}). This will be referred to as the **bigram prefix** in the code and remainder of this document.
3) Calculate the probabilities for words that complete **observed** trigrams from equation 12.: $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$
4) Calculate the probabilities for words that complete **unobserved** trigrams from equation 17.: $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$.
5) Select w_i with the highest $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$ as the prediction.

Calculating Probabilities for Unobserved N-grams

The steps described above are straightforward, but step 4. deserves futher explaination. Each q_{BO} term in equation 17. needs to be calculated using either equation 10. or equation 16. depending on whether the bigram (w_{i-1}, w_i) is observed or unobserved. If the bigram is observed, equation 10. should be used. If it is unobserved, equation 16. should be used.

Example of Applying the Algorithm: The Little Corpus That Could

As noted earlier, a corpus is a body of text from which we build and test LMs. To illustrate how the mathematical formulation of the KBO Trigram model works, it's helpful to look at a simple corpus that is small enough to easily keep track of the n-gram counts, but large enough to illustrate the impact of unobserved n-grams on the calculations. The following sample corpus was extended from an example provided by Michael Collins in week 1 of his class referenced in **Appendix 1**. This corpus is listed in **Appendix 2** at the end of this document.

```
ltcorpus <- readLines("little_test_corpus1.txt")
ltcorpus
```

```
## [1] "SOS buy the book EOS"    "SOS buy the book EOS"
## [3] "SOS buy the book EOS"    "SOS buy the book EOS"
## [5] "SOS sell the book EOS"   "SOS buy the house EOS"
## [7] "SOS buy the house EOS"   "SOS paint the house EOS"
```

In this corpus, SOS and EOS are tokens used to denote *start of sentence* and *end-of-sentence*.

Step 1. i. Unigram, Bigram and Trigram counts

This work used the **quanteda** package written by Ken Benoit and Paul Nulty to construct the n-gram tables. My experience with this package is that it performs much faster than **tm** and **RWeka** for these types of tasks, but of course, your mileage may vary. A nice little getting started guide to quanteda can be found here [9].

The following code was used to create the unigram, bigram, and trigrams tables shown below the code.

```
## Returns a named vector of n-grams and their associated frequencies
## extracted from the character vector dat.
##
## ng - Defines the type of n-gram to be extracted: unigram if ng=1,
##      bigram if ng=2, trigram if n=3, etc.
## dat - Character vector from which we want to get n-gram counts.
## ignores - Character vector of words (features) to ignore from frequency table
## sort.by.ngram - sorts the return vector by the names
## sort.by.freq - sorts the return vector by frequency/count
getNgramFreqs <- function(ng, dat, ignores=NULL,
                          sort.by.ngram=TRUE, sort.by.freq=FALSE) {
    # http://stackoverflow.com/questions/36629329/
    # how-do-i-keep-intra-word-periods-in-unigrams-r-quanteda
    if(is.null(ignores)) {
        dat.dfm <- dfm(dat, ngrams=ng, toLower = FALSE, removePunct = FALSE,
                       what = "fasterword", verbose = FALSE)
    } else {
        dat.dfm <- dfm(dat, ngrams=ng, toLower = FALSE, ignoredFeatures=ignores,
```

```
                        removePunct = FALSE, what = "fasterword", verbose = FALSE)
    }
    rm(dat)
    # quanteda docfreq will get the document frequency of terms in the dfm
    ngram.freq <- docfreq(dat.dfm)
    if(sort.by.freq) { ngram.freq <- sort(ngram.freq, decreasing=TRUE) }
    if(sort.by.ngram) { ngram.freq <- ngram.freq[sort(names(ngram.freq))] }
    rm(dat.dfm)

    return(ngram.freq)
}

## Returns a 2 column data.table. The first column: ngram, contains all the
## unigrams, bigrams, or trigrams in the corpus depending on whether
## ng = 1, 2, or 3 respectively. The second column: freq, contains the
## frequency or count of the ngram found in linesCorpus.
##
## ng - Defines the type of n-gram to be extracted: unigram if ng=1,
##      bigram if ng=2, trigram if n=3, etc.
## linesCorpus - character vector: each element is a line from a corpus file
## prefixFilter - character vector: If not NULL, tells the function to return
##                only rows where the ngram column starts with prefixFilter.
##                If NULL, returns all the ngram and count rows.
getNgramTables <- function(ng, linesCorpus, prefixFilter=NULL) {
    ngrams <- getNgramFreqs(ng, linesCorpus)
    ngrams_dt <- data.table(ngram=names(ngrams), freq=ngrams)
    if(length(grep('^SOS', ngrams_dt$ngram)) > 0) {
        ngrams_dt <- ngrams_dt[-grep('^SOS', ngrams_dt$ngram),]
    }
    if(!is.null(prefixFilter)) {
        regex <- sprintf('%s%s', '^', prefixFilter)
        ngrams_dt <- ngrams_dt[grep(regex, ngrams_dt$ngram),]
    }

    return(ngrams_dt)
}

unigs <- getNgramTables(1, ltcorpus)
bigrs <- getNgramTables(2, ltcorpus)
trigs <- getNgramTables(3, ltcorpus)
unigs; bigrs; trigs
```

```
##    ngram freq
## 1:  book    5
## 2:   buy    6
## 3:   EOS    8
## 4: house    3
## 5: paint    1
## 6:  sell    1
## 7:   the    8
```

```
##        ngram freq
## 1:  book_EOS    5
```

```
## 2:   buy_the    6
## 3: house_EOS    3
## 4: paint_the    1
## 5:  sell_the    1
## 6:  the_book    5
## 7: the_house    3
```

```
##              ngram freq
## 1:    buy_the_book    4
## 2:   buy_the_house    2
## 3: paint_the_house    1
## 4:   sell_the_book    1
## 5:    the_book_EOS    5
## 6:   the_house_EOS    3
```

Step 1. ii. Selecting bigram and trigram discounts

For this example, we'll use $\gamma_2 = \gamma_3 = 0.5$ for the purpose of illustration. In practice, these values would be obtained by cross-validation. A great treatment of cross-validation can be found in Chapter 5 of this (free) book [10] which are discussed by the authors in these three videos: [11], [12], and [13].

Step 2. Select Bigram Prefix of Word to be Predicted

For this example, we'll select the bigram: `sell the`

Step 3. Calculate Probabilities of Words Completing Observed Trigrams

The code below finds the observed trigrams starting with the selected bigram prefix and calculates their probabilities. In our simple example, we can look at the table of trigrams above and see that there is only one trigram that starts with `sell the` which is `sell the book`. Applying equations 12. and 13, we get $q_{BO}(book \mid sell,\ the) = (1 - 0.5)/1 = 0.5$ which is also the result provided from the code below.

```
gamma2 <- 0.5  # bigram discount
gamma3 <- 0.5  # trigram discount
bigPre <- 'sell_the'

## Returns a two column data.frame of observed trigrams that start with the
## bigram prefix (bigPre) in the first column named ngram and
## frequencies/counts in the second column named freq. If no observed trigrams
## that start with bigPre exist, an empty data.frame is returned.
##
## bigPre -  single-element char array of the form w2_w1 which are the first
##           two words of the trigram we are predicting the tail word of
## trigrams - 2 column data.frame or data.table. The first column: ngram,
##            contains all the trigrams in the corpus. The second column:
##            freq, contains the frequency/count of each trigram.
getObsTrigs <- function(bigPre, trigrams) {
    trigs.winA <- data.frame(ngrams=vector(mode = 'character', length = 0),
                             freq=vector(mode = 'integer', length = 0))
    regex <- sprintf("%s%s%s", "^", bigPre, "_")
    trigram_indices <- grep(regex, trigrams$ngram)
    if(length(trigram_indices) > 0) {
        trigs.winA <- trigrams[trigram_indices, ]
```

```
    }

    return(trigs.winA)
}

## Returns a two column data.frame of observed trigrams that start with bigram
## prefix bigPre in the first column named ngram and the probabilities
## q_bo(w_i | w_i-2, w_i-1) in the second column named prob calculated from
## eqn 12. If no observed trigrams starting with bigPre exist, NULL is returned.
##
## obsTrigs - 2 column data.frame or data.table. The first column: ngram,
##            contains all the observed trigrams that start with the bigram
##            prefix bigPre which we are attempting to the predict the next
##            word of in a give phrase. The second column: freq, contains the
##            frequency/count of each trigram.
## bigrs - 2 column data.frame or data.table. The first column: ngram,
##         contains all the bigrams in the corpus. The second column:
##         freq, contains the frequency/count of each bigram.
## bigPre -  single-element char array of the form w2_w1 which are first two
##           words of the trigram we are predicting the tail word of
## triDisc - amount to discount observed trigrams
getObsTriProbs <- function(obsTrigs, bigrs, bigPre, triDisc=0.5) {
    if(nrow(obsTrigs) < 1) return(NULL)
    obsCount <- filter(bigrs, ngram==bigPre)$freq[1]
    obsTrigProbs <- mutate(obsTrigs, freq=((freq - triDisc) / obsCount))
    colnames(obsTrigProbs) <- c("ngram", "prob")

    return(obsTrigProbs)
}

obs_trigs <- getObsTrigs(bigPre, trigs)  # get trigrams and counts
# convert counts to probabilities
qbo_obs_trigrams <- getObsTriProbs(obs_trigs, bigrs, bigPre, gamma3)
qbo_obs_trigrams
```

```
##           ngram prob
## 1 sell_the_book  0.5
```

Step 4. Calculate Probabilities of Words Completing Unobserved Trigrams

This is the the most complex step as it involves backing off to the bigram level. Here is a breakdown of the sub-steps for these calculations:

i. Find all the words that complete unobserved trigrams. These are the words in the set $w \in \mathcal{B}(w_{i-2}, w_{i-1})$ described earlier.
ii. Calculate $\alpha(w_{i-1})$ from equation 14.
iii. Calculate q_{BO} for each bigram in the denominator of equation 17. using equation 10. if the bigram is observed or equation 16. if it is unobserved.
iv. Calculate $\alpha(w_{i-2},\, w_{i-1})$ from equation 15.
v. Calculate $q_{BO}(w_i \,|\, w_{i-2},\, w_{i-1})$ for each w_i from equation 17.

Step 4. i. Find unobserved trigram tail words:

```
## Returns a character vector which are the tail words of unobserved trigrams
## that start with the first two words of obsTrigs (aka the bigram prefix).
## These are the words w in the set B(w_i-2, w_i-1) as defined in the section
## describing the details of equation 17.
##
## obsTrigs - character vector of observed trigrams delimited by _ of the form:
##            w3_w2_w1 where w3_w2 is the bigram prefix
## unigs - 2 column data.frame of all the unigrams in the corpus:
##         ngram = unigram
##         freq = frequency/count of each unigram
getUnobsTrigTails <- function(obsTrigs, unigs) {
    obs_trig_tails <- str_split_fixed(obsTrigs, "_", 3)[, 3]
    unobs_trig_tails <- unigs[!(unigs$ngram %in% obs_trig_tails), ]$ngram
    return(unobs_trig_tails)
}

unobs_trig_tails <- getUnobsTrigTails(obs_trigs$ngram, unigs)
unobs_trig_tails
```

```
## [1] "buy"   "EOS"   "house" "paint" "sell"  "the"
```

Step 4. ii. Calculate discounted probability mass at the bigram level $\alpha(w_{i-1})$:

The code below implements equation 14. to calculate $\alpha(w_{i-1})$

```
## Returns the total probability mass discounted from all observed bigrams
## calculated from equation 14.  This is the amount of probability mass which
## is redistributed to UNOBSERVED bigrams. If no bigrams starting with
## unigram$ngram[1] exist, 0 is returned.
##
## unigram - single row, 2 column frequency table. The first column: ngram,
##           contains the w_i-1 unigram (2nd word of the bigram prefix). The
##           second column: freq, contains the frequency/count of this unigram.
## bigrams - 2 column data.frame or data.table. The first column: ngram,
##           contains all the bigrams in the corpus. The second column:
##           freq, contains the frequency or count of each bigram.
## bigDisc - amount to discount observed bigrams
getAlphaBigram <- function(unigram, bigrams, bigDisc=0.5) {
    # get all bigrams that start with unigram
    regex <- sprintf("%s%s%s", "^", unigram$ngram[1], "_")
    bigsThatStartWithUnig <- bigrams[grep(regex, bigrams$ngram),]
    if(nrow(bigsThatStartWithUnig) < 1) return(0)
    alphaBi <- 1 - (sum(bigsThatStartWithUnig$freq - bigDisc) / unigram$freq)

    return(alphaBi)
}

unig <- str_split(bigPre, "_")[[1]][2]
unig <- unigs[unigs$ngram == unig,]
alpha_big <- getAlphaBigram(unig, bigrs, gamma2)
alpha_big
```

```
## [1] 0.125
```

Step 4. iii. Calculate backed off probabilities q_{BO} for bigrams

The code below implements equation 10. to calculate $q_{BO}(w_i \mid w_{i-1})$ for observed bigrams and equation 16. for unobserved bigrams:

```
## Returns a character vector of backed off bigrams of the form w2_w1. These
## are all the (w_i-1, w) bigrams where w_i-1 is the tail word of the bigram
## prefix bigPre and w are the tail words of unobserved bigrams that start with
## w_i-1.
##
## bigPre - single-element char array of the form w2_w1 which are first two
##          words of the trigram we are predicting the tail word of
## unobsTrigTails - character vector that are tail words of unobserved trigrams
getBoBigrams <- function(bigPre, unobsTrigTails) {
    w_i_minus1 <- str_split(bigPre, "_")[[1]][2]
    boBigrams <- paste(w_i_minus1, unobsTrigTails, sep = "_")
    return(boBigrams)
}

## Returns a two column data.frame of backed-off bigrams in the first column
## named ngram and their frequency/counts in the second column named freq.
##
## bigPre -  single-element char array of the form w2_w1 which are first two
##           words of the trigram we are predicting the tail word of
## unobsTrigTails - character vector that are tail words of unobserved trigrams
## bigrs - 2 column data.frame or data.table. The first column: ngram,
##         contains all the bigrams in the corpus. The second column:
##         freq, contains the frequency/count of each bigram.
getObsBoBigrams <- function(bigPre, unobsTrigTails, bigrs) {
    boBigrams <- getBoBigrams(bigPre, unobsTrigTails)
    obs_bo_bigrams <- bigrs[bigrs$ngram %in% boBigrams, ]
    return(obs_bo_bigrams)
}

## Returns a character vector of backed-off bigrams which are unobserved.
##
## bigPre -  single-element char array of the form w2_w1 which are first two
##           words of the trigram we are predicting the tail word of
## unobsTrigTails - character vector that are tail words of unobserved trigrams
## obsBoBigram - data.frame which contains the observed bigrams in a column
##               named ngram
getUnobsBoBigrams <- function(bigPre, unobsTrigTails, obsBoBigram) {
    boBigrams <- getBoBigrams(bigPre, unobsTrigTails)
    unobs_bigs <- boBigrams[!(boBigrams %in% obsBoBigram$ngram)]
    return(unobs_bigs)
}

## Returns a dataframe of 2 columns: ngram and probs.  Values in the ngram
## column are bigrams of the form: w2_w1 which are observed as the last
## two words in unobserved trigrams.  The values in the prob column are
## q_bo(w1 | w2) calculated from from equation 10.
##
## obsBoBigrams - a dataframe with 2 columns: ngram and freq. The ngram column
##                contains bigrams of the form w1_w2 which are observed bigrams
##                that are the last 2 words of unobserved trigrams (i.e. "backed
```

```
##                off" bigrams). The freq column contains integers that are
##                the counts of these observed bigrams in the corpus.
## unigs - 2 column data.frame of all the unigrams in the corpus:
##         ngram = unigram
##         freq = frequency/count of each unigram
## bigDisc - amount to discount observed bigrams
getObsBigProbs <- function(obsBoBigrams, unigs, bigDisc=0.5) {
    first_words <- str_split_fixed(obsBoBigrams$ngram, "_", 2)[, 1]
    first_word_freqs <- unigs[unigs$ngram %in% first_words, ]
    obsBigProbs <- (obsBoBigrams$freq - bigDisc) / first_word_freqs$freq
    obsBigProbs <- data.frame(ngram=obsBoBigrams$ngram, prob=obsBigProbs)

    return(obsBigProbs)
}

## Returns a dataframe of 2 columns: ngram and prob.  Values in the ngram
## column are unobserved bigrams of the form: w2_w1.  The values in the prob
## column are the backed off probability estimates q_bo(w1 | w2) calculated
## from from equation 16.
##
## unobsBoBigrams - character vector of unobserved backed off bigrams
## unigs - 2 column data.frame of all the unigrams in the corpus:
##         ngram = unigram
##         freq = frequency/count of each unigram
## alphaBig - total discounted probability mass at the bigram level
getQboUnobsBigrams <- function(unobsBoBigrams, unigs, alphaBig) {
    # get the unobserved bigram tails
    qboUnobsBigs <- str_split_fixed(unobsBoBigrams, "_", 2)[, 2]
    w_in_Aw_iminus1 <- unigs[!(unigs$ngram %in% qboUnobsBigs), ]
    # convert to data.frame with counts
    qboUnobsBigs <- unigs[unigs$ngram %in% qboUnobsBigs, ]
    denom <- sum(qboUnobsBigs$freq)
    # converts counts to probabilities
    qboUnobsBigs <- data.frame(ngram=unobsBoBigrams,
                               prob=(alphaBig * qboUnobsBigs$freq / denom))

    return(qboUnobsBigs)
}

bo_bigrams <- getBoBigrams(bigPre, unobs_trig_tails)  # get backed off bigrams
# separate bigrams which use eqn 10 and those that use 16
obs_bo_bigrams <- getObsBoBigrams(bigPre, unobs_trig_tails, bigrs)
unobs_bo_bigrams <- getUnobsBoBigrams(bigPre, unobs_trig_tails, obs_bo_bigrams)
# unobs_bo_bigrams = c("the_buy", "the_EOS", "the_paint", "the_sell", "the_the")
# calc obs'd bigram prob's from eqn 10
qbo_obs_bigrams <- getObsBigProbs(obs_bo_bigrams, unigs, gamma2) #ngram     probs
# calc alpha_big & unobs'd bigram prob's from eqn 16             #the_house 0.3125
unig <- str_split(bigPre, "_")[[1]][2]
unig <- unigs[unigs$ngram == unig,]
# distrib discounted bigram prob mass to unobs bigrams in prop to unigram ML
qbo_unobs_bigrams <- getQboUnobsBigrams(unobs_bo_bigrams, unigs, alpha_big)
qbo_bigrams <- rbind(qbo_obs_bigrams, qbo_unobs_bigrams)
qbo_bigrams
```

```
##       ngram        prob
## 1 the_house 0.312500000
## 2   the_buy 0.031250000
## 3   the_EOS 0.041666667
## 4 the_paint 0.005208333
## 5  the_sell 0.005208333
## 6   the_the 0.041666667
```

Checking the Bigram Calculations

Before doing the final calculations for the unobserved trigrams, let's do a simple check on our calculations at the bigram level. In the previous table, all the bigrams except `the_house` are unobserved which means that if we sum all the unobserved bigram probabilities, we should get the total bigram discount which is $\alpha(w_{i-1})$. As we see below, this looks like it checks out.

```
unobs <- qbo_bigrams[-1,]
sum(unobs$prob)
```

```
## [1] 0.125
```

Step 4. iv. Calculate discounted probability mass at the trigram level $\alpha(w_{i-2}, w_{i-1})$

The **getAlphaTrigram** function shown below implements equation 15. to compute the trigram discount. Here we use it to compute the trigram discount for $q_{BO}(house \mid sell, the)$:

```
## Returns the total probability mass discounted from all observed trigrams.
## calculated from equation 14. This is the amount of probability mass which is
## redistributed to UNOBSERVED trigrams. If no trigrams starting with
## bigram$ngram[1] exist, 0 is returned.
##
## obsTrigs - 2 column data.frame or data.table. The first column: ngram,
##            contains all the observed trigrams that start with the bigram
##            prefix we are attempting to the predict the next word of. The
##            second column: freq, contains the frequency/count of each trigram.
## bigram - single row frequency table where the first col: ngram, is the bigram
##          which are the first two words of unobserved trigrams we want to
##          estimate probabilities of (same as bigPre in other functions listed
##          prior) delimited with '_'. The second column: freq, is the
##          frequency/count of the bigram listed in the ngram column.
## triDisc - amount to discount observed trigrams
getAlphaTrigram <- function(obsTrigs, bigram, triDisc=0.5) {
    if(nrow(obsTrigs) < 1) return(0)
    alphaTri <- 1 - sum((obsTrigs$freq - triDisc) / bigram$freq[1])

    return(alphaTri)
}

bigram <- bigrs[bigrs$ngram %in% bigPre, ]
alpha_trig <- getAlphaTrigram(obs_trigs, bigram, gamma3)
alpha_trig
```

```
## [1] 0.5
```

Step 4. v. Calculate unobserved trigram probabilities $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$:

```
## Returns a dataframe of 2 columns: ngram and prob.  Values in the ngram
## column are unobserved trigrams of the form: w3_w2_w1.  The values in the prob
## column are q_bo(w1 | w3, w2) calculated from equation 17.
##
## bigPre -  single-element char array of the form w2_w1 which are first two
##           words of the trigram we are predicting the tail word of
## qboObsBigrams - 2 column data.frame with the following columns -
##                 ngram: observed bigrams of the form w2_w1
##                 probs: the probability estimate for observed bigrams:
##                        qbo(w1 | w2) calc'd from equation 10.
## qboUnobsBigrams - 2 column data.frame with the following columns -
##                   ngram: unobserved bigrams of the form w2_w1
##                   probs: the probability estimate for unobserved bigrams
##                          qbo(w1 | w2) calc'd from equation 16.
## alphaTrig - total discounted probability mass at the trigram level
getUnobsTriProbs <- function(bigPre, qboObsBigrams,
                             qboUnobsBigrams, alphaTrig) {
    qboBigrams <- rbind(qboObsBigrams, qboUnobsBigrams)
    qboBigrams <- qboBigrams[order(-qboBigrams$prob), ]
    sumQboBigs <- sum(qboBigrams$prob)
    first_bigPre_word <- str_split(bigPre, "_")[[1]][1]
    unobsTrigNgrams <- paste(first_bigPre_word, qboBigrams$ngram, sep="_")
    unobsTrigProbs <- alphaTrig * qboBigrams$prob / sumQboBigs
    unobsTrigDf <- data.frame(ngram=unobsTrigNgrams, prob=unobsTrigProbs)

    return(unobsTrigDf)
}

qbo_unobs_trigrams <- getUnobsTriProbs(bigPre, qbo_obs_bigrams,
                                       qbo_unobs_bigrams, alpha_trig)
qbo_unobs_trigrams
```

```
##            ngram        prob
## 1 sell_the_house 0.357142857
## 2   sell_the_EOS 0.047619048
## 3   sell_the_the 0.047619048
## 4   sell_the_buy 0.035714286
## 5 sell_the_paint 0.005952381
## 6  sell_the_sell 0.005952381
```

Step 5. Select w_i with the highest $q_{BO}(w_i \mid w_{i-2},\, w_{i-1})$

We've done all the calculations required to make our prediction. These are summarized in the table below:

```
getPredictionMsg <- function(qbo_trigs) {
    # pull off tail word of highest prob trigram
    prediction <- str_split(qbo_trigs$ngram[1], "_")[[1]][3]
    result <- sprintf("%s%s%s%.4f", "highest prob prediction is >>> ", prediction,
                      " <<< which has probability = ", qbo_trigs$prob[1])
    return(result)
}
```

```
qbo_trigrams <- rbind(qbo_obs_trigrams, qbo_unobs_trigrams)
qbo_trigrams <- qbo_trigrams[order(-qbo_trigrams$prob), ]  # sort by desc prob
out_msg <- getPredictionMsg(qbo_trigrams)
out_msg
```

```
## [1] "highest prob prediction is >>> book <<< which has probability = 0.5000"
```

Going a little deeper: Exploring an interesting question

Any good data scientist at this point would be asking themselves some questions about their results, especially if they have not worked with a particular algorithm before. A simple first check might be to test whether all the $q_{BO}(w_i \mid w_{i-2}, w_{i-1})$ values sum to 1:

```
sum(qbo_trigrams$prob)
```

```
## [1] 1
```

That looks O.K., so let's explore something more interesting to see if we can deepen our understanding.

In the example we just completed, our prediction of **book** was based on the fact that $q_{BO}(book \mid sell, the)$ was higher than any other $q_{BO}(w_i \mid sell, the)$. But this wasn't really very interesting because *sell the book* was an observed trigram and the next closest probability $q_{BO}(house \mid sell, the)$ was based on an unobserved trigram *sell the house*. This leads us to wonder if observed trigrams always trump unobserved trigrams.

We can prove to ourselves that this is not the case with a simple experiment. Let's redo the above calculations with increased discount rates at both bigram and trigram levels. If we increase our discount rates from 0.5 to 0.7, what happens? If we set $\gamma_2 = \gamma_3 = 0.7$, these are the results we get:

```
gamma2=0.7; gamma3=0.7  # initialize new discount rates

obs_trigs <- getObsTrigs(bigPre, trigs)
unobs_trig_tails <- getUnobsTrigTails(obs_trigs$ngram, unigs)
bo_bigrams <- getBoBigrams(bigPre, unobs_trig_tails)
# separate bigrams which use eqn 10 and those that use 16
obs_bo_bigrams <- getObsBoBigrams(bigPre, unobs_trig_tails, bigrs)
unobs_bo_bigrams <- getUnobsBoBigrams(bigPre, unobs_trig_tails, obs_bo_bigrams)
# calc obs'd bigram prob's from eqn 10
qbo_obs_bigrams <- getObsBigProbs(obs_bo_bigrams, unigs, gamma2)
# calc alpha_big & unobs'd bigram prob's from eqn 16
unig <- str_split(bigPre, "_")[[1]][2]
unig <- unigs[unigs$ngram == unig,]
alpha_big <- getAlphaBigram(unig, bigrs, gamma2)
# distrib discounted bigram prob mass to unobs bigrams in prop to unigram ML
qbo_unobs_bigrams <- getQboUnobsBigrams(unobs_bo_bigrams, unigs, alpha_big)
# calc trigram probabilities - start with observed trigrams: eqn 12
qbo_obs_trigrams <- getObsTriProbs(obs_trigs, bigrs, bigPre, gamma3)
# finally, calc trigram unobserved probabilities: eqn 17
bigram <- bigrs[bigrs$ngram %in% bigPre, ]
alpha_trig <- getAlphaTrigram(obs_trigs, bigram, gamma3)
qbo_unobs_trigrams <- getUnobsTriProbs(bigPre, qbo_obs_bigrams,
                                       qbo_unobs_bigrams, alpha_trig)
qbo_trigrams <- rbind(qbo_obs_trigrams, qbo_unobs_trigrams)
```

```
qbo_trigrams <- qbo_trigrams[order(-qbo_trigrams$prob), ]
getPredictionMsg(qbo_trigrams)
```

```
## [1] "highest prob prediction is >>> house <<< which has probability = 0.4351"
```

While changing the discount rate doesn't always change a prediction, this example shows how it can. As mentioned earlier, the values we should use for these discount rates should be determined by cross-validation and will be strongly dependent upon the corpus used for the training. Determining the parameters for this model is the focus of Part 4 of this series (expected release Sep 2016).

References

[1] Wikipedia: Katz's Back-off Model -
https://en.wikipedia.org/wiki/Katz's_back-off_model
[2] Wikipedia: Generative Grammar -
https://en.wikipedia.org/wiki/Generative_grammar
[3] Dan Jurafsky on N-grams -
https://www.youtube.com/watch?v=s3kKlUBa3b0
[4] Dan Jurafsky on Estimating N-grams Probabilities -
https://www.youtube.com/watch?v=o-CvoOkVrnY
[5] Dan Jurafsky on Good-Turing -
https://www.youtube.com/watch?v=XdjCCkFUBKU
[6] Dan Jurafsky on Kneser-Ney -
https://www.youtube.com/watch?v=wtB00EczoCM
[7] Wikipedia: Chain rule for conditional probabilities -
https://en.wikipedia.org/wiki/Chain_rule_(probability)
[8] Large Language Models in Machine Translation - Brants, Popat, Xu, Och, and Dean -
http://www.aclweb.org/anthology/D07-1090.pdf
[9] Quanteda Quick Start -
https://cran.r-project.org/web/packages/quanteda/vignettes/quickstart.html
[10] An Introduction to Statistical Learning -
http://www-bcf.usc.edu/~gareth/ISL/
[11] Trevor Hastie & Rob Tibshirani on Cross-Validation (part 1) -
https://www.youtube.com/watch?v=_2ij6eaaSl0
[12] Trevor Hastie & Rob Tibshirani on K-Fold Cross-Validation (part 2) -
https://www.youtube.com/watch?v=nZAM5OXrktY
[13] Trevor Hastie & Rob Tibshirani on Right & Wrong Ways to do CV (part 3) -
https://www.youtube.com/watch?v=S06JpVoNaA0

Appendix 1 - Michael Collins Coursera Lectures

Michael Collins of Columbia gave an excellent class on NLP which used to be available on Coursera but was removed sometime during the first half of 2016. At the time of this writing, many of these lectures had been put on youtube and the two referenced on this page could be found at the following links:

Discounting methods: Part 1 - Katz Bigram - https://www.youtube.com/watch?v=hsHw9F3UuAQ

Discounting methods: Part 2 - Katz Trigram - https://www.youtube.com/watch?v=FedWcgXcp8w

Appendix 2 - The little test corpus:

```
SOS buy the book EOS
SOS buy the book EOS
```

```
SOS buy the book EOS
SOS buy the book EOS
SOS sell the book EOS
SOS buy the house EOS
SOS buy the house EOS
SOS paint the house EOS
```