

CS224N Final project: Language Model Methods and Metrics

Gary Luu
Stanford University
gluu@cs.stanford.edu

Ryan Fortune
Stanford University
rfortune@stanford.edu

June 7, 2008

Abstract

We first present skip n -grams interpolated with various other n -grams and measure their ability to improve performance of the language models in terms of perplexity. We then present a language model that relies on content words, words which are uncommon. We then present a bag generation algorithm, and use bag generation as a metric for our language models. We then present a language model that uses clustering by part of speech tags.

1 Introduction

In this paper, we examine various methods of improving language models, as well as discuss a possible metric. All language models were trained and tested using the `europarl` corpus. All interpolated models have their weights λ_i calibrated by using a discretized sample space over $[0, 1]$ and iterating the discrete weight values. The typical discretization was 0.05.

Most of our methods are motivated by a tutorial presented by Goodman, titled *The State of the Art in Language Modeling* [4].

2 Skip N -grams

One of the limitations of n -grams is that as n grows, the memory requirements grow rapidly due to dimensionality, and also because of dimensionality, our training data becomes more sparse relative to the possible number of n -grams. One way to extend the influence of words further away from our query word without the excessive memory requirement is to use skip n -grams. Skip n -grams are also useful in many different permutations, for example, a 3-dimensional skip n -gram that uses w_{i-1} and w_{i-3} as the history, i.e. $P(w_i|w_{i-1}w_{i-3})$.

2.1 Method

Our method was to take the unsmoothed skip n -gram $P(w_i|w_{i-2})$ and see how performance would improve when we interpolated it with a bigram as we increased the training set size, and also compare it to a standard trigram model. That way we could see the utility in using the skip n -gram as a replacement for the trigram.

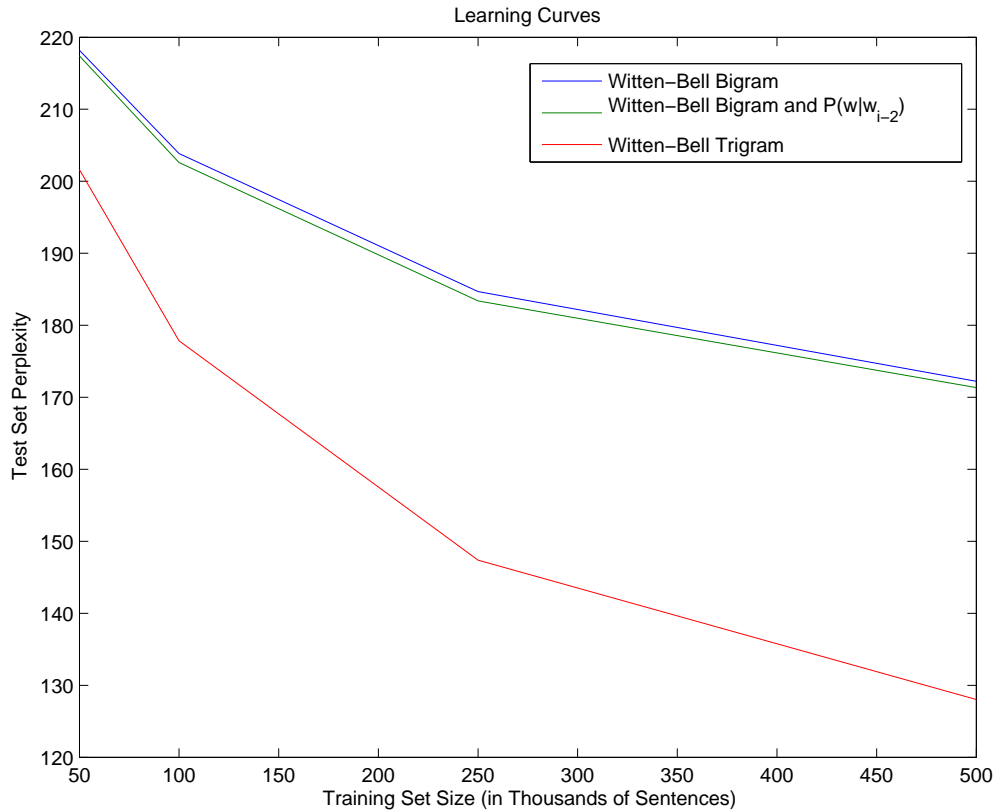


Figure 1: Skip n -gram performance relative to bigram and trigram baselines

2.2 Learning Curve

Here we show the value of skip n -grams as the training set size grows relative to the baseline bigram and trigram models, which use Witten-Bell smoothing. See Figure 1.

2.2.1 Discussion

As one can see in the data, using a skip n -gram with a bigram model did cause perplexity scores to decrease similarly to the bigram, with a slight constant difference. The trigram model had a much steeper amount of performance improvement with more data. It is likely that the increased training data is more beneficial for higher dimensional models than lower dimensional models, which do not suffer as much from sparsity with the smaller data sets as the higher dimensional models do.

3 Content Word Model

Oftentimes, n -grams perform poorly because their history is limited. Because of this limited history, phrases can often be rearranged and be scored similarly by the n -gram. In essence, we attempted to capture the idea of sentence *context* using a “content” word model, where a “content” word is

one which is uncommon, since the most common words are articles and prepositions. However, since prepositions and articles are required to be grammatically correct and fluent, a content word model could not strictly be used by itself.

We created the content word model as follows. Let S be a sentence. Let C be the last content word found in the sentence, where C is the special token ϵ if we have not seen any content words. Then our language model is

$$P(S) = \prod_i [\lambda_1 P(w_i | w_{i-n+1}^{i-1}) + \lambda_2 P(w_i | C)]$$

Where $P(w_i | w_{i-n+1}^{i-1})$ can be replaced by any arbitrary n -gram like language model, and $\lambda_1 + \lambda_2 = 1$. $P(w_i | C)$ is computed from the joint bigram distribution of content words and all words like so

$$P(w_i | C) = \frac{P(w, C)}{\sum_w P(w, C)}$$

Where the bigram distribution is calculated using a simple MLE count divided by the total number of content words and all words count. Note that a content word $C \neq w_i$ when computing $P(w_i | C)$ unless C occurs more than once in the sentence. Also note that C is not a fixed position away from w_i . We did not perform smoothing on this model. C is always the most recent content word in the history.

3.1 Content Word Definition

We examined using different sets of words as the non-content words. Our method was to find a list of the most commonly used words in the English language, and look for a cutoff where we could improve the perplexity of the language model. We started with an initial set of 250 most common english words, in order, obtained online through an internet search [7].

3.2 Method

We interpolated the content word model with various other models and plotted the performance improvement. We reduced the common words list down to the 25 most common words, as larger lists had no effect. We tested this content model with Kneser-Ney and Good-Turing smoothing. See Figure 2.

3.3 Discussion

The content word model provided little to no benefit with these language models. In the case of the trigram model, there was absolutely no benefit at all. This is likely due to the fact that the common word removal schemes behave somewhat like trigram and skip n -grams, but with little to add in terms of grammatical fluency. It would be worth looking into building a composite bigram/content word model $P(w_i | w_{i-1}, C)$ in future work.

4 Bag Generation Based Metrics

We developed a metric based on *bag generation*, as discussed in exercise 4.9 in Jurafsky and Martin [1]. More specifically, the *bag generation* task uses a language model and searches for the most

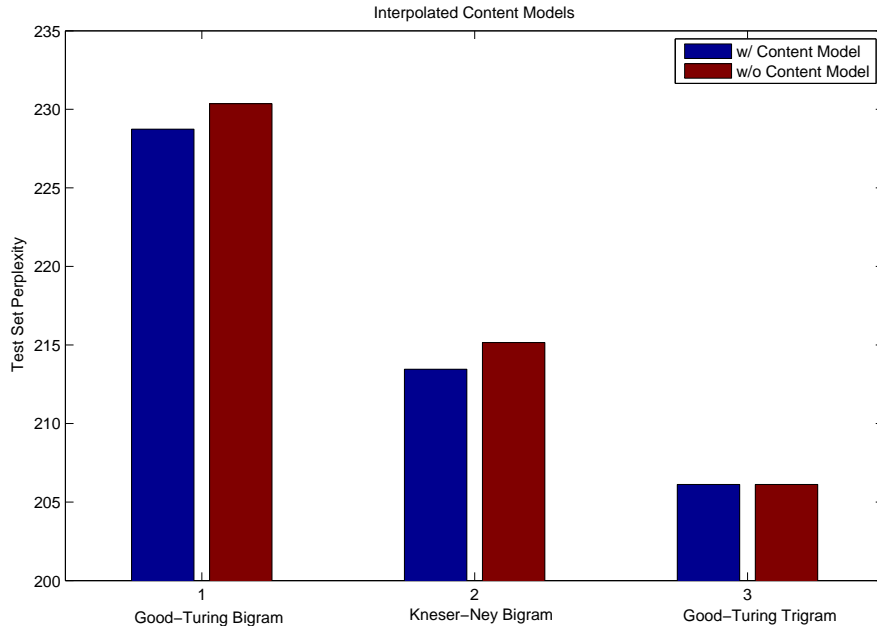


Figure 2: Performance of Various Language Models Interpolated with Content Model

probable sequence of words given a *bag* (a set) of words. This task is NP-complete. Informally, this task is very similar to machine translation tasks without constraints to make it solvable in polynomial time. Arbitrary word-ordering is known to be NP-complete [3]. Therefore, we resorted to greedy algorithms in order to run this metric.

4.1 Greedy Algorithm Description

The algorithm can be essentially described as random-restart greedy hill-climbing. We initialize the hill-climbing search with permutations drawn uniformly at random, then greedily look for more probable permutations. Each node in the search tree represents a permutation, and we limit the expansion of each node to a number of nodes equivalent to the length of the sentence. Each child node resulting from this expansion is the result of swapping the first word of the parent node and another word in the parent node’s sequence. In this way, each word can move to its correct position within two node expansions. However, the problem with this is that swapping to the first position can easily be a local optima. On average, it was found that the average number of nodes expanded during search was typically less than 10. Therefore, as often is the case with random-restart greedy hill-climbing algorithms, it is more beneficial to try many initial positions since the search is rather shallow, and there are many local optima. The fact that this algorithm does not always find the optimal permutation was taken into account in designing our metrics.

4.2 Stability Metric

Since our algorithm does not always find the optimal permutation, one way to measure the quality of a language model is to instantiate the initial permutation with the correct sentence. In this

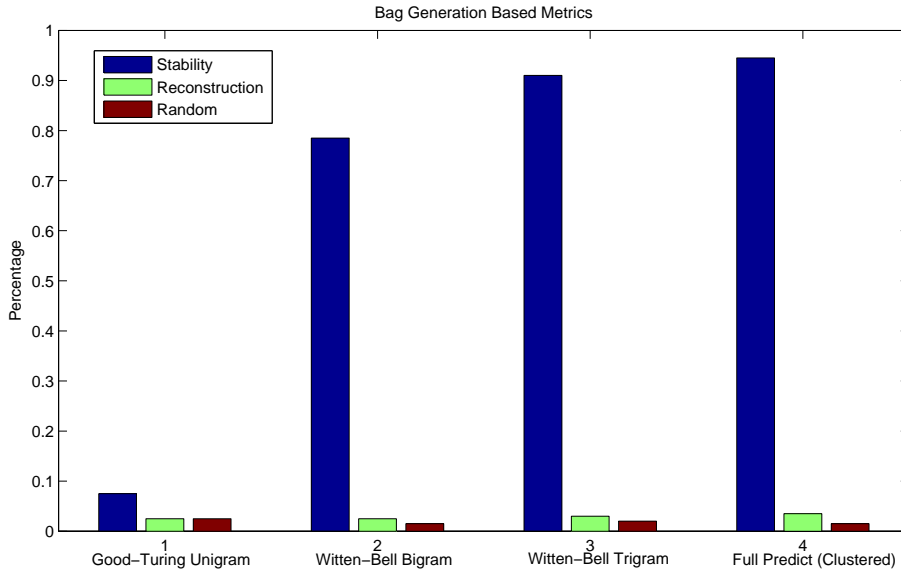


Figure 3: Bag Generation Based Metrics Across Increasingly Superior Models

way, if a more probable permutation of the words is found, we can examine the found permutation and see if it is also a sentence which is as fluent as the correct sentence. We found that this was generally not the case, unless a sentence was short, in which case the optimal permutation was usually found. In general, if a sentence was found by the language model to be better than the correct sentence, it tended to be logical in phrases, but nonsensical in its full construction. This is most likely because transitions between phrases were all roughly equally unlikely. We describe language models that perform well in this metric as *stable*.

4.3 Reconstruction Metric

Another method of using bag generation to measure the quality of a language model is to initialize it with random permutations and see if it can reconstruct the original sentence, or at least a coherent, fluent sentence. We first measured the frequency with which a random permutation was the correct sentence in the first place. If the language model tended to be stable, it would have luckily reconstructed the original and maintained its position as the most probable sentence. Therefore we looked for how much better than random our language models performed in this metric.

4.4 Discussion

We plot the metric using models of various quality, where quality is based upon measured test set perplexity. At the low end, we use a unigram, then progress through bigrams, trigrams, and then our clustering model. The “Random” category indicates the probability of successful reconstruction of the correct sentence. See Figure 3.

4.4.1 Computational Cost

The problem with bag generation based metrics is that bag generation itself is NP-complete. Any metric that uses bag generation is almost certainly going to use the language models sentence probabilities as a part of any search heuristic. We found that for certain language models, namely our clustered language model, that the cost of determining a sentences probability increases by a constant c , where c is the number of language models being interpolated between. Therefore, for each sentence, determining its probability is cn , which can be even worse if the algorithm for determining the sentence probability is non-linear. For each node we expanded in search, we computed the sentence probability. Since on average, we expanded roughly 10 nodes per hill-climbing search, and each expansion incurred n probability computations, and we randomly restarted 10 times, we performed an $O(n^2)$ algorithm for each sentence, with large constant factors. For m sentences, where m is typically much larger than n , we incur a cost of $O(mn^2)$ with large constant factors. Compare this with perplexity, which only incurs a cost of $O(mn)$ with much smaller constant factors, and we can see that computing this metric over a large test set becomes overly computationally expensive relative to perplexity.

4.4.2 Stability

Despite these shortcomings however, the stability metric is useful in determining flaws and idiosyncracies in a language model. When a language model produces a most probable sentence which is not the correct sentence, it reveals the peculiarities of that particular language model, and shows how the language model either underfits, or more typically, overfits the training data.

4.4.3 Reconstruction

The reconstruction metric, on the other hand, is much less valuable. On average, very few permutations using our greedy algorithm lead to a successful reconstruction of the original sentence, and it is difficult to attribute improvement in this metric to much more than luck in finding a good initial condition in the search.

5 Part of Speech Clustering

Clustering is a method that has been demonstrated to improve the quality of language modeling. In the Goodman paper, the most effective clustering method that was not a linear combination of others was the `IBMFullPredictModel`.

$$P_{fullibmpredict}(w|w_{i-2}, w_{i-1}) = (\lambda P(W|w_{i-2}, w_{i-1}) + (1 - \lambda)P(W|W_{i-2}, W_{i-1})) * \\ (\mu P(w|w_{i-2}, w_{i-1}, W) + (1 - \mu)P(w|W_{i-2}, W_{i-1}, W))$$

This model estimates both the probability of a cluster appearing given the history and the probability of a word given its cluster and history and multiplies the two together. Each probability is smoothed by the equivalent probability using the cluster labels of the history instead.

5.1 Part of Speech Tagging

The next step in building the model was choosing the method of clustering. Clustering by parts of speech seemed like a natural choice as languages have grammar rules that can be learned or at least approximated by clustering techniques. Parsing sentences to find parts of speech wasn't a viable option primarily because each word needs to appear in a single cluster for the model to work. We found the breadth of vocabulary of WordNet to be an asset [5]. WordNet doesn't have an explicit part of speech tagger, but given a word, it will provide a list of all possible semantic meanings for the word. Each semantic meaning has a part of speech label, and we simply count the labels. Which ever label is most frequent is used as the tag. Remember that creating useful clusters is the objective, not creating the most accurate tagger.

The interface of WordNet we used only tagged root words, so plurals and different verb tenses were unrecognized [6]. There was a simple stemmer provided, but this could not handle irregular forms. We manually tagged different forms of the verb "be" and a handful of other common words. Additionally, WordNet only contains nouns, adjectives, adverbs, and verbs, so the most common words in English were without a tag. We decided to create a list of articles, many prepositions, interrogatives, different forms of pronouns, demonstratives, interjections, conjunctions, and helping verbs. All together, nearly 180 words were manually listed and many of them were some of the most common English words. In the final version of the part of speech tagger, any word not in WordNet or the manual list was tagged as unknown.

5.2 IBM Full Predict Model

The IBM Full Predict model combines four separate trigram models. The Goodman paper suggests that using a Kneser-Ney trigram model is effective, but since we already had a Witten-Bell trigram model implemented, we used this as the base for our four models.

5.2.1 Cluster Models

The models for finding the conditional probability of a cluster had relatively straight forward implementations. The exception was with the unigram model being used at the base. The original language model used a Good-Turing Unigram model, however with the small number of possible clusters, 21, the Good-Turing algorithm was inadequate. Instead we created a basic empirical unigram cluster plus one model. The small number of clusters relative to overall totals outweighs the negatives of the methods. As shown in the first assignment, the Witten-Bell algorithm produces a valid probability distribution given valid base models, which the above model and the two MLE models are. These models combine to produce $P(W_i|w_{i-2}, w_{i-1})$.

5.2.2 Conditional Models

The other two models are slightly more complex. Although they are both trigram dependent in that they consider a word and the two preceding words, the models condition on four pieces of information. This means that the unigram base for the Witten-Bell algorithm is actually a conditional probability, in particular the probability of a word given its cluster. The computation of this probability is much simpler because of an assumption we made earlier, that a word be

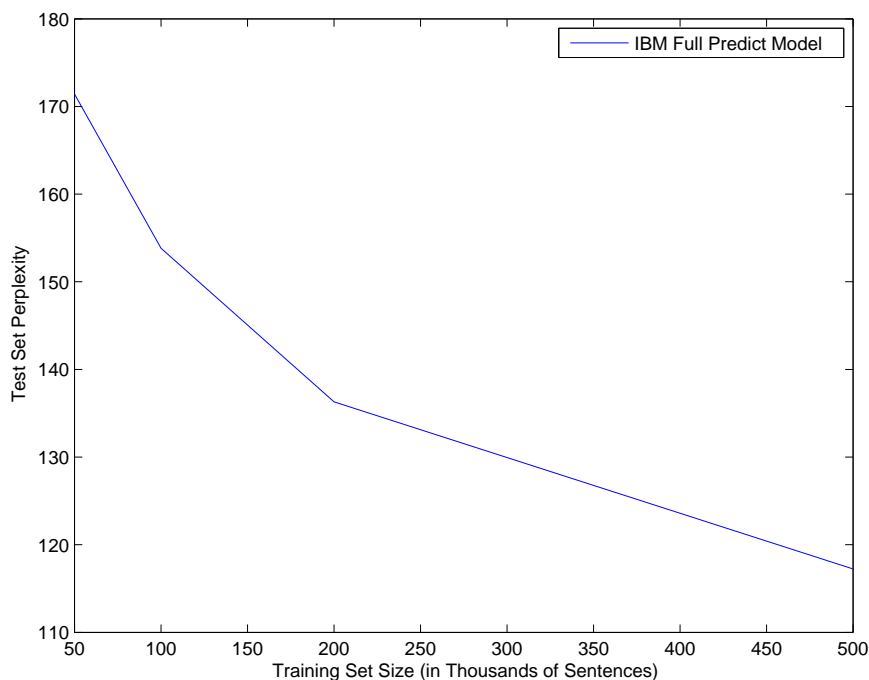


Figure 4: Learning Curve for IBM Full Predict Model

assigned to a single cluster. Bayes' rule states:

$$P(w_i|W_i) = P(W_i|w_i) * \frac{P(w_i)}{P(W_i)}$$

The assumption sets $P(W_i|w_i) = 1$ leaving the probability we're looking for as the likelihood of a word divided by the likelihood of a cluster. We already have models for both of these, the Good-Turing Unigram for the word, and the empirical unigram cluster plus one for the cluster. The only issue is how to handle words that were not present in the training data. Good-Turing classifies them as unknown, but the part of speech tagger could assign the word to any cluster. Our solution is to assume that an unknown word is equally likely to be a part of any cluster, and use that to estimate the probability. This groups multiple words into one category, but the alternative is to limit the number of words in the vocabulary. We adjust the other probabilities to maintain the proper distribution.

With all four models completed, the next step was setting the two parameters. Running the model is time consuming, so we decided to set the parameters before running instead of determining them dynamically. After several iterations, we settled on parameter values of $\lambda = .85$ and $\mu = .75$. These values agreed with the intuition that using specific word histories is more meaningful than using the clusters of those words as the history.

5.3 Results

As the results indicate, the full predict model was a definite improvement over the original Witten-Bell trigram with a drop of 30 in the perplexity with the standard training set. Increasing the training set allowed us to reduce the perplexity to 117 with a 500k sentence training set. See Figure 4. As expected, clustering provided a noticeable benefit to language modeling. The Goodman paper suggests that combining clustering with other techniques including skip-grams is the optimal approach to language modeling.

References

- [1] Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing*. 2008.
- [2] Stanley Chen and Joshua Goodman. *An Empirical Study of Smoothing Techniques for Language Modeling*. 1998.
- [3] Ulrich Germann, Michael Jahr, Kevin Knight, Daniel Marcu, and Kenji Yamada. *Fast Decoding and Optimal Decoding for Machine Translation*. 2001.
- [4] Joshua Goodman. *The State of the Art in Language Modeling*. 2002.
- [5] Princeton University. *WordNet 3.0*. <http://wordnet.princeton.edu> 2006.
- [6] Mark A Finlayson. *MIT Java WordNet Interface*. <http://www.mit.edu/markaf/projects/wordnet> 2008.
- [7] Jerry Jones. *1000 Most Common Vocabulary Words in English*. http://esl.about.com/library/vocabulary/bl1000_list1.htm. 2008.