Statistical language modeling with n-grams for speech recognition

Arnaud CHEVALIER
arnche@kth.se

Nicolas ROUSSEL
nroussel@kth.se

Abstract

In this project, we develop statistical language models using n-grams. We trained our models on a corpus of English sentences and investigated different smoothing methods. Their performances are measured using two tools: perplexity which is a metric to evaluate the predictive power of a probability model, and word error rate which measures the distance between two sequences of words. Although the experiments with perplexity are not really convincing, we manage to compare the smoothing methods using the word error rate metric on held out data.

1 Introduction

1.1 Context in speech recognition

Automatic speech recognizers use many different models in order to transcript speech. They use acoustic, lexical and language models. Those models, applied to the feature extracted from the raw data in a decoder are supposed to help transcribing the pronounced words.
Language models are of different kinds. Some of them are grammar based, such as Chomsky's formal grammar [1]. It contains a set of constituents and rules to combine them. Other models are purely statistical, they are called n-grams models.

1.2 Statistical models for language

A n-gram model is a probability distribution over a set of sequences of n words. By counting the number of occurrences of sequences of words in a training set and dividing by the number of different such sequences, it is possible to assign probabilities to each of these sequences. For instance, one of the most widely used models is the trigram, which computes probabilities for all sequences of three words existing in the training set. What is hidden behind this very intuitive approach is in fact the maximum likelihood estimator.
When going through sentences of a test set, one can then estimate the probability of a word w_i given the n-1 previous words denoted by $w_{i-n+1}, \ldots, w_{i-1}$. Using the context given by these past words, we are in fact approximating $P(w_i)$ by P($w_i \mid w_{i-n+1}, \ldots, w_{i-1}$), thus making the assumption that w_i depends only on its n-1 previous words and not on those before. This is known as the n^{th}-order Markov assumption.
A question is how to choose the right order n? The efficiency of the model depends largely on the amount and quality of the training data. These factors influence the choice of the order. Intuitively, n should be limited by the amount of data available for training. If we want sophisticated models, it is tempting to go for high orders of n-grams, the models may suffer from the lack of data. As an example, most 4-grams or 5-grams do not occur in an average-sized training corpus so for them it does not make sense to use models with n > 3. Nonetheless,too small orders will not give good results either because they do not use enough context. We will see later methods to combine different orders of n-grams.

29th Conference on Neural Information Processing Systems (NIPS 2016), Barcelona, Spain.

1.3 The need for smoothing

A fundamental issue with n-grams models is that they might encounter new words or unseen contexts for known words in a test set. Because those sequences have never been seen in the training set, they would be assigned a probability of zero by the model. A good model should not assign a probability equal to zero to a sentence as soon as it contains an event unseen during training. This problem is solved thanks to smoothing methods.
In the field of natural language processing, we call smoothing any technique which aims at improving the maximum likelihood distribution by slightly modifying the weights of some or all of the n-grams. Doing this helps balancing the distribution between frequent and rare n-grams and hence compensate the data sparsity by sharing some probability mass. Different smoothing methods exist to make the model more flexible to unknown events, but they all lead to different performances of the model. In this work, we study n-grams models and try to compare how smoothing methods perform.

1.4 Model evaluation

The main error measure is the word error rate of sentences taken in a test set. Spoken utterances are inputed to a speech recognizer which in turn outputs different hypotheses of text sentences with associated probabilities. These hypotheses are then re-scored with our n-gram language model and the word error rate of the best hypothesis is computed.

2 Methods

Using a trained n-grams model, the probability $P(w_1^k)$ of a complete sentence $w_1, ..., w_k$ is computed thanks to the chain rule like this :

$$P(w_1^k) = P(w_k|w_1^{k-1})P(w_1^{k-1}) = \prod_{i=1}^{k} P(w_i|w_{i-n+1}^{i-1}) \tag{1}$$

This formula shows that $P(w_1^k) = 0$ if w_n is unknown or seen in an unknown context. It also appear necessary to add n-1 "Start Of Sentence" (SOS) tokens at the beginning of the sentence in order to compute the n-1 first terms of the product.
In this section, we explain how a model copes with unknown words, what the different smoothing techniques are, and how they combine several n-grams orders together (back-off and interpolation).

2.1 Fixing unknown words

As said in introduction, unknown words in a test set are a problem for n-grams models. A first solution is to work with a close vocabulary. In this framework, the test set only contains words existing in the training set. This hypothesis seems to be too restrictive for a speech recognizer so we chose to work with an open vocabulary. In this new framework, the unknown-words issue is solved by adding a word "unknown" (UNK). Each word of the training set appearing only once is modified by UNK and the first occurrence of the other words are counted as occurrences of UNK. After this transformation of the training data the pseudo-word UNK is treated as any other word for the training of the n-gram. [2] (D. Jurafsky and al , 2014, chapter 4)
To solve the issue of known words appearing in an unknown context, several smoothing methods can be used. We implemented a few of these and will compare their performance in the section “results”.

2.2 Laplace smoothing

The principle of this smoothing method is to make every n-gram possible, by adding one occurrence for every one of them. As a consequence, n-grams never occurring are now occurring once, and n-grams occurring k times now occur k+1 times. Probabilities are then normalized by a factor taking into account the new number of occurrences of all the n-grams [2] (D. Jurafsky and al , 2014, chapter 4).

2.3 Good-Turing

This smoothing method aims at distributing probability density of n-grams appearing many times to those appearing fewer times. Particularly, n-grams which are never seen have some probability mass allocated. If r denotes the count of an n-gram and n_r the number of n-grams with this count then the Good Turing approximation modifies the counts such as :

$$r* = (r+1)\frac{n_{r+1}}{n_r} \tag{2}$$

Then it is necessary to change those counts into probabilities. The counts $r*$ are normalized by a coefficient N which value is given by [3] Good, I.J (1953) :

$$N = \sum_{r=1}^{\infty} r n_r \tag{3}$$

This method has some issues because $r*$ depends on n_{r+1} and n_{r+1} may be equal to zero. Good Turing approximation is then not perfect and requires some improvements. For example it can be used along with a backoff method.

2.4 Katz Back-off

The idea of back-off is to reduce the context if the word is not known in the large context. Instead of looking for the n-1 previous words, the algorithm will only look at the n-2 previous words and see if this is an existing context of words in the n-1 -gram model. We can of course back off further and decrease the order of n-gram until a non-zero probability is found.

In [4], Katz gives the following recursive formula to compute the probabilities of a n-gram model with backoff:

$$P_{backoff}(w_i|w_{i-n+1}^{i-1}) = \begin{cases} P(w_i|w_{i-n+1}^{i-1}) \text{ if } count(w_{i-n+1}^{i}) > k \\ \alpha(w_{i-n+1}^{i-1}) P_{backoff}(w_i|w_{i-n+2}^{i-1}) \text{ else} \end{cases} \tag{4}$$

and $\alpha(w_{i-n+1}^{i-1})$ is given by :

$$\alpha(w_{i-n+1}^{i-1}) = \frac{1 - \sum_{w_i : c(w_{i-n+1}^{i})} P(w_i|w_{i-n+1}^{i-1})}{1 - \sum_{w_i : c(w_{i-n+1}^{i})} P(w_i|w_{i-n+2}^{i-1})} \tag{5}$$

When backing off, this formula uses a pre-factor α which is a normalization weight.

2.5 Interpolation

We can solve the issue mentioned in the previous section in a different way. Instead of backing off to a smaller order, we can compute the probability of w_n given $w_1, \ldots, w_{n-1}$ as a weighted sum of the probabilities in every k-gram model for $k \leq n$. [5] (M Collins)

$$P_{interpolation}(w_i|w_{i-n+1}^{i-1}) = \sum_{k=2}^{n} \lambda_k P(w_i|w_{i-k+1}^{i-1}) + \lambda_1 P(w_i) \tag{6}$$

The coefficients lambdas have to be determined such that this interpolation algorithm is optimized. One way to do that is to compute the lambdas using the expectation maximization algorithm to optimize the likelihood of some validation data given these parameters. In [6], Brychcin details the application of this iterative algorithm to the case of tuning the interpolation's parameters λ:

- Initialize weights λ_k^0 such that $\sum_{k=1}^{K} \lambda_k^0 = 1$.

- Expectation: for time iteration t, compute the expected value of the parameters with:

$$\lambda_j^t* = \frac{\lambda_j^t P_j(w_i|w_{i-j+1}^{i-1})}{\sum_{w_{i-n+1}^i} \lambda_k^t P_k(w_i|w_{i-k+1}^{i-1})} \tag{7}$$

  In this equation, P_k denotes the probability computed by the k-gram model.

- Maximization: normalize the weights for the next step time by:

$$\lambda_j^{t+1} = \frac{\lambda_j^t*}{\sum_{k=1}^K \lambda_k^t*} \tag{8}$$

- Iterate expectation and maximization steps until the condition $|\lambda_j^{t+1} - \lambda_j^t| < \epsilon$ is verified for any j, for a predefined threshold ϵ.

3 Experiments

This section presents our experiments, the data used, which models are tested and how their performances are measured.

3.1 Data

We used a dataset containing 93 424 sentences in English. It is available on-line at the following address [7]:

`http://www.manythings.org/anki/`

This website offers data bases of pairs of sentences : the first is in a source language and the second in a target language. We chose the set translating from English to French. Its size is 141 382 pairs of sentences. We have to do a preprocessing step on this dataset in order to extract only the English sentences. The data set also contains different French translations for the same English sentence so we had to remove the sentences appearing several times, that is why the final size of our data set is only 93 424 sentences. Finally we reduce the complexity of the data removing all special character such as "; , ! ? : '" and changing every letter to lowercase.
Then we separate the data set in a training set and test set. The training set contains 80% of the data set and the test set the 20% left.

3.2 Protocol

In this work we compare smoothing methods. We implemented Laplace smoothing, the Good-Turing approximation, the Katz's back-off and interpolation. Different order of n-grams are also compared (from 2 to 5). According to [9], it is pertinent to compare smoothing methods on bigrams and trigrams models. Also the Katz's back-off and interpolation should be the best methods, after should come the Good-Turing approximation and finally the Laplace smoothing.
For the interpolation, we did not implement the expectation maximization algorithm to determine the coefficients but chose them with some arguments exposed later in the result part.
All those methods compute probabilities for a whole sentence as a product of probabilities (chain rule). Since this usually gives really small numbers, we do all the calculations in log-domain to avoid underflow.
We use two different metrics : perplexity and word error rate. The first one is an intrinsic evaluation of the model, meaning that it is based on a part of the data. The second method is extrinsic, indeed it evaluates the model directly on the output of a Speech Recognizer.

3.3 Perplexity

Intuitively perplexity measures, for a test set, the average number of words that could possibly follow a given sequence. For instance, if a test set has a perplexity of 200, it means that the model has the choice between 200 words for the next word. The better the model the lower its perplexity.

We compute the perplexity of our models thanks to the formula from [8] (J. Hockenmaier, 2012). That one gives the cross-entropy:

$$H(p,q) = -\sum_{x \in TestSet} p(x) log_2 q(x) \tag{9}$$

The perplexity is then given by:

$$P = 2^{H(p,q)} \tag{10}$$

In those formula, x in the sum runs through all the possible n-grams appearing in the test data, $p(x)$ denotes the frequency count of x in the test set and $q(x)$ the probability associated to x by the model we evaluate.

The results we got with perplexity are not convincing. Indeed we often obtain + infinity. This is due to the fact that in the training set used to build the models, the different n-grams have only few occurrences so their probabilities are very small and their log probabilities are very high negative numbers. The probabilities of n-grams in the test set are not as small as the ones of the model because the test set is smaller than the training set so less different n-grams appear. The consequence is that the product $p_i log(q_i)$ is a high negative number, typically around -20 and so the perplexity is very high. A solution could be to have larger training and validation sets.

3.4 Extrinsic evaluation of the language model

Perplexity is an intrinsic way of evaluating the language model (on a held-out part of the data). An extrinsic way of evaluating a LM is to use it at the output of a speech recognizer system which produces text when inputing a sound file of a spoken utterance. One can then compute the word error rate by counting the deletions, insertions and substitutions to see how well the language model performs. In this section, we explain the experiments we performed to assess the results.

We used IBM's Watson as a speech recognizer. A free trial version of the program is available here [10]:

`https://speech-to-text-demo.mybluemix.net/`

We used sentences from the test set as inputs of Watson's recognizer.
Given a speech sentence recorded with the microphone or a sound file, Watson computes the most likely text using Hidden Markov Models deep neural networks. When a sentence is unclear, it outputs several possibilities for some or every words in the sentence, along with the associated confidence score. We then form every possible combinations and input all the formed sentences in our different n-gram language models. The n-gram models computes their own scores for each sentence using probabilities obtained by the chain rule, and do their own ranking of the different hypotheses (sentences). It is then possible to compare the models using word error rate.
The word error rate is computed only counting the number of substitutions because our models doesn't allow insertions or deletions. Watson produces deletions and insertions but these are not errors from our model.

Definition of the metric
We used a special metric to compare the models. The performance of a model for a given sentence is defined as the average error rate over the 10% best rated hypotheses for this sentence.
The overall performance is the average for all the test utterances.

4 Results

4.1 Numerical results

Here are the results in terms of word error rate for the following models:

- $LAPLACE_N$: model using Laplace smoothing, up to maximal order of n-gram $n = N$.
- $GOOD - TURING_N$: model using Good-Turing approximation, up to maximal order of n-gram $n = N$.
- $INTERP_N^{equal}$: model using interpolation, up to maximal order of n-gram $n = N$. The weights λ_k are all equal.
- $INTERP_N$: model using interpolation, up to maximal order of n-gram $n = N$. The weights λ_k are proportional to the number of k-grams.

The reason for the choice of weights in the model $INTERP_3$ is that there are fewer unigrams than trigrams for instance, so in average unigrams receive higher probabilities. In consequence, a "naive" model would tend to privilege unigrams over trigrams, since they would give higher, so in this model we compensate by assigning higher weights to higher orders.

Model name	Average Word Error Rate
$LAPLACE_2$	0.261
$GOOD - TURING_2$	0.379
$INTERP_2^{equal}$	0.267
$INTERP_2$	0.263
$LAPLACE_3$	0.329
$GOOD - TURING_3$	0.370
$INTERP_3^{equal}$	0.1346
$INTERP_3$	0.1325
$LAPLACE_4$	0.409
$GOOD - TURING_4$	0.365
$INTERP_4^{equal}$	0.246
$INTERP_4$	0.248
$LAPLACE_5$	0.432
$GOOD - TURING_5$	0.361
$INTERP_5^{equal}$	0.244
$INTERP_5$	0.246

Table 1: Average Word Error Rate for different models

For each model, the result is an average over 15 spoken utterances recorded from sentences of the test set.

We can then make a plot of the Word Error Rate (WER) versus the order of the ngram.

4.2 Interpretation

The results of those experiments show a general trend (see figure 1). As expected, Laplace smoothing and Good-Turing approximation are less efficient than the two types of interpolation. It is consistent with the literature [9]. Without using our n-grams, Watson gives a mean Word Error Rate of 0.205 on the same test sentences. This means that on this test sentences at least, the interpolation model with $n = 3$ performed better than Watson. Of course, we would have to conduct further experiments with bigger test datasets to conclude on the global performance of the model.

Nonetheless, other observations are surprising. We would expect the word error rate to be decreasing with the order of the n-gram model. This behavior is only observed for the Good-Turing approximation which is slightly decreasing but the Laplace smoothing is increasing and the interpolation reaches a minimum for n=3. We don't really know how to explain the behavior of the interpolation; indeed, as the probability of an n-gram with the interpolation is a weighted sum of the smaller orders, we would expect the behavior of the n-gram model to be at least as good as the k-gram models k<n.

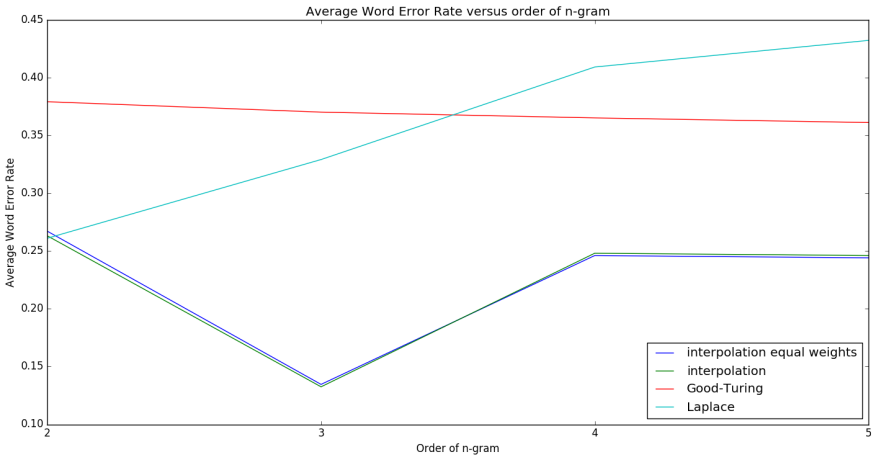

Figure 1: Word Error Rate versus order of the n-gram for different models

Also, in some cases our models show good abilities in re-scoring the hypothesis given by Watson (see figure 2). The models learned some grammatically right word combination.

do you know the right whom	do you need a ride home
do you know the ride whom	do you need a ride whom
do you know the right cool	do you know a ride home
do you know the ride cool	do you know the right whom
do you know the right who ll	do you need a ride will

Figure 2: 5 best hypotheses for Watson (left) vs interpolation n = 3 (right)

On the above figure, we see that for this particular sentence Watson gives as most probable result a sentence which is not grammatically correct ("do you know the right whom") so most likely not the right one, whereas the interpolation trigram model gives the real spoken utterance ("do you need a ride home") as the most probable.

Our models probably suffered from a lack of data. The counts of the high orders n-grams (n>3) were often very low so our models are very sparse.

5 Conclusion

In this paper we compared smoothing methods of n-grams models at different orders. Our trials to measure perplexity of the models were not successful. We think that it is because of a lack of data. Nevertheless, thanks to Word Error Rate measurements we were able to distinguish the models. Interpolation seems to give better error rates compared to Laplace and Good-Turing smoothings. Generally our models had poor performances but could somehow give sensitive scores to grammatically correct sentences when we applied them on top of a speech recognizer. We did not manage to measure perplexity.

Some ways to improve those results would be to use more training data and implement other smoothing methods such as Kneser-Ney smoothing.

References

[1] Noam Chomsky (1956). *Three models for the description of language* , IRE Transactions on Information Theory, no 2, p. 113–124.

[2] Daniel Jurafsky & James H. Martin (2014). *Speech and Language Processing.* Copyright c 2014. All rights reserved. Draft of September 1, 2014.

[3] Good, I.J (1953) *The population frequencies of species and the estimation of population parameters*, In Biometrika, 40 (3 and 4):237-264.

[4] Katz, S. M. (1987). *Estimation of probabilities from sparse data for the language model component of a speech recognizer*, IEEE Transactions on Acoustics, Speech, and Signal Processing.

[5] Michael Collins. *Course notes for NLP, Chapter 1.* Columbia University.

[6] Tomas Brychcin (2012). *Unsupervised methods for language modeling.* PhD study report, p 15.

[7] Tab-delimited bilingual sentence pairs, *French-English*, manythings.org (`http://www.manythings.org/anki/`)

[8] Julia Hockenmaier (2012) *Lecture 4 : Evaluating language models.* **CS498JH**: Introduction to NLP, university of Illinois.

[9] S. F. Chen and J. Goodman. (1996) *An empirical study of smoothing techniques for language modeling.* In Proceedings of the 34th annual meeting on Association for Computational Linguistics, pages 310–318. Association for Computational Linguistics

[10] Watson audio transcription, IBM : `https://speech-to-text-demo.mybluemix.net/`