Course Notes for COMS w4705: Language Modeling

Michael Collins

1 The Language Modeling Problem

Our task is as follows. Assume that we have a *corpus*, which is a set of sentences in some language. For example, we might have several years of text from the New York Times, or we might have a very large amount of text from the web. Given this corpus, we'd like to estimate the parameters of a *language model*.

A language model is defined as follows. First, assume that the set of all words in the language is $\mathcal{V}$: for example, we might have $\mathcal{V} = \{$the, dog, laughs, saw, barks, cat, $\ldots\}$. We assume that $\mathcal{V}$ is a finite set. A *sentence* in the language is a sequence of words

$$x_1 x_2 \ldots x_n$$

where the number of words $n \geq 1$, we have $x_1 \ldots x_{n-1} \in \mathcal{V}$, and we assume that x_n is a special symbol, STOP (we assume that STOP is not a member of $\mathcal{V}$). We'll soon see why it is convenient to assume that each sentence ends in the STOP symbol. Example sentences could be

the dog barks STOP
the cat laughs STOP
the cat saw the dog STOP
the STOP
cat the dog the STOP
cat cat cat STOP
STOP
...

We'll define $\mathcal{V}^\dagger$ to be the set of all sentences with the vocabulary $\mathcal{V}$: this is an infinite set.

A language model is then defined as follows:

Definition 1 (Language Model) *A language model consists of a finite set $\mathcal{V}$, and a function $p(x_1, x_2, \ldots x_n)$ such that:*

1. *For any $\langle x_1 \ldots x_n \rangle \in \mathcal{V}^\dagger$, $p(x_1, x_2, \ldots x_n) \geq 0$*
2. *In addition,*
$$\sum_{\langle x_1 \ldots x_n \rangle \in \mathcal{V}^\dagger} p(x_1, x_2, \ldots x_n) = 1$$

Hence $p(x_1, x_2, \ldots x_n)$ is a probability distribution over the sentences in $\mathcal{V}^\dagger$.

As one example of a (very bad) method for learning a language model from a training corpus, consider the following. Define $c(x_1 \ldots x_n)$ to be the number of times that the sentence $x_1 \ldots x_n$ is seen in our training corpus, and N to be the total number of sentences in the training corpus. We could then define

$$p(x_1 \ldots x_n) = \frac{c(x_1 \ldots x_n)}{N}$$

This is, however, a very poor model: in particular it will assign probability 0 to any sentence not seen in the training corpus, which seems like a terrible idea.

At first glance the language modeling problem seems like a rather strange task, so why are we considering it? There are a couple of reasons:

1. Language models are very useful in a broad range of applications, the most obvious perhaps being speech recognition and machine translation. In many applications it is very useful to have a good "prior" distribution $p(x_1 \ldots x_n)$ over which sentences are or aren't probable in a language. For example, in speech recognition the language model is combined with an acoustic model that models the pronunciation of different words: one way to think about it is that the acoustic model generates a large number of candidate sentences, together with probabilities; the language model is then used to reorder these possibilities based on how likely they are to be a sentence in the language.

2. The techniques we describe for defining the function p, and for estimating the parameters of the resulting model from training examples, will be useful in several other contexts during the course: for example in hidden Markov models, which we will see next, and in models for natural language parsing.

We now turn to a critical question: given a training corpus, how do we learn the function p? We first describe *Markov models*, a central idea from probability theory; and then describe *trigram language models*, an important class of language models that build directly on ideas from Markov models.

2 Markov Models

2.1 Markov Models for Fixed-length Sequences

Consider a sequence of random variables, $X_1, X_2, \ldots, X_n$. Each random variable can take any value in a finite set $\mathcal{V}$. For now we will assume that the length of the sequence, n, is some fixed number (e.g., $n = 100$). In the next section we'll describe how to generalize the approach to cases where n is also a random variable, allowing different sequences to have different lengths.

Our goal is as follows: we would like to model the probability of any sequence $x_1 \ldots x_n$, where $n \geq 1$ and $x_i \in \mathcal{V}$ for $i = 1 \ldots n$, that is, to model the joint probability

$$P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n)$$

There are $|\mathcal{V}|^n$ possible sequences of the form $x_1 \ldots x_n$: so clearly, it is not feasible for reasonable values of $|\mathcal{V}|$ and n to simply list all $|\mathcal{V}|^n$ probabilities. We would like to build a much more compact model.

In a first-order Markov process, we make the following assumption, which considerably simplifies the model:

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}) \quad (1)$$

$$= P(X_1 = x_1) \prod_{i=2}^{n} P(X_i = x_i | X_{i-1} = x_{i-1}) \quad (2)$$

The first step, in Eq. 1, is exact: by the chain rule of probabilities, *any* distribution $P(X_1 = x_1 \ldots X_n = x_n)$ can be written in this form. So we have made no assumptions in this step of the derivation. However, the second step, in Eq. 2, is not necessarily exact: we have made the assumption that for any $i \in \{2 \ldots n\}$, for any $x_1 \ldots x_i$,

$$P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-1} = x_{i-1})$$

This is a (first-order) *Markov assumption*. We have assumed that the identity of the i'th word in the sequence depends only on the identity of the previous word, x_{i-1}. More formally, we have assumed that the value of X_i is conditionally independent of $X_1 \ldots X_{i-2}$, given the value for X_{i-1}.

In a second-order Markov process, which we will also refer to as a *trigram model*, we make a slightly weaker assumption, namely that each word depends on the previous *two* words in the sequence:

$$P(X_i = x_i | X_1 = x_1 \ldots X_{i-1} = x_{i-1}) = P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

It follows that the probability of an entire sequence is written as

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \quad (3)$$

For convenience, we will assume that $x_0 = x_{-1} = *$ in this definition, where * is a special "start" symbol in the sentence.

2.2 Markov Sequences for Variable-length Sentences

In the previous section, we assumed that the length of the sequence, n, was fixed. In many applications, however, the length n can itself vary. Thus n is itself a random variable. There are various ways of modeling this variability in length: in this section we describe the most common approach for language modeling.

The approach is simple: we will assume that the n'th word in the sequence, X_n, is always equal to a special symbol, the STOP symbol. This symbol can only appear at the end of a sequence. We use exactly the same assumptions as before: for example under a second-order Markov assumption, we have

$$P(X_1 = x_1, X_2 = x_2, \ldots X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) \quad (4)$$

for any $n \geq 1$, and for any $x_1 \ldots x_n$ such that $x_n =$ STOP, and $x_i \in \mathcal{V}$ for $i = 1 \ldots (n-1)$.

We have assumed a second-order Markov process where at each step we generate a symbol x_i from the distribution

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

where x_i can be a member of $\mathcal{V}$, or alternatively can be the STOP symbol. If we generate the STOP symbol, we finish the sequence. Otherwise, we generate the next symbol in the sequence.

A little more formally, the process that generates sentences would be as follows:

1. Initialize $i = 1$, and $x_0 = x_{-1} = *$

2. Generate x_i from the distribution

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

3. If x_i = STOP then return the sequence $x_1 \dots x_i$. Otherwise, set $i = i + 1$ and return to step 2.

3 Trigram Language Models

There are various ways of defining language models, but we'll focus on a particularly important example, the trigram language model, in this note. This will be a direct application of Markov models to the language modeling problem.

As in Markov models, we model each sentence as a sequence of n random variables, $X_1, X_2, \dots X_n$. The length, n, is itself a random variable (it can vary across different sentences). We always have X_n = STOP. Under a second-order Markov model, the probability of any sentence $x_1 \dots x_n$ is then

$$P(X_1 = x_1, X_2 = x_2, \dots X_n = x_n) = \prod_{i=1}^{n} P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1})$$

where we assume as before that $x_0 = x_{-1} = *$.

We will assume that for any i, for any x_{i-2}, x_{i-1}, x_i,

$$P(X_i = x_i | X_{i-2} = x_{i-2}, X_{i-1} = x_{i-1}) = q(x_i | x_{i-2}, x_{i-1})$$

where $q(w|u, v)$ for any (u, v, w) is a parameter of the model. We will soon see how to derive estimates of the $q(w|u, v)$ parameters from our training corpus. Our model then takes the form

$$p(x_1 \dots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1})$$

for any sequence $x_1 \dots x_n$.

This leads us to the following definition:

Definition 2 (Trigram Language Model) *A trigram language model consists of a finite set $\mathcal{V}$, and a parameter*

$$q(w|u, v)$$

for each trigram u, v, w such that $w \in \mathcal{V} \cup \{STOP\}$, and $u, v \in \mathcal{V} \cup \{*\}$. The value for $q(w|u, v)$ can be interpreted as the probability of seeing the word w immediately after the bigram (u, v). For any sentence $x_1 \dots x_n$ where $x_i \in \mathcal{V}$

for $i = 1 \ldots (n-1)$, and $x_n =$ STOP, the probability of the sentence under the trigram language model is

$$p(x_1 \ldots x_n) = \prod_{i=1}^{n} q(x_i | x_{i-2}, x_{i-1})$$

where we define $x_0 = x_{-1} = *$. $\square$

For example, for the sentence

the dog barks STOP

we would have

$$p(\text{the dog barks STOP}) = q(\text{the}|*, *) \times q(\text{dog}|*, \text{the}) \times q(\text{barks}|\text{the, dog}) \times q(\text{STOP}|\text{dog, barks})$$

Note that in this expression we have one term for each word in the sentence (*the, dog, barks*, and *STOP*). Each word depends only on the previous two words: this is the trigram assumption.

The trigram assumption is arguably quite strong, and linguistically naive (see the lecture slides for discussion). However, it leads to models that are very useful in practice. The key problem we are left with is to estimate the parameters of the model, namely

$$q(w|u, v)$$

where w can be any member of $\mathcal{V} \cup \{\text{STOP}\}$, and $u, v \in \mathcal{V} \cup \{*\}$. There are around $|\mathcal{V}|^3$ parameters in the model. This is likely to be a very large number. For example with $|\mathcal{V}| = 10,000$ (this is a realistic number, most likely quite small by modern standards), we have $|\mathcal{V}|^3 \approx 10^{12}$.

3.1 Maximum-Likelihood Estimates

We first start with the most generic solution to the estimation problem, the *maximum-likelihood estimates*. We will see that these estimates are flawed in a critical way, but we will then show how related estimates can be derived that work very well in practice.

First, some notation. Define $c(u, v, w)$ to be the number of times that the trigram (u, v, w) is seen in the training corpus: for example, $c(\text{the, dog, barks})$ is the number of times that the sequence of three words *the dog barks* is seen in the training corpus. Similarly, define $c(u, v)$ to be the number of times that the bigram (u, v) is seen in the corpus. For any w, u, v, we then define

$$q(w|u, v) = \frac{c(u, v, w)}{c(u, v)}$$

As an example, our estimate for $q(\text{barks}|\text{the, dog})$ would be

$$q(\text{barks}|\text{the, dog}) = \frac{c(\text{the, dog, barks})}{\text{c(the, dog)}}$$

This estimate is very natural: the numerator is the number of times the entire trigram *the dog barks* is seen, and the denominator is the number of times the bigram *the dog* is seen. We simply take the ratio of these two terms.

Unfortunately, this way of estimating parameters runs into a very serious issue. Recall that we have a very large number of parameters in our model (e.g., with a vocabulary size of $10,000$, we have around 10^{12} parameters). Because of this, many of our counts will be zero. This leads to two problems:

- Many of the above estimates will be $q(w|u,v) = 0$, due to the count in the numerator being 0. This will lead to many trigram probabilities being systematically underestimated: it seems unreasonable to assign probability 0 to any trigram not seen in training data, given that the number of parameters of the model is typically very large in comparison to the number of words in the training corpus.

- In cases where the denominator $c(u,v)$ is equal to zero, the estimate is not well defined.

We will shortly see how to come up with modified estimates that fix these problems. First, however, we discuss how to evaluate a language model.

3.2 Perplexity

So how do we measure the quality of a language model? A very common method is to evaluate the *perplexity* of the model on some held-out data.

The method is as follows. Assume that we have some test data sentences $s_1, s_2, \ldots s_m$. Each test sentence is again a sequence of words $x_1 \ldots x_n$. It is critical that the test sentences are "held out", in the sense that *they are not part of the corpus used to estimate the language model.* In this sense, they are examples of new, unseen sentences.

For any test sentence s_i, we can measure its probability $p(s_i)$ under the language model. A natural measure of the quality of the language model would be the probability it assigns to the entire set of test sentences, that is

$$\prod_{i=1}^{m} p(s_i)$$

The intuition is as follows: the higher this quantity is, the better the language model is at modeling unseen sentences.

The perplexity on the test corpus is derived as a direct transformation of this quantity. Define M to be the total number of words in the test corpus. Then the average log probability under the model is defined as

$$\frac{1}{M} \log \prod_{i=1}^{m} p(s_i) = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

This is just the log probability of the entire test corpus, divided by the total number of words in the test corpus. Again, the higher this quantity is, the better the language model.

The *perplexity* is then defined as

$$2^{-l}$$

where

$$l = \frac{1}{M} \sum_{i=1}^{m} \log p(s_i)$$

Thus we take the negative of the average log probability, and raise it to the power two (I'm assuming in this section that $\log$ is log base two). The perplexity is a positive number. The *smaller* the value of perplexity, the better the language model is at modeling unseen data.

Some intuition behind perplexity is as follows. Say we have a vocabulary $\mathcal{V}$, where $|\mathcal{V}| = N$, and the model predicts

$$q(w|u, v) = \frac{1}{N}$$

for all u, v, w. Thus this is the dumb model that simply predicts the uniform distribution over the vocabulary. In this case, it can be shown that the perplexity is equal to N. So *under a uniform probability model, the perplexity is equal to the vocabulary size*. Perplexity can be thought of as the effective vocabulary size under the model: if, for example, the perplexity of the model is 120 (even though the vocabulary size is say $10,000$), then this is roughly equivalent to having an effective vocabulary size of 120.

One useful fact about perplexity is the following. If for any trigram u, v, w seen in test data, we have the estimate

$$q(w|u, v) = 0$$

then the perplexity will be ∞. To see this, note that in this case the probability of the test corpus under the model will be 0, and the average log probability will be

$-\infty$. Thus if we take perplexity seriously as our measure of a language model, then we should avoid giving 0 estimates at all costs.

Finally, some intuition about "typical" values for perplexity. On newswire text, with a vocabulary size of say $20,000$, typical perplexity values for a trigram language model might be in the range 100-200. The value for a bigram model might be in the range 700-1000, and for a unigram model might be in the thousands (these numbers are a little rough—I'm recalling them from memory). The perplexity for a model that simply assigns probability $1/20,000$ to each word in the vocabulary would be $20,000$. So the trigram model clearly gives a big improvement over bigram and unigram models.

3.3 Linear Interpolation

We now describe a method for estimating the parameters of a trigram language model. We define the following *trigram*, *bigram*, and *unigram* maximum-likelihood estimates as

$$q_{ML}(w|u,v) = \frac{c(u,v,w)}{c(u,v)}$$

$$q_{ML}(w|v) = \frac{c(v,w)}{c(v)}$$

$$q_{ML}(w) = \frac{c(w)}{c()}$$

where we have extended our notation: $c(w)$ is the number of times word w is seen in the training corpus, and $c()$ is the total number of words seen in the training corpus.

The trigram, bigram, and unigram estimates have different strengths and weaknesses. The unigram estimate will never have the problem of its numerator or denominator being equal to 0: thus the estimate will always be well-defined, and will always be greater than 0 (providing that each word is seen at least once in the training corpus, which is a reasonable assumption). However, the unigram estimate completely ignores the context (previous two words), and hence discards very valuable information. In contrast, the trigram estimate does make use of context, but has the problem of many of its counts being 0. The bigram estimate falls between these two extremes.

The idea in linear interpolation is to use all three estimates, by defining the trigram estimate as follows:

$$q(w|u,v) = \lambda_1 \times q_{ML}(w|u,v) + \lambda_2 \times q_{ML}(w|v) + \lambda_3 \times q_{ML}(w)$$

Here λ_1, λ_2 and λ_3 are three additional parameters of the model, which satisfy

$$\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0$$

and

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Thus we take a weighted average of the three estimates.

There are various ways of estimating the λ values. A common one is as follows. Say we have some additional held-out data, which is separate from both our training and test corpora. We will call this data the *development data*. Define $c'(u, v, w)$ to be the number of times that the trigram (u, v, w) is seen in the development data. It is easy to show that the log-likelihood of the development data, as a function of the parameters $\lambda_1, \lambda_2, \lambda_3$, is

$$\begin{aligned} L(\lambda_1, \lambda_2, \lambda_3) &= \sum_{u,v,w} c'(u, v, w) \log q(w|u, v) \\ &= \sum_{u,v,w} c'(u, v, w) \log \left(\lambda_1 \times q_{ML}(w|u, v) + \lambda_2 \times q_{ML}(w|v) + \lambda_3 \times q_{ML}(w)\right) \end{aligned}$$

We would like to choose our λ values to make $L(\lambda_1, \lambda_2, \lambda_3)$ as high as possible. Thus the λ values are taken to be

$$\arg \max_{\lambda_1, \lambda_2, \lambda_3} L(\lambda_1, \lambda_2, \lambda_3)$$

such that

$$\lambda_1 \geq 0, \lambda_2 \geq 0, \lambda_3 \geq 0$$

and

$$\lambda_1 + \lambda_2 + \lambda_3 = 1$$

Finding the optimal values for $\lambda_1, \lambda_2, \lambda_3$ is fairly straightforward (see the lecture slides for one way of doing this).

3.4 Discounting Methods, and Katz Back-off

We now describe an alternative estimation method, which is commonly used in practice. Consider first a method for estimating a bigram language model, that is, our goal is to define

$$q(w|v)$$

for any $w \in \mathcal{V} \cup \{\text{STOP}\}$, $v \in \mathcal{V} \cup \{*\}$.

The first key step will be to define *discounted counts*. For example, with a discount value of 0.5, for any bigram $c(v,w)$ such that $c(v,w) > 0$, we define the discounted count as

$$c^*(v,w) = c(v,w) - 0.5$$

Thus we simply subtract a constant value, 0.5, from the count. This reflects the intuition that if we take counts from the training corpus, we will systematically over-estimate the probability of bigrams seen in the corpus (and under-estimate bigrams not seen in the corpus).

For any bigram (v,w) such that $c(v,w) > 0$, we can then define

$$q(w|v) = \frac{c^*(v,w)}{c(v)}$$

Thus we use the discounted count on the numerator, and the regular count on the denominator of this expression.

For any context v, this definition leads to some *missing mass*, defined as

$$\alpha(v) = 1 - \sum_w \frac{c^*(v,w)}{c(v)}$$

The intuition behind discounted methods is to divide this "missing mass" between the words w such that $c(v,w) = 0$.

More formally, the complete definition of the estimate is as follows. For any v, define the sets

$$\mathcal{A}(v) = \{w : c(v,w) > 0\}$$

and

$$\mathcal{B}(v) = \{w : c(v,w) = 0\}$$

Then the estimate is defined as

$$q_{BO}(w|v) = \begin{cases} \frac{c^*(v,w)}{c(v)} & \text{If } w \in \mathcal{A}(v) \\ \alpha(v) \times \frac{q_{ML}(w)}{\sum_{w \in \mathcal{B}(v)} q_{ML}(w)} & \text{If } w \in \mathcal{B}(v) \end{cases}$$

Thus if $c(v,w) > 0$ we return the estimate $c^*(v,w)/c(v)$; otherwise we divide the remaining probability mass $\alpha(v)$ in proportion to the unigram estimates $q_{ML}(w)$.

The method can be generalized to trigram language models in a natural, recursive way: for any bigram (u,v) define

$$\mathcal{A}(u,v) = \{w : c(u,v,w) > 0\}$$

and

$$\mathcal{B}(u,v) = \{w : c(u,v,w) = 0\}$$

Define $c^*(u, v, w)$ to be the discounted count for the trigram (u, v, w): for example,

$$c^*(u, v, w) = c(u, v, w) - 0.5$$

Then the trigram model is

$$q_{BO}(w|u, v) = \begin{cases} \frac{c^*(u,v,w)}{c(u,v)} & \text{If } w \in \mathcal{A}(u, v) \\ \alpha(u, v) \times \frac{q_{BO}(w|v)}{\sum_{w \in \mathcal{B}(u,v)} q_{BO}(w|v)} & \text{If } w \in \mathcal{B}(u, v) \end{cases}$$

where

$$\alpha(u, v) = 1 - \sum_{w \in \mathcal{A}(u,v)} \frac{c^*(u, v, w)}{c(u, v)}$$

is again the “missing” probability mass. Note that we have divided the missing probability mass in proportion to the bigram estimates $q_{BO}(w|v)$, which were defined previously.