STATISTICAL LANGUAGE MODELS

Francois Role - francois.role@parisdescartes.fr

September 2017

1 Language Models

1.1 N-gram Language Models

A **language model** is designed to measure how likely it is that a sequence would be uttered. This is useful in many settings. One are where language model are widely is speech recognition.

More precisely, a language model is a function that takes a word sequence and returns a probability that it would be uttered.

A good language model will assign a greater probability to the sequence "le ciel est bleu" than to the sequence "est le bleu ciel".

In theory, given a sequence of words $W = w_1, w_2, \cdots, w_n$ its probability can be estimated as :

$$p(w_1, w_2, \cdots, w_n) = p(w_1)p(w_2|w_1), \cdots, p(w_n|w_1, w_2, \cdots, w_{n-1}) \tag{1}$$

$$= \prod_{i=1}^{n} p(w_i|w_1^{i-1}) \tag{2}$$

In practice, because of **data sparsity**, the history of previous words has to be limited to m words using the **Markov assumption** that considering only a small number of previous m words is enough to determine the probability of the next words. So, we have :

$$p(w_1, w_2, \cdots, w_n) = p(w_1)p(w_2|w_1), \cdots, p(w_n|w_{n-m}, \cdots, w_{n-1}) \tag{3}$$

$$= \prod_{i=1}^{n} p(w_i|w_{i-m}^{i-1}) \tag{4}$$

A model based on a two-word history is called a **trigram model**, where statistics are collected over sequences of three words called **trigrams**: a two-word history is used to predict a third word. A model based on a one-word history is a **bigram model**. If a model relies on single words the langage model is a **unigram model**.

1.2 Estimating the conditional probabilities

N-gram probabilities are estimated by dividing the observed frequency of a particular sequence by the observed frequency of a prefix of this sequence. The ratio between thes frequency counts is called a **relative frequency**.

Here is an example of how to estimate a conditional probability for a trigram model:

$$p(w_3|w_1,w_2) = \frac{count(w_1,w_2,w_3)}{\sum_w count(w_1,w_2,w)} \tag{5}$$
$$= \frac{count(w_1,w_2,w_3)}{count(w_1,w_2)} \tag{6}$$

More generally, the formula for estimating N-gram probabilities (that is, using an history of $m = N-1 words$ is:

$$p(w_i|w_{i-N+1}^{i-1}) = \frac{count(w_{i-N+1}^{i})}{count(w_{i-N+1}^{i-1})}$$

However, basing the estimations on raw frequency counts has severe limitations.

Suppose we have to estimate the probability of the sequence A B C D E using a bigram language model and the corpus from which the model was built did not include the sequence B C. So our estimation for $p(C|B)$ was $\frac{count(B,C)}{\sum_w count(B,w)} = \frac{0}{\sum_w count(B,w)} = O$ and so the predicted probability for A B C D E will be 0.

More generally, all unseen n-grams will be assigned a probability of 0 which is a very hard decision. We want to be able to generalize in presence of unseen data.

1.3 Smoothing Techniques

Add-one (Laplace) smoothing To avoid zero counts and generalize better, we pretend that we saw each word one more time than we did.

The estimation is now:

$$p(w_i|w_{i-1}) = \frac{count(w_{i-1},w_i)+1}{\sum_w [1+count(w_{i-1},w)]} = \frac{count(w_{i-1},w_i)+1}{|V| + \sum_w (count(w_{i-1},w)}$$

where V is the vocabulary.

The limit of this additive smoothing is that it treats unseen bigrams involving frequent words and unseen bigrams involving rare words on the same footing.

More Advanced Smoothing Techniques Kneser-Ney and Good-Turing smoothing are more complex methods that allow to estimate the count of things we've never seen in a more principled way.

Exercise

(a) According to a bigram model trained on the following sentences (and without using any smmoothing technique):
`I saw the boy the girl is in the garden the boy is funny the girl is funny the baby is smiling the girl is smiling`

1. What is the probability $p(funny|is)$?
2. What is the most probable word after the sequence "the boy is" ?
3. Rank the following sequences based on their probabilities ? `I saw the similing` `the bird I saw` `the baby is funny`
4. What is the probabilty of `I saw the similing`, this time using Add-one smoothing.

1.4 Measuring the Quality of a Language Model

Task-based Evaluation To compare two models A and B, a wa y is to assess (e.g. accuracy) how each performs on a given task. I requires designing, setting up and running a task. A cheaper way is to compute some quality measures directly from the model.

Cross-entropy and Perplexity Let L be a language. Using a training set T drawn from L, we estimate the unknown probability distribution p associated with L. This results in an estimated probability distribution q. To sum up p is the true (unknown) distribution of the words in L and q is the distribution of words estimated from the training corpus.

To evaluate q we assess how well it predicts (what probability it gives) to a separate test sample $x_1, x_2, \cdots, x_n$ also drawn from L. The better q the higher probabilities it should assign to the test events.

The perplexity of the model q is defined to be:

$$2^{-\sum_{i=1}^{n} \frac{1}{N} \log_2 q(x_i)}$$

The better q, the higher probabilities it should assign to the test events, so the lower perplexity. How is this quantity derived? We will show it in the case of a bigram model.

Let T be a (improperly named) **test set**, that is a sequence of tokens $w_1, w_2, \cdots, w_n$,

and let q be a probability distribution trained from a training set. For a bigram model, the likelihood of the test set T according to q is:

$$L = \prod_{i=1}^{n} q(w_i|w_{i-1}) \tag{7}$$

We want to minimize the opposite of the log-likelihood$\sum_{i=1}^{n} - \log q(w_i|w_{i-1})$, or, equivalently, the mean of this value :

$$\frac{1}{n}\sum_{i=1}^{n} - \log q(w_i|w_{i-1}) = \frac{1}{n}\left(\sum_{i=1}^{n} \log \frac{1}{q(w_i|w_{i-1}}\right) \tag{8}$$

This quantity is precisely the exponent in the perplexity formula. It also turns out that it equals the geometric mean of the inverses of the probabilities) is:

$$\log \sqrt[n]{\prod_{i=1}^{n} \frac{1}{q(w_i|w_{i-1})}} = \log \left(\prod \frac{1}{q(w_i|w_{i-1})} \right)^{1/n} \tag{9}$$

$$= \frac{1}{n} \sum_{i=1}^{n} \log \frac{1}{q(w_i|w_{i-1})} \tag{10}$$

$$= \frac{1}{n} \sum_{i=1}^{n} - \log q(w_i|w_{i-1}) \tag{11}$$

So, the exponent $-\sum_{i=1}^{n} \frac{1}{N} \log_2 q(x_i)$ found in the perplexity formula is the log of the geometric mean of the **word perplexities** (inverse predicted probabilities for the words). This exponent $-\sum_{i=1}^{n} \frac{1}{N} \log_2 q(x_i)$ is in fact an approximation of the cross-entropy of a (long-enough) sequence $x_1, \cdots, x_n$ according to the Shannon-McMillan-Breiman theorem. This is why some say that "the cross-entropy is the logarithm of perplexity".

To summarize, the perplexity of a sequence \$x_1, $\cdots$, $x\_n\$ can be computed as$:

$$2^{-\sum_{i=1}^{n} \frac{1}{N} \log_2 q(x_i)} = 2^{\log \sqrt[n]{\prod_{i=1}^{n} \frac{1}{q(w_i|w_{i-1})}}} = \sqrt[n]{\prod_{i=1}^{n} \frac{1}{q(w_i|w_{i-1})}}$$

Exercise

(b)
Using the model of execise **(a)**, compute the perplexity of the sequence:
`I saw the baby`

Aside Note:

The fact that the cross-entropy is the logarithm of perplexity can also be demonstrated in the context of logistic regression.

For one instance and K classes the cross-entropy error is:

$$-\sum_{k=1}^{K} t_k \log y_k = -t_k^* \log y_k^* = -\log y_k^*$$

where y_k^* is the probability computed by the model for the correct word.

Perplexity is the inverse predicted probability for the correct word:

$$\frac{1}{y_k^*}$$

Since $-\log y_k^* = \log \frac{1}{y_k^*}$ the cross-entropy is the logarithm of perplexity.

Minimizing the arithmetic mean of the cross-entropy is equivalent to minimizing the log of the geometric mean of the perplexity.

$$\frac{(-\log y_1*) + \cdots + (-\log y_n*)}{n} =$$

$$\frac{\log \frac{1}{y_1*} + \ldots + \log \frac{1}{y_n^*}}{n}$$

$$\frac{\log(\frac{1}{y_1*} \times \ldots \times \frac{1}{y_n^*})}{n}$$

$$\frac{1}{n}\log(\frac{1}{y_1*} \times \ldots \times \frac{1}{y_n^*})$$

$$\log \sqrt[n]{\frac{1}{y_1*} \times \ldots \times \frac{1}{y_n^*}}$$

2 Statistical Measures of Word Association

** Positive Pointwise Mutual Information** (PMI) has been initially designed to help find interesting **collocations** in a corpus.

For two word x and y, PMI is defined as:

$$pmi(x,y) = \log \frac{p(x,y)}{p(x)p(y)}$$

and is estimated as:

$$\hat{pmi}(x,y) = \frac{count(x,y)N}{count(x)count(y)}$$

where N is the total number of tokens in the training corpus, and $count(x,y)$, $count(x,y)$, $count(x,y)$ are the number of occurrences of the bigram xy, of the word x and of the word y resp.

PMI has several shortcomings:

- Its values are not bounded: they range from negative to positive infinity.
- Negative PMI values are difficult to interpret.
- it is biased towards bigrams involving rare words.

These problems are partially dealt with as follows. First, to get rid of negative values, one can use a variant called **Positive PMI** (PPMI) instead of PMI. PPMI is defined as:

$$PPMI(x,y) = max(\log \frac{p(x,y)}{p(x)p(y)}, 0)$$

To overcome the bias problem, one can either raise the dividend $p(x,y)$ to a power greater than 0 (typically 2 or 3). This has the effect of boosting frequent pairs. Alternatively, one can raise the terms in th denominator to a power slightly below 1 (typically 0.75). This will increase the probability assigned to rare words compared to frequent words, and thus diminish the PMI values of pairs involving rare words.

Today, PMI is not only used to spot exact collocations but, more generally to discover pairs of words tending to occur within a **window** of n words of each other. The counts are then retrieved from a **word-word matrix** where the cell i,j represents the number of times word i occurs in a window of $\pm n$ words around word j.

Today, PMI is widely applied in NLP to produce semantic representations for words and documents. Recently, connections have been made between PMI and **word embeddings**.

Other techniques, inspired from the hypothesis testing field, are also used. Given two words w and y the null hypothesis is that the two words re independent.

$$t - test(x, y) = \frac{p(x, y) - p(x)p(y)}{\sqrt{p(x)p(y)}}$$

Exercise

(a) Given the following message recently received from the $Z\mu125\alpha$ planet
'aaa bbb aaa ccc eee ddd eee fff aaa eee fff eee aaa ccc fff eee bbb aaa

1. Compute $pmi(aaa, ccc)$
2. (bonus question) What is the meaning of the message?