

Python Course: Lecture 15

February 7, 2006

1 Multiple Information Sources

- There appear to be multiple information sources available for use in understanding language.
- For example, if we are listening to someone talk to us, all of the following factors might impinge on our understanding:
 - The words they use
 - The pitch of their voice, word and intra-word pause durations ¹
 - The syntactic structure of their utterance—e.g. it might be helpful to know that a phrase like “the boy” occurring at the beginning of a sentence is likely the subject.
 - Our knowledge of the semantic content of the conversation—e.g. I’ll attach a different significance to the word “bear” depending on whether we’re discussing a camping trip or the stock market.
 - Our knowledge of our interlocutor’s interests, idiosyncratic speech patterns, and current emotional state.
 - Our knowledge of how the world works in general.
- Some of these information sources (e.g. human emotion, general world knowledge) are beyond the current ability of computers to model. Others (e.g. words, syntax, prosody) we have a fighting chance of incorporating into a natural language application.
- Therefore we would like a mathematically rigorous method for combining multiple information sources.

¹Linguists use the word *prosody* to refer to this general set of acoustic parameters.

2 Mixture Models

- Say we have two probability distributions $p_1(x)$ and $p_2(x)$ defined over the same support.
- We could combine them using *linear interpolation*. The resulting probability distribution is called a *mixture model*.

$$p_{MIX}(x; \lambda_1) = \lambda_1 p_1(x) + \lambda_2 p_2(x) \quad (1)$$

Here λ_1 and λ_2 are constants called *mixture weights* used to weight the relative importance of the two probability distributions. If we want the combination P_{MIX} to itself be a probability distribution, we must add the following constraints on the mixture weights.²

$$\lambda_1 + \lambda_2 = 1 \quad (2)$$

$$0 \leq \lambda_i \leq 1 \quad (3)$$

- For example, say we have two birdwatching probability estimates. We could combine them like so.

$$p_1(\text{robin}) = 0.23 \quad (4)$$

$$p_2(\text{robin}) = 0.38 \quad (5)$$

$$\lambda_1 = 0.7 \quad (6)$$

$$\lambda_2 = 0.3 \quad (7)$$

$$p_{MIX}(\text{robin}; \lambda_1 = 0.7) = 0.7(0.23) + 0.3(0.38) \quad (8)$$

$$= 0.275 \quad (9)$$

The mixture probability for seeing a robin lies somewhere between the probabilities given by the individual components, though it's closer to p_1 's because that component's mixture weight λ_1 is larger.

- This can be generalized from 2 to m mixture components.

$$p_{MIX}(x; \vec{\lambda}) = \sum_{i=1}^m \lambda_i p_i(x) \quad (10)$$

$$\vec{\lambda} = \langle \lambda_1, \lambda_2 \dots \lambda_{m-1} \rangle \quad (11)$$

$$\sum_{i=1}^m \lambda_i = 1 \quad (12)$$

$$0 \leq \lambda_i \leq 1 \quad (13)$$

²i.e. The mixture weights must themselves define a probability distribution.

- Mixture models are properly normalized. To see this, sum over all possible values of x .

$$\sum_{x \in X} p_{MIX}(x; \vec{\lambda}) = \sum_{x \in X} \sum_{i=1}^m \lambda_i p_i(x) \quad (14)$$

$$= \sum_{i=1}^m \lambda_i \sum_{x \in X} p_i(x) \quad (15)$$

$$= \sum_{i=1}^m \lambda_i \quad (16)$$

$$= 1 \quad (17)$$

Step (15) follows from the fact that the mixture weights are not functions of x . Step (16) follows from the fact that each individual mixture component is a probability distribution. Step (17) follows from the normalization constraint on the mixture weights imposed in (12).

- The values returned by the mixture model will depend on the choice of mixture weights, represented in (10) by the vector $\vec{\lambda}$. The elements of this vector are extra *degrees of freedom* that we have incorporated into our model.
- Because of the normalization constraint in (12), there are $m - 1$ degrees of freedom in a mixture model with m components. That is because after choosing $m - 1$ mixture weight values, the last one must just be whatever is leftover to make all of them sum to one.

$$\sum_{i=1}^m \lambda_i = 1 \quad (18)$$

$$\lambda_m + \sum_{i=1}^{m-1} \lambda_i = 1 \quad (19)$$

$$\lambda_m = 1 - \sum_{i=1}^{m-1} \lambda_i \quad (20)$$

3 N-gram Mixture Models

- We can apply mixture model techniques to N-gram models.
- Say we have a set of N-gram language models $p_i(w|h)$ which estimate the probability of seeing a word w given a history h of all the previously seen words. We can combine them just as in (10).

$$p_{MIX}(w|h; \vec{\lambda}) = \sum_{i=1}^m \lambda_i p_i(w|h) \quad (21)$$

With the same normalization constraints on the λ_i mixture weights as before.

- These different language models could be anything. For example, they could be trained on different corpora, or utilize different smoothing techniques.
- A very common sort of N-gram mixture model is one in which all the components are trained on the same data with the same smoothing technique, but are of different orders. For example, we might combine a 3-gram, 2-gram, and unigram model like so.

$$p_{BO}(w_n|h; \vec{\lambda}) = \lambda_3 p_3(w_n|w_{n-2}^{n-1}) + \lambda_2 p_2(w_n|w_{n-1}) + \lambda_1 p_1(w_n) \quad (22)$$

This is called a *backoff* language model because you can think of it as getting information from the trigram context, then backing off to the bigram, then backing off to the unigram.

- In n-gram modeling we always have to make a tradeoff between the richer linguistic structure captured by high order models and the data sparsity issues we encounter when training them. Backoff models are a way of trying to get the best of both worlds.
- We can get benefits, but only at the cost of adding the additional degrees of freedom (and therefore additional complexity) to our language model.
- For these models to be usable, we need some principled way to choose $\vec{\lambda}$.
- At a high level, we want to choose the $\vec{\lambda}$ that works “best”. But in order to do this we need to define an evaluation metric for language models.

4 Evaluation Metric

- The general strategy of a language model is to have some training corpus act as a proxy for a language in general. Then samples of text will be judged to be representatives of that language to the extent that they resemble the training corpus.
- Say we have some body of test data that we know to be reasonable English. A good language model will give that data a high probability. A bad language model will give that data a low probability.
- But since longer test samples will always have a lower probability, we need some way of normalizing by length. The simplest method is to divide by the number of words in the sample.
- For a given string of test tokens $w_1 \dots w_N$ and model p we will look at the following evaluation metric.

$$H(w_1 \dots w_N; p) = -\frac{1}{N} \log_2 p(w_1 \dots w_N) \quad (23)$$

The log function is there because it’s easier to work with log probabilities. The negative sign is there because the log of a number between 0 and 1 is negative.

- An equivalent evaluation metric is called the *perplexity* of the model.

$$h(w_1 \dots w_N; p) = 2^{\frac{1}{N} \log_2 p(w_1 \dots w_N)} \quad (24)$$

$$= 2^{\log_2 p(w_1 \dots w_N)^{\frac{1}{N}}} \quad (25)$$

$$= p(w_1 \dots w_N)^{\frac{1}{N}} \quad (26)$$

Because (24) doesn't add a negative sign, a smaller perplexities means you have a better language model.

- Numbers like perplexity that measure the fitness between a model and some sample of data are called *figures of merit*.
- The term “perplexity” is a useful mnemonic. The higher your model's perplexity, the greater degree to which is it surprised by (= “assigns a low probability to” = “is perplexed by”) your test data. So if we know *a priori* that the test data is a reasonable representative sample of language, higher perplexity means a worse language model.

5 Test, Training, and Development Data

- Armed with a figure of merit, we can sketch out a strategy for choosing the mixture weights $\vec{\lambda}$. Namely, we pick some sample of language that we know *a priori* to be reasonable and then choose the particular combination of λ_i values that assigns the highest log probability ³ to that sample.
- This is easier said than done.
- In general choosing these values may be a mathematically difficult problem that requires sophisticated iterative numerical optimization techniques.
- The details of these techniques are beyond the scope of this class. (In homeworks we'll just try a few random guesses.)
- I will just mention, however, that one of the most common optimization techniques for this kind of problem is called *expectation maximization*.
- There are a few guidelines for handling your data independent of the particular mathematical details.
- Any machine learning problem (of which N-gram modeling is an example) has at least two separate bodies of data: *training* data and *test* data.
- Training data is what you use to train your model, e.g. what provides the raw N-gram counts. It is a proxy for all your *a priori* knowledge about the language you are modeling.

³Or, equivalently, the lowest perplexity.

- Test data is what you use to evaluate your model. It is a proxy for all novel linguistic constructions you want to be able to handle.
- Since the point of language modeling is extrapolate from the known to the unknown, there must be no overlap between the training and test sets.
- Testing on your training data is a kind of cheating. Your model will always perform well on the training data. This tells you nothing about its ability to extrapolate to unknown data.
- Sometimes we have a third set of data called the *development* or *tuning* data. We use this data when debugging our implementation of the model, or doing optimization techniques to determine model parameters.
- The tuning data is a proxy for unseen samples of training data, so for the same reasons of keeping yourself honest, it must also be separate from the training and test data.