

Kevin Knight's Exercises for IBM models

Jean Mark Gawron
Linguistics 582
San Diego State University
gawron@mail.sdsu.edu
<http://www.rohan.sdsu.edu/~gawron>

2007 Sep 27

1. Assume f and e are not independent. That is, assume:

$$P(e, f) = P(e) * P(f | e)$$

Complete the following:

$$P(e, f) = P(f) * ?$$

2. The following equation express something called Bayes Rule. It's important.

$$P(e | f) = \frac{P(e) * P(f | e)}{P(f)} \quad (1)$$

Prove this, using the results from the previous exercise.

3. Using Bayes Rule, we can rewrite the expression for the most likely translation:

$$\operatorname{argmax}_e P(e | f) = \operatorname{argmax}_e \underbrace{P(e)}_{\text{language model}} * \underbrace{P(f | e)}_{\text{translation model}} \quad (2)$$

Read

$$\operatorname{argmax}_e P(e | f)$$

As the english word e that maximizes $P(e | f)$. How did we get from

$$\operatorname{argmax}_e P(e | f) = \operatorname{argmax}_e \frac{P(e) * P(f | e)}{P(f)}$$

to (2)? In other words, what happened to $P(f)$?

4. To represent the addition of integers from 1 to n , we write:

$$\sum_{i=1}^n i = 1 + 2 + 3 + \dots + n \quad (3)$$

For the product of integers from 1 to n , we write:

$$\prod_{i=1}^n i = 1 * 2 * 3 * \dots * n \quad (4)$$

If there's a factor inside a summation that does not depend on what's being summed over, it can be taken outside:

$$\sum_{i=1}^n ki = k1 + k2 + k3 + \dots + kn = k \sum_{i=1}^n i \quad (5)$$

Question:

$$\prod_{i=1}^n i * k = ?$$

5. One idea suggested by separating the model $P(f, e)$ into $P(f | e)$ (translation model) and $P(e)$ (language model) is that we can do translation word by word via $P(f | e)$, and let $P(e)$ figure out how to arrange results into a proper-sounding English string. So $P(f | e)$ knows about translation, but it doesn't know anything more about English; in particular it doesn't know about English word order. This leads to the notion of **Bag generation**. The translation model produces a *bag of words*. We use the language model to order these in to a proper-sounding English string. To illustrate, put the following sets of words in canonical order

- (a) { have, programming, a, seen, never, I, language, better }
- (b) { actual, the, hashing, is, since, not, collision,-free, usually, the, is, less, perfectly, the, of, somewhat, capacity, table }
- (c) { loves, John, Mary }

The last exercise is hard. It seems like $P(f | e)$ needs to know something about word order after all. It can't simply suggest a bag of English words and be done with it. But, maybe it only needs to know a little bit about word order, not everything.

What kind of knowledge are you applying here? Do you think a machine could do this job? Can you think of a way to automatically test how well a machine is doing, without a lot of human checking?

6. Language model question

Which has a higher probability: "I like snakes" or "I hate snakes"?
Type both in to www.altavista.com or google and find out (include quotation marks in your query to ensure that the words appear together and in order in the retrieved documents).

7. Check the answer to the last example against the English language model in our local MT system:

```
ngram -debug 1 -order 5 -ppl test -lm /opt/corpora/mt/wmt07/lm/europarl.lm
```

Where test is the name of a file containing:

```
I hate snakes
I like snakes
```

Comment on the output. What does "OOV" mean? What causes that to happen?

- 8. What is the probability of: "I like snakes that are not poisonous"? [Use google or altavista, compare to ngram]
- 9. How do you know if one model "works better" than another? One way to pit language models against each other is to gather up a bunch of previously unseen English test data, then ask: What is the probability of a certain model

(generative story plus particular parameter values), given the test data that we observe? We can write this symbolically as:

$$P(\text{model} \mid \text{test-data})$$

Using Bayes rule:

$$P(\text{model} \mid \text{test-data}) = \frac{P(\text{model}) * P(\text{test-data} \mid \text{model})}{P(\text{data})}$$

Let's suppose that $P(\text{model})$ is the same for all models. That is, without looking at the test data, we have no idea whether "0.95" is a better number than "0.07" deep inside some model. Then, the best model is the one that maximizes $P(\text{test-data} \mid \text{model})$.

What happened to $P(\text{data})$?

10. Perplexity

If the test data is very long, then an n-gram model will assign a $P(e)$ value that is the product of many small numbers, each less than one. Some n-gram conditional probabilities may be very small themselves. So $P(e)$ will be tiny and the numbers will be hard to read. A more common way to compare language models is to compute

$$-\frac{\log_2 P(e)}{N}$$

which is called the **perplexity of a model**. N is the number of words in the test data. Dividing by N helps to normalize things, so that a given model will have roughly the same perplexity no matter how big the test set is. The logarithm is base two.

As $P(e)$ increases, perplexity decreases. A good model will have a relatively large $P(e)$ and a relatively small perplexity. The lower the perplexity, the better.

Suppose a language model assigns the following conditional n-gram probabilities to a 3-word test set: $1/4$, $1/2$, $1/4$. Then

$$P(\text{test-set}) = \frac{1}{4} * \frac{1}{2} * \frac{1}{4} = 0.03125.$$

What is the perplexity?

11. Suppose another language model assigns the following conditional n-gram probabilities to a different 6-word test set: 1/4, 1/2, 1/4, 1/4, 1/2, 1/4. What is its P(test-set)? What is its perplexity?
12. If $P(\text{test-set}) = 0$, what is the perplexity of the model?
Why do you think it is called “perplexity”?
13. Another problem with $P(e)$ is that tiny numbers will easily underflow any floating point scheme. Suppose $P(e)$ is the product of many factors $f_1, f_2, f_3 \dots f_n$, each of which is less than one. Then there is a trick:

$$\log P(e) = \log(f_1 * f_2 * f_3 * \dots * f_n) \quad (6)$$

$$= \log(f_1) + \log(f_2) + \log(f_3) + \dots + \log(f_n) \quad (7)$$

If we store and manipulate the log versions of probabilities, then we will avoid underflow. Instead of multiplying two probabilities together, we add the log versions. You can get used to log probabilities with this table:

p	$\log p$
0.0	-infinity
0.1	-3.32
0.2	-2.32
0.3	-1.74
0.4	-1.32
0.5	-1.00
0.6	-0.74
0.7	-0.51
0.8	-0.32
0.9	-0.15
1.0	-0.00

For example,

$$\begin{aligned} \log(0.6 * 0.4) &= \log 0.6 + \log 0.4 \\ &= -0.74 + -1.32 \\ &= -1.05 \end{aligned}$$

(a) What is

$$\log(0.5 * 0.5 * 0.5 * 0.5) = \log 0.5^4$$

(b) What is

$$\log(\underbrace{0.5 * 0.5 * 0.5 * 0.5 * \dots * 0.5}_n) = \log 0.5^n$$

14. Start with the sentence “my child, you must go home” and transform it into a foreign-language translation using the IBM Model 3 generative story (use the version that includes NULL).
15. If the maximum fertility for English words were 2, how long is the longest French sentence that can be generated from an English sentence of length l_e ? (Don’t forget about NULL-generated words). [Note: the answer is NOT infinity. In fact, as part of your answer, first explain why it’s not infinity. Hint: Look at what the probability model on slide ?? says.]
16. Take 10,000 English sentences and rewrite them into Spanish, storing all intermediate strings. No, make that a million English sentences! Ha, ha, just kidding. Don’t do this exercise.