Chapter 3

Text Clustering

3.1 Introduction

Let's go back to the naïve Bayes model and consider the case where some or all of the classes k_i are unobserved. For example, if we have 1,000 e-mails labeled as spam or ham, and 1 million more unlabeled e-mails, maybe we can learn a better classifier by combining both labeled and unlabeled data. Maybe from the labeled data we learn that "award" often implies spam, and from the unlabeled data we learn that "award" often occurs in the same documents as "deposit," then we can infer that "deposit," too, is associated with spam.

If all of the k_i are unobserved, then there's no way that the computer can learn to label emails as spam or ham. The best that we can do is tell the computer that there are two kinds of emails, and it should try to find some way of classifying them into class 1 and class 2. Then, just maybe, class 1 will be ham and class 2 will be spam, or class 1 will be spam and class 2 will be ham. Then again, maybe the computer will find some other possibly meaningful way of classifying the emails instead. We can also choose more than two classes and see what happens. This is known as text *clustering*. Some of the most interesting NLP problems are unsupervised learning problems, although unsupervised learning also often fails.

3.2 Expectation-Maximization: "hard" version

Assume that all of the k_i are unobserved. We have to choose a number of classes in advance. Then we want to maximize the likelihood, that is, the probability of the observed data. Since we only observe words without classes, the likelihood looks different from before:

$$L = \prod_i P(d_i) \tag{3.1}$$

$$= \prod_i \sum_k p(k) \prod_{w \in d_i} p(w \mid k). \tag{3.2}$$

The bad news is this does not have a closed-form solution, and we have to use an iterative method. While we could use a generic method like gradient ascent, it's far more common to use a specialized method called *expectation-maximization* (Dempster, Laird, and Rubin, 1977).

We start with the "hard" (as opposed to "soft") version. The basic idea is pretty intuitive. If we have labels, then we know how train the model (this is just like training in the supervised case). And if we have

a model, we know how to predict the labels (this is just like testing in the supervised case). So, we can start by initializing the model parameters to random values; then guess labels for all the documents, then use those labels to retrain the model, and repeat.

initialize parameters $p(k)$ and $p(w \mid k)$ randomly
repeat
  for each i **do**
    predict $k_i^* = \arg\max_k P(k \mid d_i)$
  end for
  estimate the parameters from the d_i and k_i^*
until done

It seems that one of three things could happen: nothing, or the model will get worse and worse as noise takes over, or somehow the model will get better.

If you consider the partially supervised case (where some of the documents are labeled and some are not labeled), it's easier to see how it might actually get better. Suppose, as in the example from the beginning, that we've seen emails labeled as spam that contain the word "award" but never the word "deposit." But we've seen unlabeled emails containing both words; when we guess the label for those emails, they're likely to be labeled as spam. If they are, then when we retrain the model, we'll learn that "deposit" is mildly associated with spam, which is good.

In the fully unsupervised case, it's a little harder to see, but let's work out a simple example.

```
award notification
enron canada
enron america
award payment
```

Initialize the model randomly:

$$p(1) = 0.5 \qquad p(2) = 0.5$$
$$p(\texttt{america} \mid 1) = 0.1 \qquad p(\texttt{america} \mid 2) = 0.2$$
$$p(\texttt{award} \mid 1) = 0.1 \qquad p(\texttt{award} \mid 2) = 0.1$$
$$p(\texttt{canada} \mid 1) = 0.1 \qquad p(\texttt{canada} \mid 2) = 0.2$$
$$p(\texttt{enron} \mid 1) = 0.2 \qquad p(\texttt{enron} \mid 2) = 0.2$$
$$p(\texttt{notification} \mid 1) = 0.4 \qquad p(\texttt{notification} \mid 2) = 0.2$$
$$p(\texttt{payment} \mid 1) = 0.1 \qquad p(\texttt{payment} \mid 2) = 0.1$$

Guess a class for each example:

	$P(1,d)$	$P(2,d)$	k^*
`award notification`	0.02	0.01	1
`enron canada`	0.01	0.02	2
`enron america`	0.01	0.02	2
`award payment`	0.005	0.005	1 (arbitrary)

For the last document, there was a tie, which we broke arbitrarily. Retrain the model:

$$p(1) = 0.5 \qquad p(2) = 0.5$$
$$p(\texttt{america} \mid 1) = 0 \qquad p(\texttt{america} \mid 2) = 0.25$$
$$p(\texttt{award} \mid 1) = 0.5 \qquad p(\texttt{award} \mid 2) = 0$$
$$p(\texttt{canada} \mid 1) = 0 \qquad p(\texttt{canada} \mid 2) = 0.25$$
$$p(\texttt{enron} \mid 1) = 0 \qquad p(\texttt{enron} \mid 2) = 0.5$$
$$p(\texttt{notification} \mid 1) = 0.25 \qquad p(\texttt{notification} \mid 2) = 0$$
$$p(\texttt{payment} \mid 1) = 0.25 \qquad p(\texttt{payment} \mid 2) = 0$$

Guess a class for each example:

	$P(1,d)$	$P(2,d)$	k^*
`award notification`	0.0625	0	1
`enron canada`	0	0.0625	2
`enron america`	0	0.0625	2
`award payment`	0.0625	0	1

And it's not hard to see that nothing changes after this. The algorithm clustered the documents by noticing that two documents had the word "award" in common and two documents had the word "enron" in common.

But what if we had broken that tie in the other direction? Retrain the model:

$$p(1) = 0.25 \qquad p(2) = 0.75$$
$$p(\texttt{america} \mid 1) = 0 \qquad p(\texttt{america} \mid 2) = 0.167$$
$$p(\texttt{award} \mid 1) = 0.5 \qquad p(\texttt{award} \mid 2) = 0.167$$
$$p(\texttt{canada} \mid 1) = 0 \qquad p(\texttt{canada} \mid 2) = 0.167$$
$$p(\texttt{enron} \mid 1) = 0 \qquad p(\texttt{enron} \mid 2) = 0.333$$
$$p(\texttt{notification} \mid 1) = 0.5 \qquad p(\texttt{notification} \mid 2) = 0$$
$$p(\texttt{payment} \mid 1) = 0 \qquad p(\texttt{payment} \mid 2) = 0.167$$

Guess a class for each example:

	$P(d,1)$	$P(d,2)$	k^*
`award notification`	0.0625	0	1
`enron canada`	0	0.0417	2
`enron america`	0	0.0417	2
`award payment`	0	0.0208	2

So hard EM is able to discover some things, but is also kind of brittle.

3.3 Expectation-Maximization: real version

The real version of EM has a slight difference that makes it possible to actually prove that the likelihood (3.1) gets better, or is at a local maximum.

Before, we used the model to predict the best label for each document, by using the model's best guess for each. But its guesses aren't certain. For a document d, it only gives probabilities $P(1 \mid d)$ and $P(2 \mid d)$. So, it's more reasonable to say that we observed d a fraction of a time in class 1, and a fraction of a time in class 2.

We do this for all the documents and add up all the counts to get *expected counts*, which may be fractional. Then, just like before, we train the model from those counts. It doesn't matter that the counts are fractional; we just count and divide like before.

Let's try this with our example, with the same random initialization as before. Instead of guessing the best class for each document, we compute the distribution over classes:

	$P(1, d)$	$P(2, d)$	$P(1 \mid d)$	$P(2 \mid d)$
`award notification`	0.02	0.01	0.667	0.333
`enron canada`	0.01	0.02	0.333	0.667
`enron america`	0.01	0.02	0.333	0.667
`award payment`	0.005	0.005	0.5	0.5

Note that we didn't have to do any arbitrary tie-breaking. Retrain the model:

$$p(1) = 0.458 \qquad p(2) = 0.542$$
$$p(\texttt{america} \mid 1) = 0.0909 \qquad p(\texttt{america} \mid 2) = 0.154$$
$$p(\texttt{award} \mid 1) = 0.318 \qquad p(\texttt{award} \mid 2) = 0.192$$
$$p(\texttt{canada} \mid 1) = 0.0909 \qquad p(\texttt{canada} \mid 2) = 0.154$$
$$p(\texttt{enron} \mid 1) = 0.182 \qquad p(\texttt{enron} \mid 2) = 0.308$$
$$p(\texttt{notification} \mid 1) = 0.182 \qquad p(\texttt{notification} \mid 2) = 0.0768$$
$$p(\texttt{payment} \mid 1) = 0.136 \qquad p(\texttt{payment} \mid 2) = 0.115$$

And so on. The computations are tedious to do by hand, but hopefully you can see where this is going—eventually the words "award", "notification", and "payment" will dominate class 1, and the words "enron", "america", and "canada" will dominate class 2.

The algorithm looks like this:

initialize parameters $p(k)$ and $p(w \mid k)$ randomly
repeat
  ▷ E step:
  for each i, k **do**
    compute $P(k \mid d_i) = \dfrac{P(k, d_i)}{\sum_{k'} P(k, d_i)}$
    $E[c(k)] \leftarrow E[c(k)] + P(k \mid d_i)$
    for each $w \in d_i$ **do**
      $E[c(k, w)] \leftarrow E[c(k, w)] + P(k \mid d_i) \cdot c(w \in d_i)$
    end for
  end for
  ▷ M step:
  $p(k) = \dfrac{E[c(k)]}{n}$
  $p(w \mid k) = \dfrac{E[c(k, w)]}{E[c(k)]}$
until done

This version is guaranteed to make L increase or, if at a local maximum, stay the same. See Section 3.6 for why.

In the initialization step, it's very important to set the parameters randomly.

Question 8. What happens if we initialize the parameters uniformly?

Question 9. What happens if the number of classes is greater than or equal to the number of documents?

3.4 Details

Number of iterations Theoretically we should keep performing iterations of EM until the likelihood stops increasing. In practice, it's extremely common just to set a fixed number of iterations.

Random restarts Because EM is only guaranteed to converge to a local maximum, if we run it again we might get a different result. So to try to get the best result, we might run EM many times and choose the one with the best likelihood.

Smoothing In the supervised case, we used add-one or add-δ smoothing. We can use it here too, during the M step. However, the likelihood is no longer guaranteed to increase at every iteration; it could decrease. There's another objective function that *is* guaranteed to increase, but it's not so common to explicitly compute it.

3.5 Experiment

We ran the unsupervised naïve Bayes clustering algorithm on the blog data from before, using two classes and 30 iterations.

Whatever the algorithm learns, it isn't gender:

class	male	female	total
0	921	933	1854
1	436	292	728

In both classes, the most frequent words are just the most frequent words in English; to try to find the most distinctive words, we ranked words by the formula $p(w \mid k)/(c(w)+10)$. (The $+10$ is in there so that we don't have the opposite problem of the very rarest words floating to the top.)

class	words
0	`upset personally closer excited laughing silly absolutely enjoying cry honest`
1	`founded member buried decades underneath builder luggage attended northern aspects`

It certainly seems as though the clusterer has learned to group the blog posts into more personal posts (class 0) and more objective posts (class 1), although looking at the actual posts with their classes, one doesn't get this impression.

When we try add-one smoothing, a very different result emerges: a mere four documents in class 0, and the rest in class 1. Those four documents are written in: English with weird characters, Malay/Indonesian, Portuguese, and French.

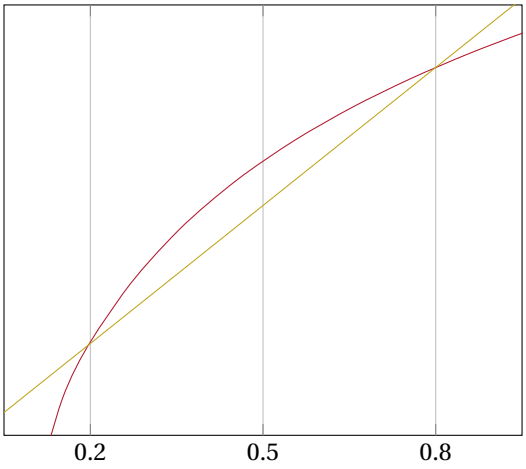

Figure 3.1: Jensen's inequality: the log of the average is greater than the average of the logs.

3.6 Optional: Proof of Correctness of EM

The goal of EM is to find the parameter values that maximize the likelihood,

$$\log L = \sum_i \log P(d_i) \tag{3.3}$$

$$= \sum_i \log \sum_k P(k, d_i) \tag{3.4}$$

$$= \sum_i \log \sum_k \frac{Q_i(k)}{Q_i(k)} P(k, d_i) \tag{3.5}$$

$$\geq \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)}. \tag{3.6}$$

The last step follows from *Jensen's inequality*, which is pictured in Figure 3.1. So let

$$F = \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \leq \log L. \tag{3.7}$$

Now $\log L$ and F are functions of the parameters, $p(k)$ and $p(w \mid k)$, and the auxiliary distributions, Q_i. We can either hold the parameters constant and optimize the Q_i, or we can hold the Q_i constant and optimize the parameters.

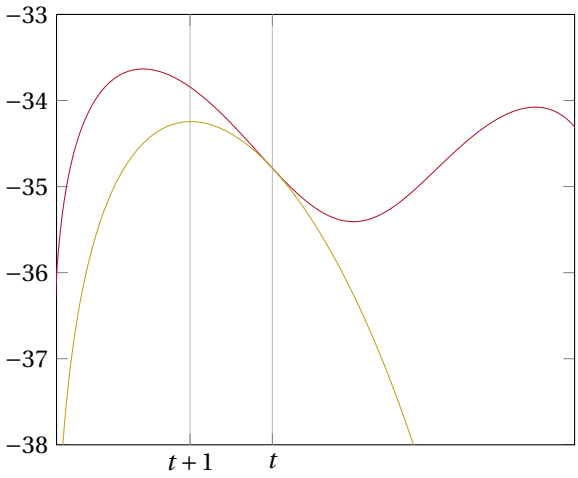

Figure 3.2: The EM algorithm tries to maximize the log-likelihood (upper curve) with respect to the model parameters. In the E step, we compute a lower bound (lower curve) and maximize it, which is guaranteed to improve (but not maximize) the log-likelihood.

E step Hold the parameters constant and optimize the Q_i. Above, we showed using Jensen's inequality that F is less than or equal to $\log L$. How much less?

$$\log L - F = \sum_i \log P(d_i) - \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \tag{3.8}$$

$$= \sum_i \sum_k Q_i(k) \left(\log P(d_i) - \log \frac{P(k, d_i)}{Q_i(k)} \right) \tag{3.9}$$

$$= \sum_i \sum_k Q_i(k) \log \frac{Q_i(k)}{P(k \mid d_i)}, \tag{3.10}$$

which is known as the *Kullback-Leibler divergence* of $P(k \mid d_i)$ from $Q_i(k)$, a measure of distance between two distributions. It is always nonnegative and is zero if and only if the two distributions are equal. So we set

$$Q_i(k) = P(k \mid d_i), \tag{3.11}$$

which makes $F = \log L$. So F is not only a lower bound of L, but it now touches L at the current parameter values (see Figure 3.2). This means that if we can adjust the parameters to improve F, we are guaranteed to improve L also.

M step Hold the Q_i constant and optimize the parameters. We can rewrite the lower bound (3.7) as

$$F = \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \tag{3.12}$$

$$= \sum_i \sum_k Q_i(k) \log P(k, d_i) - \sum_i \sum_k Q_i(k) \log Q_i(k) \tag{3.13}$$

$$= \sum_i \sum_k Q_i(k) \log P(k, d_i) + \sum_i H[Q_i(k)]. \tag{3.14}$$

The first term is just the likelihood of the observed data as though, for each document d_i, we had observed d_i with class k, $Q_i(k)$ times. (The second term is the *entropy* of the Q_i, and doesn't concern us here except that it's independent of the parameters.) So we do maximum likelihood estimation using the $Q_i(k)$ to hallucinate the classes. That is, just count and divide. As argued above, if we can improve F, then we will improve the likelihood L as well. Further work would be needed to show that if F is already at a local maximum, then L is also at a local maximum.

References

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *Journal of the Royal Statistical Society, Series B* 39.1, pp. 1–38.