Chapter 12

Bags of Words

We normally view text as a sequence of words, or we can impose additional structure on it like syntax. Or, we can dissolve some structure, namely, the order of the words. This gives a *bag of words*, which simply counts how many times each word occurs in a sentence (or document).

$$\text{the cat sat on the mat} \rightarrow \{\text{cat}, \text{mat}, \text{on}, \text{sat}, \text{the}, \text{the}\}$$

Why? First, this representation is computationally extremely easy to work with. Second, this representation melts a text down into bits of meaning and serves as a crude way of capturing what the text is "about." Crude, but often effective.

Tokenization We assume that all the sentences/documents have been tokenized so that the word boundaries are unambiguous. A commonly used English tokenizer is part of Stanford CoreNLP,[1] and LDC has a simple rule-based tokenizer.[2] Both are implemented/wrapped in Python by NLTK.[3]

Stemming It is also common in bag-of-word approaches to do morphological *stemming*, that is, removing affixes like *-ed*, *-ing*, etc. A classic stemmer for English is the Porter stemmer,[4] and another is the Lancaster (Paice/Husk) stemmer.[5] Both are implemented/wrapped in Python by NLTK.[6]

Stop Words Finally, it's also common to remove high-frequency but low-content words, like (Manning, Raghavan, and Schütze, 2008, p. 26):

a an and are as at be by for from has he in
is it its of on that the to was were will with

[1] `http://nlp.stanford.edu/software/corenlp.shtml`
[2] `http://www.cis.upenn.edu/~treebank/tokenizer.sed`
[3] `http://www.nltk.org/api/nltk.tokenize.html`
[4] `http://tartarus.org/martin/PorterStemmer/`
[5] `http://www.comp.lancs.ac.uk/computing/research/stemming/index.htm`
[6] `http://www.nltk.org/api/nltk.stem.html`

12.1 Classification

Suppose we have a collection of documents in different classes, and we want to learn to classify new documents. For example:

- We have a collection of e-mails that are labeled either as "spam" or "ham" and we want to learn to classify new e-mails.
- We have some texts whose author is known (e.g., Alexander Hamilton, James Madison, and John Jay) and want to classify anonymous texts (the Federalist Papers).
- We have a database of product reviews together with star ratings, and we want to be able to predict star ratings based on reviews alone.
- We have samples of (transcribed) speech from children diagnosed as autistic or not autistic, and we want to automatically diagnose other children using their speech.
- We have tweets that are known to be in various languages and want to automatically identify the language of new tweets.

Question 13. What are some other possible applications?

We'll continue working with the example of spam filtering, but just remember that this is only one of many possible applications. This is a fairly easy problem, but occasionally difficult even for humans. For example:

```
From: fas@nsf.gov
Subject: Payment
To: chiang@isi.edu

DFM has approved your requested payment and has asked the U.S. Treasury to
issue a payment to you within the next 4 working days. This payment is
being sent directly to the Bank or Financial Institution identified by
you for this purpose. If you are an NSF employee, this payment is being
sent directly to the bank/financial institution where your bi-weekly pay
is being deposited.

This payment for 560.00 is 1099 reportable if it totals $600 or more
for the year (i.e. You will receive an IRS 1099-Misc form from
NSF)P131318.

Head, Accounts Payable Section.
-------------------------------
```

More formally, we are given documents $d_1, \ldots, d_n$ together with their correct classes $k_1, \ldots, k_n$. We want to learn a model $P(k \mid d)$, where k is a class and d is a document, and given a new document d, we want to be able to find, with high accuracy,

$$k^* = \arg\max_k P(k \mid d). \tag{12.1}$$

12.1.1 Naïve Bayes

Question 14. What problem would you run into if you used the model

$$P(k \mid d) = \frac{c(k,d)}{c(d)}?$$

What might you do to repair the model?

This route turns out to be difficult (though we will return to it later when we do logistic regression). Instead of thinking about how to classify a document, let's think about how the document came to be. First, someone decided to write an e-mail; he was either a spammer or a "hammer." Then, this person authored a document. More formally:

$$\begin{aligned} \arg\max_k P(k \mid d) &= \arg\max_k P(d)P(k \mid d) && (12.2)\\ &= \arg\max_k P(k,d) && (12.3)\\ &= \arg\max_k P(k)P(d \mid k). && (12.4) \end{aligned}$$

The advantage of thinking about it this way is that it is much easier to write down a model for $P(d \mid k)$ than for $P(k \mid d)$.

We'll consider just the simplest, which is called a *naïve Bayes* classifier and looks like this:

$$P(k,d) = p(k) \prod_{w \in d} p(w \mid k), \tag{12.5}$$

where the $p(k)$ and $p(w \mid k)$ are the parameters of the model. (Let's adopt the convention that P(something) and p(something) both denote the probability of something, but P(something) might be defined in terms of other probabilities, whereas p(something) is a parameter of the model that needs to be estimated.)

Naïve Bayes is called naïve because it naïvely assumes that all the words in a document are independent of each other. All it captures is that spammers are more likely to use certain words and hammers are more likely to use certain words.

It is possible to count other things besides words. For example, if we're trying to classify a document as English or French, then the presence of an é is a pretty good sign that it's in French. So character probabilities like $p(\text{é} \mid k)$ might be useful in addition to word probabilities. Or, if we're trying to predict the positivity or negativity of a product review, then the occurrence of the word *bad* might be good sign of a negative review, unless it is immediately preceded by the word *not*. So probabilities of *bigrams* like $p(\text{not bad} \mid k)$ are important. Note that if we do this, however, the distribution $P(k,d)$ no longer sums to one, because there's no such thing as a document that contains the words {dog} but the characters {c, a, t}. So, if we only sum over feasible documents d, then $P(k,d)$ sums to less than one, that is, it is *deficient*, which is in general not considered a good thing. This is even more naïve, but might work okay in practice.

Question 15. What are some other features that might be helpful?

12.1.2 Training

Training the model, or estimating the parameters $p(k)$ and $p(w \mid k)$, is easy. It's just:

$$\begin{aligned} p(k) &= \frac{c(k)}{\sum_k c(k)} \\ p(w \mid k) &= \frac{c(k,w)}{\sum_{w'} c(k,w')}. \end{aligned} \tag{12.6}$$

This is known as *relative frequency estimation* (or "count and divide"). It's so intuitive that it may seem odd to give it a name, but it's worth dwelling a bit on why we estimate probabilities this way.

Consider just $p(k)$. Suppose we have just two theories, or *models*, about how prevalent spam is. Model 1, or m_1, says that 10% of e-mails are spam, whereas Model 2, or m_2, says that 90% of e-mails are spam.

$$p(\text{spam} \mid m_1) = \tfrac{1}{10} \qquad p(\text{spam} \mid m_2) = \tfrac{9}{10}$$
$$p(\text{ham} \mid m_1) = \tfrac{9}{10} \qquad p(\text{ham} \mid m_2) = \tfrac{1}{10}$$

Before looking at the data, suppose that we think these two models are equally valid.

$$p(m_1) = \tfrac{1}{2} \qquad p(m_2) = \tfrac{1}{2}$$

Then, suppose we look at some data and see that, out of 10 e-mails, 7 are spam and 3 are ham. Now we want to know which model is better, m_1 or m_2?

$$\begin{aligned} P(m_1 \mid \text{data}) &= \frac{p(m_1)P(\text{data} \mid m_1)}{P(\text{data})} \\ &= \frac{\frac{1}{2}\left(\frac{1}{10}\right)^7\left(\frac{9}{10}\right)^3}{P(\text{data})} \\ &= \frac{9^3}{2 \cdot 10^{10} \cdot P(\text{data})} \end{aligned} \qquad \begin{aligned} P(m_2 \mid \text{data}) &= \frac{p(m_2)P(\text{data} \mid m_2)}{P(\text{data})} \\ &= \frac{\frac{1}{2}\left(\frac{9}{10}\right)^7\left(\frac{1}{10}\right)^3}{P(\text{data})} \\ &= \frac{9^7}{2 \cdot 10^{10} \cdot P(\text{data})} \end{aligned}$$

Since $P(m_2 \mid \text{data})$ is bigger, we can conclude that m_2 is the better model.

Now suppose that we compare not two models, but an infinite number of models,

$$P(\text{spam} \mid \theta) = \theta$$
$$P(\text{ham} \mid \theta) = 1 - \theta$$

for all $\theta \in [0, 1]$. The same reasoning still holds: we want to find the model that maximizes

$$\underbrace{P(\theta \mid \text{data})}_{\text{posterior}} = \frac{\overbrace{P(\theta)}^{\text{prior}}\overbrace{P(\text{data} \mid \theta)}^{\text{likelihood}}}{\underbrace{P(\text{data})}_{\text{evidence}}}. \tag{12.7}$$

Note that the denominator is independent of θ, and if we assume that the prior $P(\theta)$ is uniform, then the only factor that matters is the likelihood. We therefore call this *maximum likelihood estimation*.

Going back to the full naïve Bayes model: we want to maximize the likelihood, which is

$$L = \prod_i P(k_i, d_i) \tag{12.8}$$
$$= \prod_i p(k_i) \prod_{w \in d_i} p(w \mid k_i). \tag{12.9}$$

And this maximization problem has a closed-form solution, namely (12.6). See Section 12.1.5 for more details on how to derive that.

12.1.3 Classification

Once we've trained the model, finding the most probable class of a new document is also easy:

$$k^* = \arg\max_k P(k \mid d) \tag{12.10}$$

$$= \arg\max_k P(k, d) \tag{12.11}$$

$$= \arg\max_k p(k) \prod_{w \in d} p(w \mid k). \tag{12.12}$$

12.1.4 Smoothing

What happens if we encounter a word we've never seen before? Then we would have $p(w \mid k) = 0$, and therefore $P(k, d) = 0$, for all k. The presence of a single unknown word zeros out the whole document probability and makes it impossible to choose a class.

The standard solution is *smoothing*. We'll talk about the simplest smoothing method now, and more advanced smoothing methods later (when we talk about language models). In *add-one smoothing*, invented by Laplace in the 18th century, we add one to the count of every event, including unseen events. Thus we would estimate $p(w \mid k)$ as:

$$p(w \mid k) = \frac{c(k, w) + 1}{\sum_{w'} c(k, w) + |V|} \tag{12.13}$$

where $|V|$ is the size of the vocabulary, including unseen words. But what if we don't know how many unseen words there are? A typical (though not entirely correct) thing to do is to set $|V|$ to the number of seen word types, plus one for a generic unseen word.

We can also add any value $\delta > 0$ to the counts instead of one:

$$p(w \mid k) = \frac{c(k, w) + \delta}{\sum_{w'} c(k, w') + |V|\delta}. \tag{12.14}$$

The increment that we add (whether 1 or δ) is known as a *pseudocount*. Typically, δ would be set by trial and error. If you have a good reason to, you can also add a different pseudocount to different word types.

These methods are not hacks; they can all be thought of as different choices of the prior probability that we saw in Equation (12.7).

Question 16. If our training data is:

ham	spam	spam
`please pass on to your groups`	`we deliver to your door within 24 hours`	`please update your account details with citibank`

and we train with add-one smoothing, what are $P(\text{spam} \mid d)$ and $P(\text{ham} \mid d)$ for the document below?

```
please forward to
your groups
```

12.1.5 Experiment

Let's tackle a more difficult classification task than spam filtering: Given a blog post, can you predict whether the author is male or female? A dataset is provided by Mukherjee and Liu (2010), consisting of 3227 posts.[7]

We split the data into two parts, *training* and *test data*. We train the model on the training data, then test the model on the test data by classifying the documents and measuring the percent accuracy. We could have also set aside a third part, called *development data*, as a proxy for the test data while we are fiddling with the model. This way, we don't artificially inflate our score on the test data by lucky modifications. Here, we used 80% for training data and 20% for testing. Here's the first male-authored blog post:

> long time no see . like always i was rewriting it from scratch a couple of times . but nevertheless it's still java and now it uses metropolis sampling to help that poor path tracing converge . btw . i did mlt on yesterday evening after 2 beers (it had to be ballmer peak[8]) . altough the implementation is still very fresh it easily outperforms standard path tracing , what is to be seen especially when difficult caustics are involved . i've implemented spectral rendering too , it was very easy actually , cause all computations on wavelengths are linear just like rgb . but then i realised that even if it does feel more physically correct to do so , whats the point ? 3d applications are operating in rgb color space , and because i cant represent a rgb color as spectrum interchangeably i have to approximate it , so as long as i'm not running a physical simulation or something i don't see the benefits (please correct me if i'm wrong) , thus i abandoned that .

And the first female-authored blog post:

> liam is nothing like natalie . natalie never went through draws or cabinets . she was always so good . liam is a bit more adventurous . and he always picks the cabinets that are the hardest to put back together . hubby just started nutrisystem and he has his own little section of the kitchen . well , liam doesn't like the order of it all and trashes my set up at least 10 times a day . yes , i need to buy one of those locks for the doors . we have one for the chemicals . but none on the food . my mistake for sure ! in the meantime , i better document it because before i know it this boy will be in college and i will be missing cleaning up after him . yeah . . that's not me . but i still decided that i wanted to start doing yoga . i've been thinking about it for quite some time . so , this weekend i bought a pair of work out shorts and a tank and off i went . the only class that was close to me was birkham yoga , the hot one . the 100 degree sweaty room one . i . almost . died . the yoga itself was wonderful . i will continue that for sure . but the heat of that room was unbearable . the best part is , the teacher specifically came up to me and said ,

When we train the naïve Bayes classifier on the training data (with add-one smoothing) and test it on the test data, we get an accuracy of 65.1%. Note that we could have gotten 50% accuracy simply by guessing randomly. So our performance is better than random, but not all that much better.

What happens if we count both words and bigrams (pairs of consecutive words)? When counting bigrams, we add a fake word <s> at the beginning and a fake word </s> at the end. This gives an accuracy of 66.5%.

[7] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html

[8] http://xkcd.com/323/

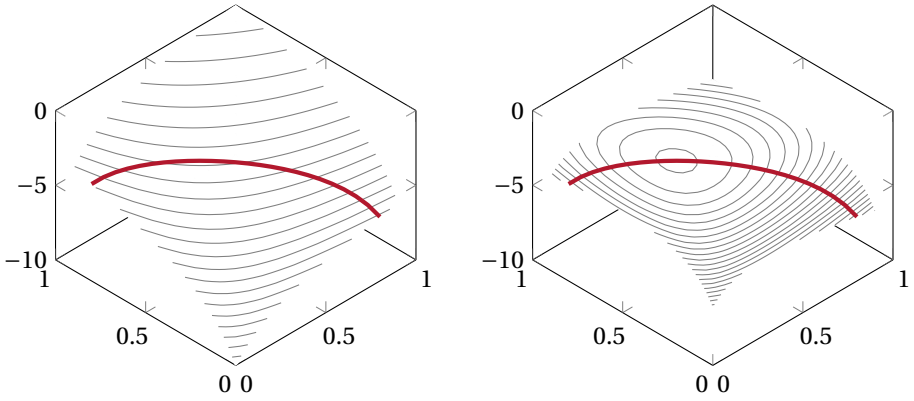

Figure 12.1: Relative frequency estimation. (a) We need to maximize the likelihood subject to the sum-to-one constraint (red line). (b) There is a value of λ that makes the Lagrangian have a maximum that does satisfy the sum-to-one constraint (red line).

Optional: Deriving relative frequency estimation

Here's how to derive the relative frequency estimate, taking $p(k)$ as an example. We want to maximize the likelihood L, or, equivalently, the log-likelihood

$$\log L = \sum_k c(k) \log p(k) \tag{12.15}$$

subject to the constraint

$$\sum_k p(k) = 1. \tag{12.16}$$

We can't just set the partial derivatives to zero because, although the surface defined by $\log L$ will be flat at its maximum along the line defined by (12.16), it will in general be sloped in the direction perpendicular to the line (see Figure 12.1). So we create a new function $\mathscr{L}$, called the *Lagrangian*, that can level the surface out:

$$\mathscr{L} = \sum_k c(k) \log p(k) - \lambda \left(\sum_k p(k) - 1 \right). \tag{12.17}$$

This is our original objective function plus a new term. This term is zero along the line defined by (12.16), and sloped in the direction perpendicular to the line, with a slope determined by the *Lagrange multiplier*, λ. At the maximum we are seeking, there is some value of λ that makes all the partial derivatives with respect to the parameters $p(k)$ zero.

The partial derivatives of $\mathscr{L}$ are:

$$\frac{\partial \mathscr{L}}{\partial p(k)} = \frac{c(k)}{p(k)} - \lambda, \tag{12.18}$$

$$\frac{\partial \mathscr{L}}{\partial \lambda} = -\left(\sum_k p(k) - 1 \right). \tag{12.19}$$

Note that the partial derivative with respect to λ is zero just in case the constraint (12.16) is satisfied. So the maximum we are seeking is the point where all the partial derivatives are zero. If we set them to zero and solve for the $p(k)$, we get:

$$p(k) = \frac{c(k)}{\sum_k c(k)}. \tag{12.20}$$

Further work (!) would be needed to verify that this is in fact the global maximum.

12.2 Clustering

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.

12.2.1 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:

$$L = \prod_i P(d_i) \tag{12.21}$$

$$= \prod_i \sum_k p(k) \prod_{w \in d_i} p(w \mid k). \tag{12.22}$$

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.

12.2.2 Expectation-Maximization: real version

The real version of EM has a slight difference that makes it possible to actually prove that the likelihood (12.21) 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:
  $p(k) = \dfrac{E[c(k)]}{\sum_{k'} E[c(k')]}$
  $p(w \mid k) = \dfrac{E[c(k, w)]}{\sum_{w'} E[c(k, w')]}$
until done

This version is guaranteed to make L increase or, if at a local maximum, stay the same. See Section 12.2.5 for why.

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

Question 17. What happens if we initialize the parameters uniformly?

Question 18. What happens if the number of classes is greater than or equal to the number of documents?

12.2.3 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 (real) 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.

12.2.4 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.

12.2.5 Optional: Proof of Correctness of EM

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

$$\log L = \sum_i \log P(d_i) \tag{12.23}$$

$$= \sum_i \log \sum_k P(k, d_i) \tag{12.24}$$

$$= \sum_i \log \sum_k \frac{Q_i(k)}{Q_i(k)} P(k, d_i) \tag{12.25}$$

$$\geq \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)}. \tag{12.26}$$

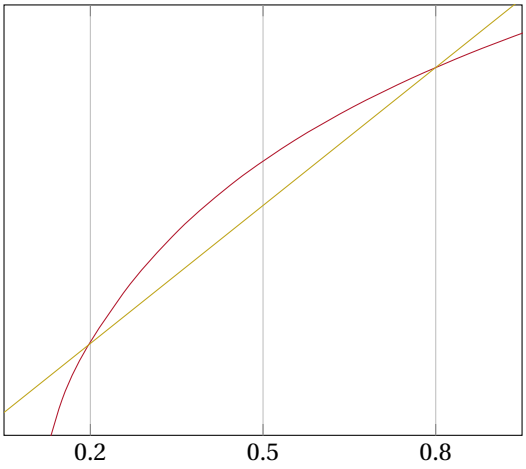

Figure 12.2: Jensen's inequality: the log of the average is greater than the average of the logs.

The last step follows from *Jensen's inequality*, which is pictured in Figure 12.2. So let

$$F = \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \leq \log L. \tag{12.27}$$

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.

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?

$$\log L - F = \sum_i \log P(d_i) - \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \tag{12.28}$$

$$= \sum_i \sum_k Q_i(k) \left(\log P(d_i) - \log \frac{P(k, d_i)}{Q_i(k)} \right) \tag{12.29}$$

$$= \sum_i \sum_k Q_i(k) \log \frac{Q_i(k)}{P(k \mid d_i)}, \tag{12.30}$$

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

$$Q_i(k) = P(k \mid d_i), \tag{12.31}$$

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 12.3). This means that if we can adjust the parameters to improve F, we are guaranteed to improve L also.

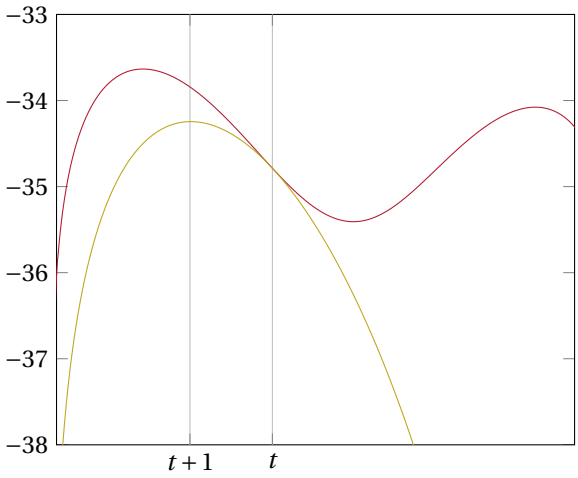

Figure 12.3: The EM algorithm tries to maximize the log-likelihood (upper curve) with respect to the model parameters. In the E step, we compute a lower bound (lower curve) and maximize it, which is guaranteed to improve (but not maximize) the log-likelihood.

M step Hold the Q_i constant and optimize the parameters. We can rewrite the lower bound (12.27) as

$$F = \sum_i \sum_k Q_i(k) \log \frac{P(k, d_i)}{Q_i(k)} \tag{12.32}$$

$$= \sum_i \sum_k Q_i(k) \log P(k, d_i) - \sum_i \sum_k Q_i(k) \log Q_i(k) \tag{12.33}$$

$$= \sum_i \sum_k Q_i(k) \log P(k, d_i) + \sum_i H[Q_i(k)]. \tag{12.34}$$

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.

12.3 Topic Modeling

Counting words can be a pretty good way of quantifying what a document is about, but not perfect. Suppose that we are trying to categorize documents into different genres. If a document contains the word "laser" or "robot" or other similar words, that is a pretty good sign that it's an engineering article or a science-fiction story. Suppose further that many of these technology-related words just occur one time – say, "photon" occurs only in an engineering article and "ramjet" appears only in a science-fiction story. That wouldn't be very good evidence that documents mentioning "photon" are engineering and documents mentioning "ramjet" are science-fiction. It would be more reasonable for a model to learn that

these are all technology-related words, and that technology-related words are indicative of both engineering and science fiction. That is, all these words belong to the *topic* of technology, and both engineering and science fiction use words in the technology topic. How might we learn such topics automatically? It turns out that these topics aren't all that helpful for classification; nevertheless, topics are interesting in their own right.

12.3.1 Naïve Bayes with topics

Let's modify the NBC as follows:

$$P(d) = p(k) \prod_{w \in d} \sum_{t} p(t \mid k) p(w \mid t). \tag{12.35}$$

What's new is the introduction of the variable t (topic), which is drawn from a finite set of predetermined size. We are given training data consisting of documents d_i and classes k_i, but we are not given the topics of the words in the d_i. The topics are called *hidden* variables.

Training

The class distribution $p(k)$ is estimated by counting and dividing:

$$p(k) = \frac{c(k)}{\sum_k c(k)}. \tag{12.36}$$

The rest of the model is trained using EM. In the E step, we compute, for each document d_i and word $w \in d_i$,

$$P(t \mid k_i, w) = \frac{p(t \mid k_i) p(w \mid t)}{\sum_{t'} p(t' \mid k_i) p(w \mid t')}. \tag{12.37}$$

Then, for each topic t,

$$E[c(k_i, t)] \leftarrow E[c(k_i, t)] + P(t \mid k_i, w) \tag{12.38}$$

$$E[c(t, w)] \leftarrow E[c(t, w)] + P(t \mid k_i, w). \tag{12.39}$$

The M step is just count and divide as usual:

$$p(t \mid k) = \frac{E[c(k, t)]}{\sum_{t'} E[c(k, t')]} \tag{12.40}$$

$$p(w \mid t) = \frac{E[c(t, w)]}{\sum_{t'} E[c(t', w)]}. \tag{12.41}$$

Experiment

We ran this model on the blog dataset, using 2 topics, 100 iterations, and add-10^{-6} smoothing (add-1 didn't work at all), and got an accuracy of 61.6%, which is no improvement over the original model (65.1%).

But what is more interesting is the topics learned by the model.

t	$P(t \mid \text{m})$	$P(t \mid \text{f})$	words
0	0.04	0.05	write ever together friends days recently took
1	0.03	0.06	sweet sugar wonderful loved dog n child
2	0.04	0.06	weekend wait favorite can't women pretty took
3	0.01	0.09	amelia prints lunch lovely en wedding raw
4	0.10	0.00	gay java users vacuum copyright eu lardo
5	0.06	0.03	war companies version engine box form number
6	0.00	0.14	sudan darfur vous slaw etsy dans deedee
7	0.00	0.12	une darfur garlic deedee etsy harriet sudan
8	0.05	0.04	seems ice power paper front thanks later
9	0.12	0.00	topps não linecook avrdude drillard liverpool ironpython
10	0.05	0.04	topics ways thats they've computer they're slowly
11	0.06	0.03	clear trend low question added themselves easily
12	0.05	0.04	quickly cut 2008 enough level care story
13	0.05	0.04	play search each sounds which goal include
14	0.13	0.00	drillard python ironpython topps wifi google's balloons
15	0.05	0.04	statement below less technical starting wrong these
16	0.03	0.07	herself sun dog coffee kids images tea
17	0.04	0.05	exhausted home until girl me positive house
18	0.07	0.02	stage film reference states system telephone google
19	0.00	0.10	le sauce du ladies à lunch et

Some of the topics are fairly coherent, and you can see that some topics are heavily favored by male bloggers and some topics are favored by female bloggers. In the next section, we'll leave text classification behind and focus our attention on discovering topics.

12.3.2 Probabilistic LSA

We can think of a document represented as a bag of words as a vector, where each component of the vector is a different word type. In many applications, this vector representation is useful in its own right. For example, we can compute the similarity between two documents using the cosine of the angle between the two document vectors.

As argued above, however, words might be too fine-grained a way of quantifying what a document is about. So what if we use vectors where each component of the vector is a different *topic*? This is the idea behind *probabilistic latent semantic analysis* (PLSA) or *probabilistic latent semantic indexing* (PLSI) (Hofmann, 1999). The only difference between PLSA and the NBC with topics is that in PLSA, every document is in its own class. Then, the probabilities $p(t \mid k_i)$ can be thought of as a vector representation of document d_i.

Training PLSA is identical to above. But after training a PLSA model, there isn't any inference step. The end product of PLSA training is the parameters themselves. We ran PLSA (20 topics, 100 iterations, add-10^{-6} smoothing). We used the same half-baked formula as before to find the most interesting words in each topic, and they were:

class	words
0	deedee kohl eric fatty adhd earthquake spud peppa beetner jeans
1	woke tomorrow mom bed snow birthday ride went sleep randa
2	garden seed gardening gardeners lambing burpee seeds sheep jacob katahdin
3	slaw não é uma para reservation um o nsobject ser
4	mangalitsa quiz nutrition lardo kotak skin vegan wax band’s organizational
5	balloons baking rice butter flour sugar psi tiramisu cream
6	drillard kombu reform republicans democrats congress sen obama directx mona
7	les linecook le une dans à vous et pas
8	divorce weight charter overweight bimal haw exercises webkit dating muscles
9	darfur sudan sudanese rebels khartoum jungle al-nur rebel marra barrel
10	printmaking prints ds yourself stable jailbird teaching sasha math forgive
11	liverpool dreamer conversations benitez gage arc reds horror arsenal liverpool’s
12	harriet { } geographers amelia font-size plone zope jsf hvar
13	# avrdude avr reddit pololu vikki orangutan hex kindle avr-gcc
14	arruda habenula ponzi brasilia heap dsn tile gems hop dominion
15	ironpython mobile microsoft lumber user marketing developers web users applications
16	baseball topps tournament mcgriff sticker fred knitting sox stickers dodgers
17	sous-vide choir low-temperature choral low-temp temperature mp3 albums album pokey
18	n ur ~ abt ll tat wat coz jus gal
19	israel hudson hezbollah borscht iran enrichment git github rake integrity

Some topics pick up foreign words, as we saw before with unsupervised naïve Bayes (3: Portuguese; 7: French; 18: SMS English), and some topics are fairly specific subjects (2: gardening; 5: baking; 6: American politics; 9: Sudan; 11: soccer; 15: programming; 16: baseball; 17: cooking and music; 19: Middle East politics and programming). Maybe you can see some others.

12.3.3 Optional: Latent Dirichlet Allocation

No one uses PLSA anymore. *Latent Dirichlet Allocation* (Blei, Ng, and Jordan, 2003) is a further development of PLSA that is by far the most widely known and used topic model. LDA is quite math-heavy and we don’t give a full description here, but just a little bit of the basic idea.

In Section 12.1.2, we introduced the notion of a *prior* distribution:

$$\underbrace{P(\theta \mid \text{data})}_{\text{posterior}} = \frac{\overbrace{P(\theta)}^{\text{prior}}\overbrace{P(\text{data} \mid \theta)}^{\text{likelihood}}}{\underbrace{P(\text{data})}_{\text{evidence}}}, \tag{12.42}$$

and we just assumed that the prior was uniform. But we can choose other distributions too. The easiest choice of prior, other than uniform, is the (symmetric) *Dirichlet distribution*:

$$\text{Dir}(\theta; \alpha) \propto \prod_i \theta_i^{\alpha-1}. \tag{12.43}$$

The parameter α is called the *concentration*.

Now we have a number of options. First, we can try to find the best model (θ) just like before. Since we are now taking the prior into account, we’re not doing maximum likelihood estimation anymore; we’re

doing *maximum a posteriori (MAP) estimation.* It's not too hard to see that the resulting estimate is exactly what the MLE estimate would have been if there were pseudocounts of $(\alpha - 1)$ for each outcome. In other words, MAP estimation with a Dirichlet prior with concentration α is equivalent to MLE estimation with add-$(\alpha - 1)$ smoothing.

But another way of thinking about it is, if we're only interested in the topics, and not θ, then we shouldn't try to estimate θ. Instead, we should integrate over all possible values of θ:

$$P(\text{topics} \mid \text{data}) = \frac{P(\text{topics}, \text{data})}{P(\text{data})} \tag{12.44}$$

$$= \frac{\int P(\text{topics}, \text{data}, \theta)\, \mathrm{d}\theta}{\int P(\text{data}, \theta) \mathrm{d}\theta} \tag{12.45}$$

That looks bad, and in all but the simplest cases, it is bad, but that doesn't stop people from trying to do it anyway.

Asuncion et al. (2009) provide a good overview and comparison of both PLSA and LDA. The LDA model looks like this:

$$P(\text{data} \mid \alpha, \beta) = \underbrace{\prod_t P(\phi_t \mid \beta) \prod_i P(\theta_i \mid \alpha)}_{\text{prior}} \underbrace{\prod_{w \in d_i} \sum_t p(t \mid i, \theta)\, p(w \mid t, \phi)}_{\text{likelihood}}. \tag{12.46}$$

The part labeled "likelihood" is more or less the same as PLSA. The part labeled "prior" has two parts; $P(\phi_t \mid \beta)$ is a Dirichlet prior on the word distributions $p(w \mid t)$, and $P(\theta_i \mid \alpha)$ is a Dirichlet prior on the topic distributions $p(t \mid i)$.

From here, we can use one of several different approximation methods to proceed. One of these (Variational Bayes or mean-field approximation) is operationally quite similar to EM. For further information, see the references listed below.

References

Asuncion, Arthur et al. (2009). "On Smoothing and Inference for Topic Models". In: *Proc. UAI*, pp. 27–34.

Blei, David M., Andrew Y. Ng, and Michael I. Jordan (2003). "Latent Dirichlet Allocation". In: *J. Machine Learning Research* 3, pp. 993–1022.

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.

Hofmann, Thomas (1999). "Probabilistic Latent Semantic Analysis". In: *Uncertainty in Artificial Intelligence (UAI)*, pp. 289–296.

Manning, Christopher D., Prabhakar Raghavan, and Hinrich Schütze (2008). *Introduction to Information Retrieval*. Cambridge University Press. URL: `http://www-nlp.stanford.edu/IR-book/`.

Mukherjee, Arjun and Bing Liu (2010). "Improving Gender Classification of Blog Authors". In: *EMNLP*, pp. 207–217.