Chapter 2

Text Classification

2.1 The Bag 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`

2.2 Classification Problem

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 1. 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,

$$k^* = \arg\max_k P(k \mid d). \tag{2.1}$$

2.3 Naïve Bayes

Question 2. 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:

$$\arg\max_k P(k \mid d) = \arg\max_k P(d)P(k \mid d) \tag{2.2}$$

$$= \arg\max_k P(k,d) \tag{2.3}$$

$$= \arg\max_k P(k)P(d \mid k). \tag{2.4}$$

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 $P(k \mid d)$.

2.3.1 Model

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

$$P(k,d) = p(k) \prod_{w \in d} p(w \mid k), \tag{2.5}$$

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 3. What are some other features that might be helpful?

2.3.2 Training

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

$$\begin{aligned} p(k) &= \frac{c(k)}{\sum_k c(k)} \\ p(w \mid k) &= \frac{c(k,w)}{\sum_{w'} c(k,w')}. \end{aligned} \tag{2.6}$$

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.

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

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

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

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?

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

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,

$$\begin{aligned} P(\text{spam} \mid \theta) &= \theta \\ P(\text{ham} \mid \theta) &= 1 - \theta \end{aligned}$$

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

$$\underbrace{P(\theta \mid \text{data})}_{\text{posterior}} = \frac{\overbrace{P(\theta)}^{\text{prior}}\overbrace{P(\text{data} \mid \theta)}^{\text{likelihood}}}{\underbrace{P(\text{data})}_{\text{evidence}}}. \tag{2.7}$$

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

$$L = \prod_i P(k_i, d_i) \tag{2.8}$$

$$= \prod_i p(k_i) \prod_{w \in d_i} p(w \mid k_i). \tag{2.9}$$

And this maximization problem has a closed-form solution, namely (2.6). See Section 2.3.6 for more details on how to derive that.

2.3.3 Classification

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

$$k^* = \arg\max_k P(k \mid d) \tag{2.10}$$

$$= \arg\max_k P(k, d) \tag{2.11}$$

$$= \arg\max_k p(k) \prod_{w \in d} p(w \mid k). \tag{2.12}$$

2.3.4 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:

$$p(w \mid k) = \frac{c(k, w) + 1}{\sum_{w'} c(k, w) + |V|} \tag{2.13}$$

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:

$$p(w \mid k) = \frac{c(k, w) + \delta}{\sum_{w'} c(k, w') + |V|\delta}. \tag{2.14}$$

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 (2.7).

Question 4. 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
```

2.3.5 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
```

[7] http://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html
[8] http://xkcd.com/323/

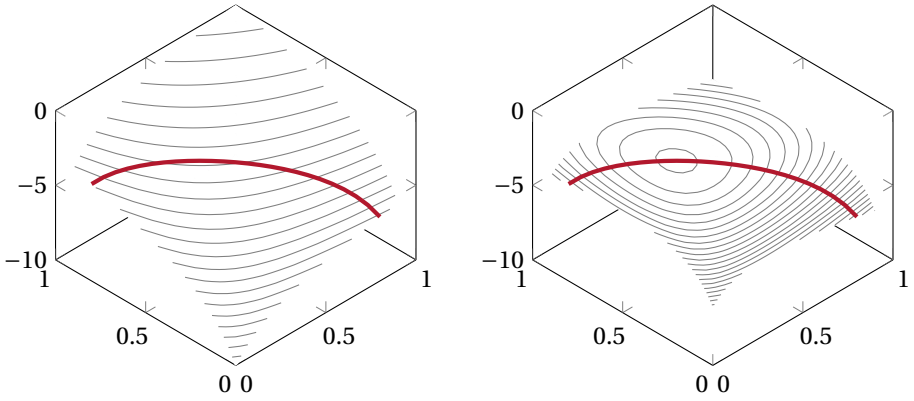

Figure 2.1: Relative frequency estimation. (a) We need to maximize the likelihood subject to the sum-to-one constraint (red line). (b) There is a value of λ that makes the Lagrangian have a maximum that does satisfy the sum-to-one constraint (red line).

```
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%.

2.3.6 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

$$\log L = \sum_k c(k) \log p(k) \tag{2.15}$$

subject to the constraint

$$\sum_k p(k) = 1. \tag{2.16}$$

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 (2.16), it will in general be sloped in the direction perpendicular to

the line (see Figure 2.1). So we create a new function $\mathscr{L}$, called the *Lagrangian*, that can level the surface out:

$$\mathscr{L} = \sum_k c(k) \log p(k) - \lambda \left(\sum_k p(k) - 1 \right). \tag{2.17}$$

This is our original objective function plus a new term. This term is zero along the line defined by (2.16), 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:

$$\frac{\partial \mathscr{L}}{\partial p(k)} = \frac{c(k)}{p(k)} - \lambda, \tag{2.18}$$

$$\frac{\partial \mathscr{L}}{\partial \lambda} = -\left(\sum_k p(k) - 1 \right). \tag{2.19}$$

Note that the partial derivative with respect to λ is zero just in case the constraint (2.16) 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:

$$p(k) = \frac{c(k)}{\sum_k c(k)}. \tag{2.20}$$

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

2.4 Logistic regression

If you were a naïve Bayes classifier, training would be like reading a bunch of texts belonging to different classes, and testing would be like reading a new text and asking yourself how plausible this text would be as an example of each of the classes. Now we're going to look at *discriminative* classifiers; if you were a discriminative classifier, training would be like reading a bunch of texts, guessing the classes of all of them, and then being told if you were right or wrong.

Above, we said that it would be harder to define a model for $P(k \mid d)$ – called a *discriminative* model as opposed to generative. It's harder, but by no means impossible. The advantage is that discriminative models are more closely matched to the task. Ultimately, we want to predict the best class given a document ($P(k \mid d)$). Generative models learn more than this; they learn $P(k, d)$, which means that they also try to learn which documents are more likely than others. In particular, they may get confused because they think that the features (words) of a document are independent, when in truth they are very interdependent. But discriminative models, because they don't care which documents are more likely than others, might do better. Ng and Jordan (2002) carry out a thorough theoretical and experimental comparison of a naïve Bayes classifier with its discriminative counterpart.

2.4.1 Definition

The discriminative version of the naïve Bayes classifier is called *logistic regression*:

$$P(k \mid d) = \frac{1}{Z(d)} \exp\left(\lambda(k) + \sum_{w \in d} \lambda(k, w) \right) \tag{2.21}$$

where $Z(d)$ is a factor that makes the distributions sum to one:

$$Z(d) = \sum_k \exp\left(\lambda(k) + \sum_{w \in d} \lambda(k, w)\right). \tag{2.22}$$

The $\lambda(k)$ and $\lambda(k, w)$ are the parameters of the model. They're analogous to $\log p(k)$ and $\log p(w \mid k)$, respectively, but they're a little harder to think about for two reasons. First, the λ's aren't probabilities; they can be positive or negative, and they can be bigger or smaller than one. Second, there's no formula analogous to (2.6) that tells us how to compute the λ's. Instead, we'll find them using numerical methods. Basically, each parameter ($\lambda(k)$ or $\lambda(k, w)$) is a knob that goes from $-\infty$ to $+\infty$; they start out zero and we adjust them to make the model fit the data.

Sometimes, this type of model is also called *log-linear* (because if you take the log of it, it's linear) or *maximum-entropy* (because of another way of deriving the model which we don't cover here).

2.4.2 Classification

Finding the most probable class of a new document is just as easy as before, because we can ignore the normalization factor:

$$k^* = \arg\max_k P(k \mid d) \tag{2.23}$$

$$= \arg\max_k \frac{1}{Z(d)} \exp\left(\lambda(k) + \sum_{w \in d} \lambda(k, w)\right) \tag{2.24}$$

$$= \arg\max_k \exp\left(\lambda(k) + \sum_{w \in d} \lambda(k, w)\right) \tag{2.25}$$

or we can even drop the exp:

$$= \arg\max_k \left(\lambda(k) + \sum_{w \in d} \lambda(k, w)\right). \tag{2.26}$$

2.4.3 Training

However, training, or estimating the parameters $\lambda(k, w)$, is more difficult than before. To simplify notation – nothing deep here, just trying to make the equations simpler – let's hallucinate a word "`<bias>`" that occurs once in every document. This makes $\lambda(k)$ and $\lambda(k, \texttt{<bias>})$ redundant, and we can safely drop $\lambda(k)$ from the model without changing it. So, from here on, we will not write $\lambda(k)$ anymore.

The likelihood, which we wish to maximize, is

$$L = \prod_i P(k_i \mid d_i) \tag{2.27}$$

$$= \prod_i \frac{1}{Z(d_i)} \exp \sum_{w \in d_i} \lambda(k_i, w). \tag{2.28}$$

Because this number is so small (among other reasons), we actually want to maximize the log-likelihood, $\log L$. In any case, there isn't a closed-form solution; instead, we have to use numerical optimization.

There are lots of methods that we can use to do this. We're going to do the easiest (but still very practical) method, *stochastic gradient ascent*.[9] It is also known as the method of steepest ascent. Imagine that the log-likelihood is an infinite, many-dimensional surface. Each point on the surface corresponds to a setting of the λ's, and the altitude of the point is the log-likelihood for that setting of the λ's. We want to find the highest point on the surface. We start at an arbitrary location (say, with all the λ's set to zero) and then repeatedly move a little bit in the steepest uphill direction.

Let $\boldsymbol{\lambda}$ be the vector of all the parameters of the model, and $\log L(\boldsymbol{\lambda})$ is the objective function, or the function we're maximizing. Gradient ascent looks like this:

initialize parameters $\boldsymbol{\lambda}$ to zero
repeat
  $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \eta \nabla \left(\log L(\boldsymbol{\lambda}) \right)$
until done

The function $\nabla(\log L)$ is the gradient of $\log L$ and gives the direction, at $\boldsymbol{\lambda}$, that goes uphill the steepest. The *learning rate* $\eta > 0$ controls how far we move at each step.

Question 5. What happens if η is too small? too big?

To guarantee convergence, η should decrease over time (for example, $\eta = 1/t$), but it's also common in practice to leave it fixed.

In stochastic gradient ascent, we break up the objective function into parts, one for each document. Thus

$$\log L(\boldsymbol{\lambda}) = \sum_i \log P(k_i \mid d_i). \tag{2.29}$$

Then the optimization looks like this:

initialize parameters $\boldsymbol{\lambda}$ to zero
repeat
  for each i **do**
    $\boldsymbol{\lambda} \leftarrow \boldsymbol{\lambda} + \eta \nabla \left(\log P(k_i \mid d_i) \right)$
  end for
until done

This usually requires less memory and is easier to implement.

The above algorithm can implemented using an automatic differentiation package. Alternatively, we

[9] If we're minimizing a function, then we use stochastic gradient *de*scent, and this is the name that the method is more commonly known by.

can implement the gradient ourselves. It is the vector of all the partial derivatives, $\frac{\partial}{\partial\lambda(k,w)}\log L$:

$$\frac{\partial}{\partial\lambda(k,w)}\log L = \sum_i \frac{\partial}{\partial\lambda(k,w)}\log P(k_i \mid d_i) \tag{2.30}$$

$$= \sum_i \frac{\partial}{\partial\lambda(k,w)}\left(\sum_{w'\in d_i}\lambda(k_i,w') - \log Z(d_i)\right) \tag{2.31}$$

$$= \sum_i \left(\delta(k,k_i)c(w\in d_i) - \frac{\partial}{\partial\lambda(k,w)}\log Z(d_i)\right) \tag{2.32}$$

$$= \sum_i \left(\delta(k,k_i)c(w\in d_i) - \frac{1}{Z(d_i)}\frac{\partial}{\partial\lambda(k,w)}Z(d_i)\right) \tag{2.33}$$

$$= \sum_i \left(\delta(k,k_i)c(w\in d_i) - \frac{1}{Z(d_i)}\frac{\partial}{\partial\lambda(k,w)}\sum_{k'}\exp\sum_{w'\in d_i}\lambda(k',w')\right) \tag{2.34}$$

$$= \sum_i \left(\delta(k,k_i)c(w\in d_i) - \frac{\exp\sum_{w'\in d_i}\lambda(k,w')}{Z(d_i)}\frac{\partial}{\partial\lambda(k,w)}c(w\in d_i)\right) \tag{2.35}$$

$$= \sum_i \frac{1}{P(k_i\mid d_i)}P(k_i\mid d_i)\left(\delta(k,k_i) - P(k\mid d_i)\right)c(w\in d_i) \tag{2.36}$$

$$= \sum_i c(w\in d_i)(\delta(k,k_i) - P(k\mid d_i)). \tag{2.37}$$

The first term is just the number of times that w has been seen in documents with class k. The second is the model's *expected* number of times that w is seen in documents with class k, without actually looking at the correct class labels k_i.

Some more calculus would demonstrate that the likeilhood is a concave function, so that the only local maximum is the global maximum, which is very good news for optimizing this function.

So the optimization algorithm looks like this:[10]

initialize parameters $\lambda(k,w)$ to zero
repeat
  for each i **do**
    for each w in d (including duplicates) **do**
      $\lambda(k_i,w) \leftarrow \lambda(k_i,w) + \eta$
      for each k **do**
        $\lambda(k,w) \leftarrow \lambda(k,w) - \eta\cdot P(k\mid d_i)$
      end for
    end for
  end for
until done

2.4.4 Regularization (smoothing)

Smoothing was important for the naïve Bayes classifier, and the most visible reason why was unknown words. Unknown words aren't as much of a problem for logistic regression, but smoothing can still help.

[10] An earlier version of this pseudocode had w loop over all the word types in the vocabulary, and the updates to the λ's were multiplied by $c(w\in d)$. The notation is now more consistent with the rest of this chapter.

Question 6. How do naïve Bayes and logistic regression react to unknown words differently? Why is it not as much of a problem for logistic regression?

The general problem that both methods face is overfitting: learning the training data so well that it fails to generalize to new data. Suppose that a word w is seen only once in the training data; it's not a particularly spammy or hammy word, but it happens to occur in a spam document. So logistic regression will learn a very high weight for $\lambda(\text{spam}, w)$, but it really shouldn't.

The solution is to add a *regularization* term to the objective function that encourages weights to be small. Let's call this new objective function F:

$$F(\boldsymbol{\lambda}) = \underbrace{-\frac{C}{2}\sum_{k,w}\lambda(k,w)^2}_{\text{regularization}} + \underbrace{\log L(\boldsymbol{\lambda})}_{\text{log-likelihood}} \tag{2.38}$$

$$= -\frac{C}{2}\sum_{k,w}\lambda(k,w)^2 + \log\left(\prod_i \frac{1}{Z(d_i)}\exp\sum_{w\in d_i}\lambda(k_i,w)\right). \tag{2.39}$$

If we think of the weights λ as a vector, the regularization term is proportional to the square of the norm of the vector. More precisely, the Euclidean norm, also known as the ℓ_2 norm. So this kind of regularization is called ℓ_2 *regularization*.

This is very easy to implement: just inside the **for** i loop, before anything else, add

for each k, w **do**
  $\lambda(k,w) \leftarrow \lambda(k,w)\cdot(1-\eta\cdot C)$
end for

2.4.5 Experiment

We trained a logistic regression classifier on the same gender-classification task as before. We used 30 iterations, a learning rate of $\eta = 0.01$, and ℓ_2 regularization with $C = 0.1$. We tried just the standard bag of words (unigram) and we also added bigram features.

method	unigram	bigram
naïve Bayes	65.1	66.5
logistic regression	66.7	69.3

So logistic regression wins, and we can see that it is more able to take advantage of the bigram features.

References

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.

Ng, Andrew Y. and Michael I. Jordan (2002). "On Discriminative vs. Generative classifiers: A comparison of logistic regression and naive Bayes". In: *Advances in Neural Information Processing Systems*. Vol. 14, pp. 841–848.