Text mining and topic models

Charles Elkan
elkan@cs.ucsd.edu

February 15, 2012

Text mining means the application of learning algorithms to documents consisting of words and sentences. Text mining tasks include

- classifier learning
- clustering, and
- theme identification.

Classifiers for documents are useful for many applications. Major uses for binary classifiers include spam detection and personalization of streams of news articles. Multiclass classifiers are useful for routing messages to recipients.

Most classifiers for documents are designed to categorize according to subject matter. However, it is also possible to learn to categorize according to qualitative criteria such as helpfulness for product reviews submitted by consumers.

In many applications of multiclass classification, a single document can belong to more than one category, so it is correct to predict more than one label. This task is specifically called multilabel classification. In standard multiclass classification, the classes are mutually exclusive, i.e. a special type of negative correlation is fixed in advance. In multilabel classification, it is important to learn the positive and negative correlations between classes.

1 The bag-of-words representation

In order to do text mining, the first question we must answer is how to represent documents. For genuine understanding of natural language one must obviously preserve the order of the words in documents. However, for many large-scale data mining tasks, including classifying and clustering documents, it is sufficient to use a simple representation that loses all information about word order.

Given a collection of documents, the first task to perform is to identify the set of all words used a least once in at least one document. This set is called the vocabulary. Often, it is reduced in size by keeping only words that are used in at least two documents. (Words that are found only once are often mis-spellings or other mistakes.) Although the vocabulary is a set, we fix an arbitrary ordering for it so we can refer to word 1 through word V where V is the size of the vocabulary.

Once the vocabulary has been fixed, each document is represented as a vector with integer entries of length $V = m$. If this vector is x then its jth component x_j is the number of appearances of word j in the document. The length of the document is $n = \sum_{j=1}^{m} x_j$. For typical documents, n is much smaller than m and $x_j = 0$ for most words j.

Many applications of text mining also eliminate from the vocabulary so-called "stop" words. These are words that are common in most documents and do not correspond to any particular subject matter. In linguistics these words are sometimes called "function" words. They include pronouns (you, he, it) connectives (and, because, however), prepositions (to, of, before), auxiliaries (have, been, can), and generic nouns (amount, part, nothing). It is important to appreciate, however, that generic words carry a lot of information for many tasks, including identifying the author of a document or detecting its genre.

A collection of documents is represented as a two-dimensional matrix where each row describes a document and each column corresponds to a word. Each entry in this matrix is an integer count; most entries are zero. It makes sense to view each column as a feature. It also makes sense to learn a low-rank approximation of the whole matrix; doing this is called latent semantic analysis (LSA) and is discussed elsewhere.

2 The multinomial distribution

Once we have a representation for individual documents, the natural next step is to select a model for a set of documents. It is important to understand the difference between a "representation" and a "model." This model is a probability distribution. Given a training set of documents, we will choose values for the parameters of the distribution that make the training documents have high probability. Then, given a test document, we can evaluate its probability according to the model. The higher this probability is, the more similar the test document is to the training set.

The probability distribution that we use is the multinomial. Mathematically, this distribution is

$$p(x;\theta) = \Big(\frac{n!}{\prod_{j=1}^m x_j!}\Big)\Big(\prod_{j=1}^m \theta_j^{x_j}\Big). \tag{1}$$

where the data x are a vector of non-negative integers and the parameters θ are a real-valued vector. Both vectors have the same length m.

Intuitively, θ_j is the probability of word j while x_j is the count of word j. Each time word j appears in the document it contributes an amount θ_j to the total probability, hence the term $\theta_j^{x_j}$. The components of θ are non-negative and have unit sum: $\sum_{j=1}^m \theta_j = 1$.

Each vector x of word counts can be generated by any member of an equivalence class of sequences of words. In Equation (1) the first factor in parentheses is called a multinomial coefficient. It is the size of the equivalence class of x, that is the number of different word sequences that yield the same counts. The second factor in parentheses in Equation (1) is the probability of any individual member of the equivalence class of x.

Like any discrete distribution, a multinomial has to sum to one, where the sum is over all possible data points. Here, a data point is a document containing n words. The number of such documents is exponential in their length n: it is m^n. The probability of any individual document will therefore be very small. What is important is the relative probability of different documents. A document that mostly uses words with high probability will have higher relative probability.

At first sight, computing the probability of a document requires $O(m)$ time because of the product over j. However, if $x_j = 0$ then $\theta_j^{x_j} = 1$ so the jth factor can be omitted from the product. Similarly, $0! = 1$ so the jth factor can be omitted from $\prod_{j=1}^m x_j!$. Hence, computing the probability of a document needs only $O(n)$ time.

Because the probabilities of individual documents decline exponentially with length n, it is necessary to do numerical computations with log probabilities:

$$\log p(x;\theta) = \log n! - [\sum_{j=1}^m \log x_j!] + [\sum_{j=1}^m x_j \cdot \log \theta_j]$$

Given a set of training documents, the maximum-likelihood estimate of the jth parameter is

$$\theta_j = \frac{1}{T}\sum_x x_j$$

where the sum is over all documents x belonging to the training set. The normalizing constant is $T = \sum_x \sum_j x_j$ which is the sum of the sizes of all training documents.

If a multinomial has $\theta_j = 0$ for some j, then every document with $x_j > 0$ for this j has zero probability, regardless of any other words in the document. Probabilities that are perfectly zero are undesirable, so we want $\theta_j > 0$ for all j. Smoothing with a constant c is the way to achieve this. We set

$$\theta_j = \frac{1}{T'}(c + \sum_x x_j).$$

The constant c is called a pseudocount. Intuitively, it is a notional number of appearances of word j that are assumed to exist, regardless of the true number of appearances. Typically c is chosen in the range $0 < c \leq 1$. Because the equality $\sum_j \theta_j = 1$ must be preserved, the normalizing constant must be $T' = mc + T$. In order to avoid big changes in the estimated probabilities θ_j, one should have $c < T/m$.

Technically, one multinomial is a distribution over all documents of a fixed size n. Therefore, what is learned by the maximum-likelihood process just described is in fact a different distribution for each size n. These distributions, although separate, have the same parameter values. Parameters are said to be "tied" if they are required to be the same for different distributions.

3 Generative processes

Suppose that we have a collection of documents, and we want to find an organization for these, i.e. we want to do unsupervised learning.

A common way to do unsupervised learning is to assume that the data were generated by some probabilistic process, and then to infer the parameters of this process. The generative process is a specification of a parametrized family of distributions. Learning is based on the principle of maximum likelihood, or some refinement of this principle such as maximum *a posteriori* (MAP) probability.

A generative process for a single document is as follows:

```
A: fix a multinomial distribution with parameter vector $\phi$ of length $V$
B: for each word in the document
       draw a word $w$ according to $\phi$
```

Above, step A sets up the probability distributions that are then used in step B to produce the observed training data.

A single multinomial distribution can only represent a category of closely related documents. For a collection of documents of multiple categories, a simple generative process is

```
A:  fix a multinomial $\alpha$ over categories 1 to $K$
    for category number 1 to category number $K$
        fix a multinomial with parameter vector $\phi_k$
B: for document number 1 to document number $M$
    draw a category $z$ according to $\alpha$
    for each word in the document
        draw a word $w$ according to $\phi_z$
```

Note that z is an integer between 1 and K. For each document, the value of z is hidden, meaning that it exists conceptually, but it is never known, not even for training data. The generative process above gives the following global probability distribution:

$$f(x) = \sum_{k=1}^{K} \alpha_k f(x; \phi_k).$$

where x is a document and ϕ_k is the parameter vector of the kth multinomial.

In general, x could be a data point of any type and ϕ_k could be the parameters of any appropriate distribution. K is called the number of components in the mixture model. For each k, $f(x; \theta_k)$ is the distribution of component number k. The scalar α_k is the proportion of component number k. A distribution like this is called a mixture distribution. In general, the components can be probability density functions (pdfs), for example Gaussians, or probability mass functions (pmfs), for example multinomials. We will see later how to do maximum likelihood estimation for mixture distributions, using a technique called expectation-maximization.

4 Latent Dirichlet allocation

The specific topic model we consider is called latent Dirichlet allocation (LDA). (The same abbreviation is also used for linear discriminant analysis, which is unrelated.) LDA is based on the intuition that each document contains words from multiple topics; the proportion of each topic in each document is different, but the topics themselves are the same for all documents.

The generative process assumed by the LDA model is as follows:

```
Given: Dirichlet distribution with parameter vector $\alpha$ of length $K$
Given: Dirichlet distribution with parameter vector $\beta$ of length $V$
for topic number 1 to topic number $K$
    draw a multinomial with parameter vector $\phi_k$ according to $\beta$
for document number 1 to document number $M$
    draw a topic distribution, i.e. a multinomial $\theta$ according to $\alpha$
    for each word in the document
        draw a topic $z$ according to $\theta$
        draw a word $w$ according to $\phi_z$
```

Note that z is an integer between 1 and K for each word.

Before we go into the LDA model in mathematical detail, we need to review the Dirichlet distribution. The Dirichlet is useful because it is a valid prior distribution for the multinomial distribution. Concretely, a Dirichlet distribution is a probability density function over the set of all multinomial parameter vectors. This set is all vectors γ of length m such that $\gamma_s \geq 0$ for all s and $\sum_{s=1}^{m} \gamma_s = 1$.

The Dirichlet distribution itself has parameter vector α of length m. There is no constraint on $\sum_{s=1}^{m} \alpha_s$, but $\alpha_s > 0$ is required for all s. Concretely, the equation for the Dirichlet distribution is

$$p(\gamma|\alpha) = \frac{1}{D(\alpha)} \prod_{s=1}^{m} \gamma_s^{\alpha_s - 1}.$$

The function D is a normalizing constant, so by definition

$$D(\alpha) = \int_{\gamma} \prod_{s=1}^{m} \gamma_s^{\alpha_s - 1}$$

where the integral is over the set of all unit vectors γ. Explicitly, the D function is

$$D(\alpha) = \frac{\prod_{s=1}^{m} \Gamma(\alpha_s)}{\Gamma(\sum_{s=1}^{m} \alpha_s)}$$

where the Γ function is the standard continuous generalization of the factorial such that $\Gamma(k) = (k-1)!$ if k is an integer. The D function is similar to the reciprocal of a multinomial coefficient.

Maximum likelihood fitting of the LDA distribution would find the optimal α and β parameter vectors. Typically, this is not done and instead α and β are treated as fixed. The goal of learning is typically to find likely values for the hidden variables, that is the multinomial parameter vectors and which topic z each word of each document belongs to.

5 Training via Gibbs sampling

For learning, the training data are the words in all documents. The prior distributions α and β are assumed to be fixed and known, as are the number K of topics, the number M of documents, the length N_m of each document, and the cardinality V of the vocabulary (i.e. the dictionary of all words). Learning has two goals: (i) to infer the document-specific multinomial θ for each document, and (ii) to infer the topic distribution ϕ_k for each topic.

The algorithm that we use for learning is called collapsed Gibbs sampling. It does not infer the θ and ϕ_k distributions directly. Instead, it infers the hidden value z for each word occurrence in each document. For an alternative exposition of this algorithm, see [Heinrich, 2005].

Although documents are represented as vectors of counts, Gibbs sampling is based on occurrences of words. Different appearances of the same word, in the same or different documents, may have be assigned to different topics; that is, each appearance of every word has its own z value.

Suppose that we know the value of z for every word occurrence in the corpus except occurrence number i. The idea of Gibbs sampling is to draw a z value for i randomly according to its distribution, then assume that we know this value, then draw a z value for another word, and so on. It can be proved that eventually this process converges to a correct distribution of z values for all words in the corpus. Note that Gibbs sampling never converges to a fixed z for each w; instead it converges to a distribution of z values for each w.

Let $\bar{w}$ be the sequence of words making up the entire corpus, and let $\bar{z}$ be a corresponding sequence of z values. Note that $\bar{w}$ is not a vector of word counts. As mentioned above, Gibbs sampling is based on viewing each document as a sequence of words. However, the particular sequence used can be arbitrary. The equations we finally obtain below will depend only on counts of words in documents, and not on any particular sequencing of words.

Use the notation $\bar{w}'$ to mean $\bar{w}$ with word number i removed, so $\bar{w} = \{w_i, \bar{w}'\}$.

Similarly, write $\bar{z} = \{z_i, \bar{z}'\}$. In order to do Gibbs sampling, we need to compute

$$p(z_i|\bar{z}', \bar{w}) = \frac{p(\bar{z}, \bar{w})}{p(\bar{z}', \bar{w})} = \frac{p(\bar{w}|\bar{z})p(\bar{z})}{p(w_i|\bar{z}')p(\bar{w}'|\bar{z}')p(\bar{z}')}$$

for $z_i = 1$ to $z_i = K$. In principle the entire denominator can be ignored, because it is a constant that does not depend on z_i. However, we will pay attention to the second and third factors in the denominator, because they lead to cancellations with the numerator. So we will evaluate

$$p(z_i|\bar{z}', \bar{w}) \propto \frac{p(\bar{w}|\bar{z})p(\bar{z})}{p(\bar{w}'|\bar{z}')p(\bar{z}')}. \tag{2}$$

We will work out the four factors of (2) one by one. Consider $p(\bar{z})$ first, and let $\bar{z}$ refer temporarily to just document number m. For this document,

$$p(\bar{z}|\theta) = \prod_{i=1}^{N_m} p(z_i) = \prod_{k=1}^{K} \theta_k^{n_{mk}}$$

where n_{mk} is the number of times $z_i = k$ within document m, and θ is the multinomial parameter vector for document m. What we really want is $p(\bar{z}|\alpha)$, which requires averaging over all possible θ. This is

$$p(\bar{z}|\alpha) = \int_\theta p(\theta|\alpha)p(\bar{z}|\theta).$$

By the definition of the Dirichlet distribution,

$$p(\theta|\alpha) = \frac{\prod_{k=1}^{K} \theta_k^{\alpha_k - 1}}{D(\alpha)}.$$

Therefore,

$$\begin{aligned} p(\bar{z}|\alpha) &= \frac{1}{D(\alpha)} \int_\theta \Big(\prod_{k=1}^{K} \theta_k^{\alpha_k - 1} \prod_{k=1}^{K} \theta_k^{n_{mk}} \Big) \\ &= \frac{1}{D(\alpha)} \int_\theta \prod_{k=1}^{K} \theta_k^{n_{mk} + \alpha_k - 1}. \end{aligned}$$

The integral above is an unnormalized Dirichlet distribution, so we get

$$p(\bar{z}|\alpha) = \frac{D(\bar{n}_m + \alpha)}{D(\alpha)}.$$

Here, $\bar{n}_m$ is the vector of topic counts $\langle n_{m1}, \ldots, n_{mK} \rangle$ for document number m. Similarly,

$$p(\bar{z}'|\alpha) = \frac{D(\bar{n}'_m + \alpha)}{D(\alpha)}$$

where $\bar{n}'_m$ is $\bar{n}_m$ with one subtracted from the count for topic number z_i. For the corpus of all M documents,

$$p(\bar{z}|\alpha) = \prod_{m=1}^{M} \frac{D(\bar{n}_m + \alpha)}{D(\alpha)} \tag{3}$$

and

$$p(\bar{z}'|\alpha) = \prod_{m=1}^{M} \frac{D(\bar{n}'_m + \alpha)}{D(\alpha)}. \tag{4}$$

Note that all factors are identical in (3) and (4), except for the document m that position i is part of.

Referring back to (2), consider $p(\bar{w}|\bar{z})$, which means $p(\bar{w}|\bar{z}, \beta)$. This is

$$\int_{\Phi} p(\Phi|\beta) p(\bar{w}|\bar{z}, \Phi)$$

where Φ is a collection of K different topic distributions ϕ_k. Again by the definition of the Dirichlet distribution,

$$p(\Phi|\beta) = \prod_{k=1}^{K} \frac{1}{D(\beta)} \prod_{t=1}^{V} \phi_{kt}^{\beta_t - 1}.$$

Now consider $p(\bar{w}|\bar{z}, \Phi)$. To evaluate this, group the words w_i together according to which topic z_i they come from:

$$p(\bar{w}|\bar{z}, \Phi) = \prod_{k=1}^{K} \prod_{i:z_i=k} p(w_i|z_i, \Phi) = \prod_{k=1}^{K} \prod_{t=1}^{V} \phi_{kt}^{q_{kt}}$$

where q_{kt} is the number of times that word t occurs with topic k in the whole corpus. We get that

$$p(\bar{w}|\bar{z}, \beta) = \int_{\Phi} \Big(\prod_{k=1}^{K} \frac{1}{D(\beta)} \prod_{t=1}^{V} \phi_{kt}^{\beta_t - 1} \Big) \Big(\prod_{k=1}^{K} \prod_{t=1}^{V} \phi_{kt}^{q_{kt}} \Big)$$

$$= \int_\Phi \Big(\prod_{k=1}^{K} \frac{1}{D(\beta)} \prod_{t=1}^{V} \phi_{kt}^{\beta_t - 1 + q_{kt}} \Big)$$

$$= \prod_{k=1}^{K} \int_{\phi_k} \frac{1}{D(\beta)} \prod_{t=1}^{V} \phi_{kt}^{\beta_t - 1 + q_{kt}}$$

$$= \prod_{k=1}^{K} \frac{1}{D(\beta)} D(\bar{q}_k + \beta)$$

where $\bar{q}_k$ is the vector of counts over the whole corpus of words belonging to topic k. This equation is similar to (3), with the corpus divided into K topics instead of into M documents.

Referring back again to (2), we get that $p(z_i|\bar{z}', \bar{w})$ is proportional to

$$\prod_{k=1}^{K} \frac{D(\bar{q}_k + \beta)}{D(\beta)} \prod_{k=1}^{K} \frac{D(\beta)}{D(\bar{q}'_k + \beta)} \prod_{m=1}^{M} \frac{D(\bar{n}_m + \alpha)}{D(\alpha)} \prod_{m=1}^{M} \frac{D(\alpha)}{D(\bar{n}'_m + \alpha)}.$$

The $D(\beta)$ and $D(\alpha)$ factors obviously cancel above. The products can be eliminated also because $\bar{q}_k + \beta = \bar{q}'_k + \beta$ except when the topic $k = z_i$, and $\bar{n}_m + \alpha = \bar{n}'_m + \alpha$ except for the document m that word i belongs to. So,

$$p(z_i|\bar{z}', \bar{w}) \propto D(\bar{q}_{z_i} + \beta) \frac{1}{D(\bar{q}'_{z_i} + \beta)} D(\bar{n}_m + \alpha) \frac{1}{D(\bar{n}'_m + \alpha)}$$

For any vector γ, the definition of the D function is

$$D(\gamma) = \frac{\prod_s \Gamma(\gamma_s)}{\Gamma(\sum_s \gamma_s)}$$

where s indexes the components of γ. Using this definition, $p(z_i = j|\bar{z}', \bar{w})$ is proportional to

$$\frac{\prod_t \Gamma(q_{jt} + \beta_t)}{\Gamma(\sum_t q_{jt} + \beta_t)} \frac{\Gamma(\sum_t q'_{jt} + \beta_t)}{\prod_t \Gamma(q'_{jt} + \beta_t)} \frac{\prod_k \Gamma(n_{mk} + \alpha_k)}{\Gamma(\sum_k n_{mk} + \alpha_k)} \frac{\Gamma(\sum_k n'_{mk} + \alpha_k)}{\prod_k \Gamma(n'_{mk} + \alpha_k)}.$$

Now $q_{jt} + \beta_t = q'_{jt} + \beta_t$ except when $t = w_i$, in which case $q_{jt} + \beta_t = q'_{jt} + \beta_t + 1$, so

$$\frac{\prod_t \Gamma(q_{jt} + \beta_t)}{\prod_t \Gamma(q'_{jt} + \beta_t)} = \frac{\Gamma(q'_{jw_i} + \beta_{w_i} + 1)}{\Gamma(q'_{jw_i} + \beta_{w_i})}.$$

Applying the fact that $\Gamma(x+1)/\Gamma(x) = x$ yields

$$\frac{\Gamma(q'_{jw_i} + \beta_{w_i} + 1)}{\Gamma(q'_{jw_i} + \beta_{w_i})} = q'_{jw_i} + \beta_{w_i}.$$

Similarly,

$$\frac{\prod_k \Gamma(n_{mk} + \alpha_k)}{\prod_k \Gamma(n'_{mk} + \alpha_k)} = n'_{mj} + \alpha_j$$

where $j = z_i$ is the candidate topic assignment of word i.

Summing over all words t in the vocabulary gives

$$\sum_{t=1}^{V} q_{jt} + \beta_t = 1 + \sum_{t=1}^{V} q'_{jt} + \beta_t.$$

so

$$\frac{\Gamma(\sum_t q'_{jt} + \beta_t)}{\Gamma(\sum_t q_{jt} + \beta_t)} = \frac{1}{\sum_t q'_{jt} + \beta_t}$$

and similarly

$$\frac{\Gamma(\sum_k n'_{mk} + \alpha_k)}{\Gamma(\sum_k n_{mk} + \alpha_k)} = \frac{1}{\sum_k n'_{mk} + \alpha_k}.$$

Putting the simplifications above yields

$$p(z_i = j|\bar{z}', \bar{w}) \propto \frac{q'_{jw_i} + \beta_{w_i}}{\sum_t q'_{jt} + \beta_t} \frac{n'_{mj} + \alpha_j}{\sum_k n'_{mk} + \alpha_k}. \tag{5}$$

This result says that occurrence number i is more likely to belong to topic j if q'_{jw_i} is large, or if n'_{mj} is large; in other words, if the same word occurs often with topic j elsewhere in the corpus, or if topic j occurs often elsewhere inside document m.

6 Implementation notes

The inner loop of the Gibbs sampling algorithm is to select a new topic label for position i. This is done by drawing a random number uniformly between 0 and 1, and using it to index into the unit interval which is divided into subintervals of length $p(z_i = j|\bar{z}', \bar{w})$. In Equation (5) the vectors α and β are fixed and known. Each value q'_{jw_i} depends on the current assignment of topics to each appearance of the word w_i throughout the corpus, not including the appearance as word number i. These assignments can be initialized in any desired way, and are then updated one at a time by the Gibbs sampling process. Each value n'_{mj} is the count of how many words within document m are assigned to topic j, not including word number i. These counts can be computed easily after the initial topic assignments are chosen, and then they can be updated in constant time whenever the topic assigned to a word changes.

The LDA generative model treats each word in each document individually. However, the specific order in which words are generated does not influence the probability of a document according to the generative model. Similarly, the Gibbs sampling algorithm works with a specific ordering of the words in the corpus. However, any ordering will do. Hence, the sequence of words inside each training document does not need to be known. The only training information needed for each document is how many times each word appears in it. Therefore, the LDA model can be learned from the standard bag-of-words representation of documents.

The standard approach to implementing Gibbs sampling iterates over every position of every document, taking the positions in some arbitrary order. For each position, Equation (5) is evaluated for each alternative topic j. For each j, evaluating (5) requires constant time, so the time to perform one epoch of Gibbs sampling is $O(NK)$ where N is the total number of words in all documents and K is the number of topics.

7 Optimizations

Note: This section is preliminary and not required for CSE 250B. Intuitively, evaluating (4) K times for each word, when just one value $k \in \{1, \ldots, K\}$ will be chosen, is too much work. Let us consider how to do less than $O(K)$ work for each word.

The starting point is Equation (5),

$$p(z_i = j|\bar{z}', \bar{w}) \propto \frac{q'_{jw_i} + \beta_{w_i}}{\sum_t q'_{jt} + \beta_t} \frac{n'_{mj} + \alpha_j}{\sum_k n'_{mk} + \alpha_k}.$$

The factor $\sum_k n'_{mk} + \alpha_k$ depends on i but not on j for a given i, so it need not be evaluated and we can write

$$p(z_i = j|\bar{z}', \bar{w}) = \frac{1}{Z} \frac{q'_{jw_i} + \beta_{w_i}}{Q'_j + \beta} (n'_{mj} + \alpha_j) \qquad (6)$$

where $Q'_j = \sum_t q'_{jt}$ and $\beta = \sum_t \beta_t$. The normalization constant Z, which is also called the partition function, is the sum

$$Z = \sum_{j=1}^{K} \frac{q'_{jw_i} + \beta_{w_i}}{Q'_j + \beta}(n'_{mj} + \alpha_j).$$

As explained above, the new topic label of word i is found by using a random number between 0 and 1 to index into the unit interval, which is divided into subintervals of length $p(z_i = j|\bar{z}', \bar{w})$.

Unfortunately, the length of every subinterval depends on Z, and computing Z requires $O(K)$ time. So, we shall compute an upper bound for Z in constant time. This upper bound will yield a lower bound for each length $p(z_i = j|\bar{z}', \bar{w})$. We shall divide each subinterval into two parts, where the length of the first part is the lower bound, and the second part is the remainder. If the random number indexes into the first part of any subinterval, then all other lengths are irrelevant, which means that the $O(K)$ computation of Z is not needed.

To obtain the first part for any given j, we want a lower bound on $p(z_i = j|\bar{z}', \bar{w})$. In general, a lower bound for a fraction is obtainable from a lower bound for its numerator and an upper bound for its denominator. Below, we provide an upper bound for Z. The rest of the expression for $p(z_i = j|\bar{z}', \bar{w})$ can be evaluated in constant time, so there is no need to find an approximation for it.

However, we do want to evaluate the righthand side of (6) for as few j values as possible. Therefore, we shall sort the j values into an ordering that makes the largest values of $p(z_i = j|\bar{z}', \bar{w})$ likely to be evaluated first. In other words, we shall attempt to order the subintervals in decreasing length. If the random number drawn is small, then we will only need to evaluate a few subinterval lengths (actually, lower bounds) before finding the subinterval in which the random number lies.

For each document m, the factors $Q'_j + \beta$ and $n'_{mj} + \alpha_j$ are almost constant regardless of i. Therefore, we sort the topics j by descending order of

$$\frac{n_{mj} + \alpha_j}{Q_j + \beta}.$$

Intuitively, this ordering tries first topics that are common in document m and rare in the whole corpus, that is with high n_{mj} and low Q_j.

Now we shall derive an upper bound that is computable in constant time for the partition function. First, let us remove the dependence on the current topic assignment of i:

$$\begin{aligned}
Z(i) &= \sum_{j=1}^{K} (q'_{jw_i} + \beta_{w_i})(Q'_j + \beta)^{-1}(n'_{mj} + \alpha_j) \\
&\leq \sum_{j=1}^{K} (q_{jw_i} + \beta_{w_i})(Q_j - 1 + \beta)^{-1}(n_{mj} + \alpha_j).
\end{aligned}$$

Next, the generalized Hölder inequality gives

$$Z(i) \leq ||q_{jw_i} + \beta_{w_i}||_p \cdot ||n_{mj} + \alpha_j||_q \cdot ||(Q_j - 1 + \beta)^{-1}||_r$$

where $1/p + 1/q + 1/r = 1$. Assuming that the three norms to be multiplied are available, this upper bound for $Z(i)$ can be computed in constant time for each i. The norms can be kept updated as follows:

- Every time some instance of w_i changes topic, $||q_{jw_i} + \beta_{w_i}||_p$ is updated in constant time by adding to one entry and subtracting from one entry in the sum of K vector components each raised to the power p.
- Every time any word in document m changes topic, $||n_{mj} + \alpha_j||_q$ is updated in a similar way in constant time.
- Every time any word in any document changes topic, $||(Q_j - 1 + \beta)^{-1}||_r$ is also updated similarly in constant time.

Many choices are possible for p, q, and r. It may be advantageous to entertain three choices simultaneously, namely

$$\begin{aligned}
\langle p, q, r \rangle &= \langle 2, 2, \infty \rangle \\
\langle p, q, r \rangle &= \langle 2, \infty, 2 \rangle \\
\langle p, q, r \rangle &= \langle \infty, 2, 2 \rangle
\end{aligned}$$

and to choose for each i the smallest of the three corresponding upper bounds. For the L_∞ norm, note that

$$||x_j||_\infty = \max_j x_j$$

and therefore

$$||x_j^{-1}||_\infty = (\min_j x_j)^{-1}.$$

References

[Heinrich, 2005] Heinrich, G. (2005). Parameter estimation for text analysis. Technical report, vsonix GmbH and University of Leipzig, Germany. Available at `http://www.arbylon.net/publications/text-est.pdf`.

CSE 250B Quiz 8, February 24, 2011

In the bag-of-words approach, a document is represented by a vector x of counts that has V entries, where V is the size of the vocabulary. Let $p(x; \theta)$ be the probability of x according to a multinomial distribution with parameters θ. The above is as seen in class.

Dr. Justin Bieber says "The total probability over all possible documents is one: $\sum_{x \in X} p(x; \theta) = 1$ where X is the set of all vectors of length V with entries that are counts (i.e., non-negative integers)."

Explain why Dr. Bieber is wrong.

CSE 250B Quiz 9, March 3, 2011

Consider a collection of N training documents, each represented as a bag-of-words vector with vocabulary size V. Each document has a discrete class label such as SPORTS. We wish to train a classifier that can predict the class of a test document. As usual, $N \ll V$ but most entries are zero in the vector representing each document.

Should we use regularization when training the classifier? Why or why not?

CSE 250B Quiz 10, March 10, 2011

The final equation that we obtained for Gibbs sampling for latent Dirichlet allocation is

$$p(z_i = j|\bar{z}', \bar{w}) \propto \Big(\frac{q'_{jw_i} + \beta_{w_i}}{\sum_{t=1}^{V} q'_{jt} + \beta_t} \Big) \Big(\frac{n'_{mj} + \alpha_j}{\sum_{k=1}^{K} n'_{mk} + \alpha_k} \Big).$$

The intuitive explanation of q'_{jw_i} and n'_{mj} is as follows: position number i in document number m is more likely to belong to topic j if the same word w_i occurs often with topic j in all documents, and/or if topic j occurs often elsewhere inside document m.

[3 points] Give a similar brief intuitive explanation of β_{w_i} and of α_j in the numerators above.

Hint: The scalars β_{w_i} and α_j are pseudocounts. Consider the effect of them being large.

CSE 250B project assignment 3, due in class on Tuesday February 23, 2010.

The objective of this project is to understand Gibbs sampling as a training method for latent Dirichlet allocation (LDA) models. As explained in class, LDA is a probabilistic model for text documents that are represented in the "bag-of-words" format. This means that each document is viewed as a vector of counts, one count for each word in the vocabulary. The number of dimensions d is the number of words in the vocabulary. Typically d is much bigger than the number of documents, and also bigger than the length of each document.

What to do. First, implement the Gibbs sampling training algorithm as described in class and in the online lecture notes. (See also the explanations in [Heinrich, 2005].) Note that making the inner loop of your LDA code fast in Matlab is a challenge, but is doable. Second, for an LDA model trained on a collection of documents, write code to print the highest-probability words for each topic. Third, write code to create a visualization of the documents based on the topics of the trained model, in 3D Euclidean space. Take advantage of the graphical display functions provided by Matlab to make visualizations maximally informative. In particular, it is easy to allow a Matlab user to rotate a 3D visualization to see it from multiple points of view.

Do experiments with your LDA implementation on two datasets. The first is the Classic400 dataset, available at `http://cseweb.ucsd.edu/users/elkan/151/classic400.mat` in Matlab format. This dataset consists of 400 documents over a vocabulary of 6205 words. Each matrix entry is how many times a given word appears in a given document. The array named `truelabels` specifies which of three domains each document comes from.

With LDA, one can learn any number of topics on a dataset. Ideally, each topic is meaningful. One way to investigate meaningfulness is to look at the highest-probability words for each topic. These should be words that are semantically related in a way that a human can understand. Separately, when you use the θ vectors (for any number of topics, but three is easiest) to plot the documents in Euclidean space, ideally the true groups of documents will be distinct, assuming that ground-truth groups are known.

The second dataset should be one that you select and obtain and preprocess. It may be an interesting collection of documents. Or, find a non-text dataset for which the LDA model is appropriate, and explain why. In any case, use your code to train LDA on the dataset, provide a selection of results in your report, and explain why these results are interesting and believable.

In your report, try to answer the following questions. The questions are in no particular order, are related to each other, and do not all have clear or easy answers.

1. Can you compute the log likelihood of a training set according to a trained model?
2. Given a fixed dataset, you can learn different models by varying α and/or β. You can also vary the number of topics, and how models are initialized. So, how can you decide which of two or more alternative LDA models is better, for a fixed dataset?
3. What is a good way of picking values for the hyperparameters α and β?
4. How can you measure objectively whether an LDA model matches well a human-specified organization for a set of documents?
5. How can you determine whether an LDA model is overfitting its training data?

CSE 250B project assignment 4, due at the final exam, 3pm on Thursday March 17, 2011.

The objective of this project is to understand Gibbs sampling as a training method for latent Dirichlet allocation (LDA) models. As explained in class, LDA is a probabilistic model for text documents that are represented in the bag-of-words format. This means that each document is viewed as a vector of counts, one count for each word in the vocabulary. The number of dimensions d is the number of words in the vocabulary. Typically d is much bigger than the number of documents, and also bigger than the length of each document.

What to do. First, implement the Gibbs sampling training algorithm as described in class and in the online lecture notes. (See also the explanations in [Heinrich, 2005].) Making the inner loop of your LDA code fast in Matlab is a challenge, but is doable. Second, for an LDA model trained on a collection of documents, write code to print the highest-probability words for each topic. Third, write code to create a visualization of the documents based on the topics of the trained model, in 3D Euclidean space. Take advantage of the graphical display functions provided by Matlab to make visualizations maximally informative. In particular, it is easy to allow a Matlab user to rotate a 3D visualization to see it from multiple points of view.

Do experiments with your LDA implementation on two datasets. The first is the Classic400 dataset, available at `http://cseweb.ucsd.edu/users/elkan/151/classic400.mat` in Matlab format. This dataset consists of 400 documents over a vocabulary of 6205 words. Each matrix entry is how many times a given word appears in a given document. The array named `truelabels` specifies which of three domains each document comes from.

With LDA, one can learn any number of topics on a dataset. Ideally, each topic is meaningful. One way to investigate meaningfulness is to look at the highest-probability words for each topic. These should be words that are semantically related in a way that a human can understand. Separately, when you use the θ vectors (for any number of topics, but three is easiest) to plot the documents in Euclidean space, ideally the true groups of documents will be distinct, assuming that ground-truth groups are known.

The second dataset should be one that you select and obtain and preprocess. It may be an interesting collection of documents. Or, find a non-text dataset for which the LDA model is appropriate, and explain why. In any case, use your code to train LDA on the dataset, provide a selection of results in your report, and explain why these results are interesting and believable.

In your report, try to answer the following questions. The questions are in no particular order, are related to each other, and do not all have clear or easy answers.

1. Can you compute the log likelihood of a training set according to a trained model?
2. Given a fixed dataset, you can learn different models by varying α and/or β. You can also vary the number of topics, and how models are initialized. So, how can you decide which of two or more alternative LDA models is better, for a fixed dataset?
3. What is a good way of picking values for the hyperparameters α and β?
4. How can you measure objectively whether an LDA model matches well a human-specified organization for a set of documents?
5. How can you determine whether an LDA model is overfitting its training data?

CSE 250B project assignment 3, due on Tuesday March 6, 2012.

As before, for this project you may work in a team of either two or three students. The joint report for your team must be submitted in hard copy at the start of the lecture on March 6.

The objective of this project is to understand Gibbs sampling as a training method for latent Dirichlet allocation (LDA) models. First, implement the Gibbs sampling training algorithm as described in class and in the online lecture notes. (See also the explanations in [Heinrich, 2005].) Making the inner loop of your LDA code fast in Matlab is a challenge, but is doable. Second, for an LDA model trained on a collection of documents, write code to print the highest-probability words for each topic. Third, write code to create a visualization of the documents based on the topics of the trained model, in 3D Euclidean space. Take advantage of the graphical display functions provided by Matlab to make visualizations maximally informative. In particular, it is easy to allow a Matlab user to rotate a 3D visualization to see it from multiple points of view.

Do experiments with your LDA implementation on two datasets. The first is the Classic400 dataset, available at `http://cseweb.ucsd.edu/users/elkan/151/classic400.mat` in Matlab format. This dataset consists of 400 documents over a vocabulary of 6205 words. Each matrix entry is how many times a given word appears in a given document. The array named `truelabels` specifies which of three domains each document comes from.

With LDA, one can learn any number of topics on a dataset. Ideally, each topic is meaningful. One way to investigate meaningfulness is to look at the highest-probability words for each topic. These should be words that are semantically related in a way that a human can understand. Separately, when you use the θ vectors (for any number of topics, but three is easiest) to plot the documents in Euclidean space, ideally the true groups of documents will be distinct, assuming that ground-truth groups are known.

The second dataset should be one that you select and obtain and preprocess. It may be an interesting collection of documents. Or, find a non-text dataset for which the LDA model is appropriate, and explain why. In any case, use your code to train LDA on the dataset, provide a selection of results in your report, and explain why these results are interesting and believable.

In your report, try to answer the following questions. The questions are all related to each other, and do not all have clear or easy answers.

1. What is a sensible way to define and compute the goodness-of-fit of an LDA model with given topics ϕ_1 to ϕ_K for a fixed dataset?

2. What is a sensible way to define and compute the goodness-of-fit of an LDA model with given hyperparameters α and β, but learned topics, for a fixed dataset?

3. What is a sensible way to compare the goodness-of-fit of LDA models with different numbers K of topics for the same dataset?

4. What is a useful simple algorithm for choosing values for the hyperparameters α and β?

5. How can you determine whether an LDA model is overfitting its training data?