Chapter 5

Topic Models, PLSA, and Gibbs Sampling

5.1 Topic Modeling

Notably absent in this book is any discussion of the *meaning* of language. For example, the entire point of translation is to capture the meaning of a sentence written in one language in a second sentence written in another, yet the models we created did not deal with the meanings of the words at all. (Though they did rely on the fact that the sentences pairs of the requisite parallel corpus meant the same thing.) Indeed, the success of MT has been due to our ability to translate without the program knowing what it is talking about.

A large fraction of current CL research is trying change this, but textbooks such as this restrict themselves to areas where there is some consensus on how things should work, and as far as meanings are concerned these are few and far between. This chapter is concerned with one of these few: topic modeling.

Intuitively texts such as, e.g., newspaper articles, are "about" something — their *topic*. So consider the following

> The crisis that has been rapidly building over North Korea's suspected nuclear weapons program seems for now to have abated. Inspectors from the International Atomic Energy Agency have been sent to Pyongyang to see what they can learn about the refueling of a key reactor that is now underway. Washington welcoming this and other recent signs hinting at a more cooperative

> attitude by the North, says that it's ready to reopen high-level contacts with Kim Il Sung's regime.

If asked to pick words from the article (which dates from about 2000) that best express what it is about we would pick, say, "nuclear", "North Korea", "inspectors", as opposed to, say, "attitude", or "contacts." Furthermore, we would not hesitate to say that this articles is about North Korea's nuclear program and is not about World-War II, or the United States Congress.

This chapter is concerned with *topic modeling*, writing programs that get at this notion of "aboutness."

5.2 Probabilistic Latent Semantic Analysis

In particular we start with a formal model called *Probabilistic Latent Semantic Analysis* or PLSA. PLSA models have much in common with the language models we dealt with in Chapter 1 in so far as they allow us to assign a probability to a text, but instead of basing the probability of a word on the previous word, we base it on the topics of the document. If our corpus consists of M "documents" (e.g., newspaper articles), each with L_d words, then the probability of our corpus c is the product of the document probabilities

$$\mathrm{P}(c) = \prod_{d=1}^{M} \mathrm{P}(w_{1,l_d}^d) \tag{5.1}$$

where w_{1,l_d}^d is the sequence of words w_{1,L_d}in document d. The next equation is the key one. We say the probability of the words is

$$\mathrm{P}(w_{1,l_d}^d) = \prod_{i=1}^{L_d} \sum_{t=1}^{N} \mathrm{P}(T_i = t \mid d)\mathrm{P}(w_i \mid t). \tag{5.2}$$

Here we assume that every word w_i is generated from a single topic, T_i which is one of N possible topics, and given the topic, the word is independent of all of the other words in the document.

Notation: T_i is the random variable for the topic of word w_i. D is the random variable denoting a particular document, and a particular value is d.

It is important to emphasize here that as in all generative models when we say "word" here we are talking about word tokens, not types. Thus a given word type may be assigned different topics in different documents. We do

not bother to generate document lengths, nor the number of documents — just the words. We choose in advance how many different topics N are allowed. We will choose, say fifty. The model is then parameterized by two sets of parameters:

$$\begin{aligned} \mathrm{P}(T = t \mid D = d) &= \delta_{d,t} \\ \mathrm{P}(W_i = w \mid T_i = t) &= \tau_{t,w} \end{aligned}$$

The generative story for this model is as follows: for each word we first pick a topic according to a document specific distribution $\delta_{d,t}$, and then pick a word from this topic according to the distribution $\tau_{t,w}$. So the δ's specify the topics of the documents, and the τ's specify the words that are used in a topic. The probably of the data is the sum over all ways it could be generated.

Here we are not interested in language modeling, but determining topics, so we ignore words that are not topic specific. In particular we use a *stopword* list — a list of words to ignore. Such lists were originally created for information retrieval purposes. Stopwords appearing in the query were ignored. These words include function words such as prepositions, pronouns, modal verbs, etc. For our purposes we also remove as some very common/general words such as "large", "big", "slowly", etc.

Example 5.1: We ran a fifty topic PLSA model run on 1000 newspaper articles, including the article excerpted at the start of this chapter. It produced the following δ parameters for the article:

Topic	$\mathrm{P}(t \mid d)$
23	0.4226
28	0.5774

These two topics account for essentially all of the probability mass. Figure 5.1 shows words associated with five topics, two associated with this article, three not. While similar, topic 23 seems to be about nuclear energy in general, while 28 is more focused on international aspects of nuclear weapons. They clearly contrast with other topics such as topic 46 which is about South Africa and apartheid.

In Figure 5.1 the words we show are those that maximize the smoothed probability of the topic given the word

$$\widetilde{\mathrm{P}}(t \mid w) = \frac{n_{t,w}(c) + \theta}{n_{\circ,w}(c) + N\theta} \tag{5.3}$$

where we set θ to 5. One of the exercises asks you to think about why we used a smoothed probability here, rather than, say, an unsmoothed maximum likelihood estimate for the probability of topic given word.

Topic	Words
1	white house watkins clinton helicopter david trip staff officials golf camp military office president course cost duffy chief presidential washington
2	u.n. rwanda united government bosnia war troops nations bosnian peace serbs military forces rebels force army peacekeeping president somalia boutros-ghali
23	korea nuclear korean south japan japanese macedonia koreans waste russia economic seoul plutonium officials culture energy weapons greece problem soviet
28	north korea nuclear sanctions fuel u.s. united officials security iaea council rods international weapons agency reactor administration clinton pyongyang china
46	africa south government mandela development countries black international nations aid african country housing program apartheid economic national lane political president

Figure 5.1: The words associated with five of the fifty topics created by a PLSA model of one thousand newspaper articles.

5.3 Learning PLSA parameters

We again turn to EM to learn the necessary parameters. As before, we first consider learning with a training corpus in which every word token in our corpus c is marked with its associated topic. The same word may have a different topic in different documents. In particular, note that the word "president" appears in topics 1, 2, and 46 in Figure 5.1. With such a corpus we can set our parameters to their maximum-likelihood estimates:

$$\delta_{d,t} = \frac{n_{d,t}(c)}{n_{d,\circ}(c)} \tag{5.4}$$

$$\tau_{t,w} = \frac{n_{t,w}(c)}{n_{t,\circ}(c)}. \tag{5.5}$$

Since we do not have labeled data, we use EM. The algorithm is as follows:

1. Pick positive initial values for $\tau_{t,w}$ for all topics t and words w (equal is fine).

2. Pick positive vales for $\delta_{d,t}$ for all documents d and topics t. They

should be almost equal, but with about 2% randomness to avoid the saddle point.

3. For $i = 1, 2, \ldots$ until convergence do:

   (a) *E-step:*
   Set $n_{d,t}$ and $n_{t,w}$ to zero for all documents d, topics t, and words w
   For each word w (that appears in its document d) do:
      i. Set $p = \sum_{t'=0}^{N} \delta_{d,t'} \tau_{t',w}$,
      ii. Set $q = \frac{\delta_{d,t} \tau_{t,w}}{p}$
      iii. Set both $n_{d,t}$ and $n_{t,w}$ += q.

   (b) *M-step:*
   Set $\delta_{d,t} = n_{d,t}/n_{d,\circ}$ and $\tau_{t,w} = n_{t,w}/n_{t,\circ}$.

This is quite similar to EM for IBM model 1, though there is one significant difference. Note the instruction to initialize the $\delta_{d,t}$'s with a small random jitter to move the initial probabilities away from the saddle point where all probabilities are equal.

Example 5.2: If we seed our random number generator with a different starting point in the above example we get a different output. In one case rather than our North Korea article falling between topics 23 and 28, we get it having all or its probability mass concentrated on topic 3. This also shows that as opposed to IBM model 1, PLSA is not unimodal.

5.4 Gibbs Sampling

As we have noted before, work in statistical computational linguistics falls under three broad headings: modeling (e.g., writing down the basic equations, and assumptions that you use to describe what is going on), inference (coming up with a computational method that allows your programs to find the model parameters required), and decoding (methods to use the model in practice).

PLSA is a model, and, as described above, we used EM as our inference algorithm. In this section we introduce a second useful inference method, *Gibbs sampling*. Gibbs sampling is the most commonly used of a slew of related sampling algorithms.

In Gibbs sampling we initially "label" every event randomly. For topic modeling this means randomly assigning every word token to a topic. Usually we use a uniform distribution. Having randomly initialized the topics,

we now pretend they are correct and repeatedly go though all of the words, and reassign words to topics randomly, but instead of using a uniform distribution, use the distribution induced by the already (randomly) assigned topics.

Example 5.3: When using this procedure on the news article quoted earlier, after the initial random assignment of word tokens to topics, our example article has words assigned to most of the fifty topics, with topic 16 having eleven (the maximum) and topics 20 and 29 having zero. Most have between one and three. By the end of the procedure the word topics have moved around quite a bit. Again we get two major topics, 33 (atomic energy) and 18 (North Korea nuclear program), but we also get a smaller contribution (about 10%) in a third which seems to be about international regulatory agencies. As before, using a different seed for the random numbers leads to different outcomes.

The basic idea is that after we have initialized the topics assigned to all of the words, in some sense we "know" the $n_{t,w}$ counts that we need in in equations 5.4. Of course, when we introduced those equations we were assuming that we had labeled training data. Nevertheless, if we assign the topics to words randomly we can still do the same mathematical operations and compute and the $\tau_{t,w}$'s and $\delta_{d,t}$'s. We now *resample* each w by picking a new topic t for it, but this time not from a random distribution, but rather according to the distribution we used in the middle of the PLSA version.

That is when we pick a new topic for w, the more likely it (according to current assignments) to be generated in topic t the more likely it will be to be put in t this time.

Example 5.4: Suppose one of our documents has 1000 words, and the word "Nato" has been assigned topics 1, 2 and 3, 1 time, 3 times and 6 times respectively (and zero for all others). If the probability of "Nato" is, say, .01 for all topics, then when we resample, the probabilities of being put into these three topics will be .1, .3 and .6. We then use a random number generator. If it produces random numbers between 0 and 999, we would pick topic one if we got a number from 0 to 99.

The idea is that when assigning topics randomly some words will end up in one or two topics more than the rest, just by the luck of the draw. In the previous example, we have "Nato" and topic 3. This means that everything else being equal, documents in which "Nato" appears will tend to have document probabilities $\delta_{d,t}$ with propensity to prefer topic 3, and thus words that appear together with "Nato" will have a higher than average probability to move to topic 3 when we resample. Done many times, over all of the words, the result is a sorting of words into somewhat coherent groups.

We complicate this idea in two minor ways. First, rather than estimating our topic and document probabilities using maximum likelihood estimates we smooth them. That is

$$\begin{aligned}\widetilde{\mathrm{P}}(t \mid d) &= \frac{n_{d,t}(c) + \alpha}{n_{d,\circ}(c) + N\alpha} \\ \widetilde{\mathrm{P}}(w \mid t) &= \frac{n_{t,w}(c) + \alpha}{n_{t,\circ}(c) + V\alpha}\end{aligned}$$

where N is the number of topics and V is our vocabulary size. To see why this is important, remember that we initially assigns words to topics randomly. Earlier we said that in our running-example news story had two topics, 20 and 29, that had no words assigned to them in our story. If we did not smooth, this means that we could never have any words from those topics assigned to words in this story because the maximum likelihood estimate for $\mathrm{P}(t \mid d)$ would be zero.

Secondly, before we resample a word we "remove" it from the current topic and document counts.

$$\widetilde{\mathrm{P}}(t \mid d) = \frac{n_{d,t}(c^-) + \alpha}{n_{d,\circ}(c^-) + N\alpha} \tag{5.6}$$

$$\widetilde{\mathrm{P}}(w \mid t) = \frac{n_{t,w}(c^-) + \alpha}{n_{t,\circ}(c^-) + V\alpha} \tag{5.7}$$

Notation: c^- denotes the corpus c *minus* the word token w that we are sampling

The resulting algorithm is shown in Figure 5.2.

For reasons we will not go into here, convergence of the data likelihood is not the appropriate measure for when to stop the iteration in this model. Instead in Figure 5.2 we say to repeat "C times", where C is just a parameter chosen by the user, presumably based upon some experimentation with how the quality of the modeling changes with the number of iterations.

5.5 Topic-model Evaluation

In our discussion of topic models we have not discussed how they are used, or how their quality is to be evaluated. As for use, there is a lot of ongoing research into situations where we need judgements about the topic under consideration at a certain point in the text. For example, suppose the computer is given some written reviews of a restaurant with the goal

1. for all document d , and words w in the document, assign w to topic t from the uniform distribution over topics.
2. do C times
   (a) for all documents d and words in the document w
      i. "remove" w from our current $n_{d,t}(c)$ and $n_{t,w}(c)$ counts — i.e., decrement each by one.
      ii. estimate the probability of the word being generated by each possible topic as done for EM, but using equations 5.6 and 5.7.
      iii. choose the next assignment of w to a topic randomly from this distribution
      iv. increment $n_{d,t}$ and $n_{t,w}$ accordingly.

Figure 5.2: Gibbs sampling algorithm for topic modeling

of summarizing the information in a formatted version under the headings of "food," "service," "price," etc. At a certain point the program finds the word "great." The program might want to use a topic model to decide what aspect of a restaurant is being discussed at that point.

As for evaluation, when feasible, evaluating a model in light of its end use is always desirable. A good model is one that correctly decides what is "great" about the restaurant under review. But there is a lot to be said for evaluations which abstract away from the end use. Our evaluation of parsers was in terms of how well they were able to match a human's judgement of the correct parse. The idea is that no matter what the application, we have some reasonable idea of what parsers are suppose to do, and if we can evaluate against that ideal, it should tell us how well the parser will perform in a wide variety of situations.

So one possibility is to give human judges the word lists produced and ask them to assess their quality and give, say, some numerical grade for each list. This would allow us to find particularly good and bad topic lists and the average score would say something about the overall quality of the program. This has been done, and while you can get some levels of agreement, the directions and the scoring are sufficiently vague that you will have a hard time saying that any program is much better than any other.

An easier evaluation mechanism to administer is called *intrusion detec-*

tion. Suppose I tell you that I am going to give you a list of words which my computer program has judged to all be on the same topic, except that one of the words was selected at random, and then mixed in with the others. Your task is to find that word, the "intruder." As you might imagine, if the intruder is say, "piano" we could do reasonably well for the topics in Figure 5.1. On the other hand, the less focused our word set, the more likely it is that the random word might look as if it belonged as well as any of the others. Intrusion detection has been show to correlate quite well with general quality judgements of a set of topical words.

However, finding intruders still requires a human evaluator. So imagine if your professor wanted to grade the programs you produced to show that you have understood the material in this chapter. He or she would have to set up intrusion detection problems, and hire people to do the evaluation separately for each program in the class. In particular, it is not be possible to throw in an intruder and have your program do the evaluation. We did this in parsing but here your program would find the task trivial. It just made up the list. Of course it can find out which one does not belong!

The problem is ultimately that there does not seem to be a "ground truth" for topic modeling. It seems hard to imagine getting people to sit down with one thousand newspaper articles and writing down the correct answer. Nevertheless, there is agreement on better and worse topics. And there has been one suggestion for finding them in an unsupervised fashion.

The intuition is that good topics will have the property that the common words for the topic (ignoring stop-words) should co-occur in the same documents. Here is a a formula that rewards exactly that:

$$\sum_{m=2}^{M} \sum_{l=1}^{m-1} log \frac{D(w_m, w_l) + 1}{D(w_l)} \tag{5.8}$$

Here $D(w_m, w_l)$ is the number of documents in the corpus in which both words w_m and w_l occur, while $D(w_m)$ is the number of documents in which w_m occurs. The plus one in the numerator ensures that we do not take log of zero in the case where w_m and w_l occur in the same document. The two summations cause the inner computation to be done for each pair of the most important M words for the topic. In Figure 5.1 we showed twenty words for each topic, so we might similarly take $M = 20$. By "most important" we can use either those words which score highly by viture of being the most common for the topic, or have a high $p(w|t)$ according to Equation 5.3.

Here are four topics from the same data-set used in Figure 5.1. This time we give two of the highest scoring classes and two of the lowest according to

Equation 5.8. The rankings definitely correspond to our intuitions in that the high scores have clear topics (middle east and d-day (in World-War II)), while the low ones do not make any sense at all.

-41.8065	israel palestinian israeli west gaza arafat police authority jewish palestinians
-47.3765	d-day normandy movie invasion beach allied june war troops battle
-92.4302	day going optional began reading left rome center street campaign
-101.072	video public trim real sense fall robert james baby author

5.5.1 Tree-Substitution Grammar

We have just seen two inference mechanisms that allow a program to infer topic-model parameters from data, one using EM, a second using Gibbs sampling. In this section we are going to look at a second problem where Gibbs is appropriate, but in this case EM is not available as a reasonable alternative.

A *tree-substitution grammar* (TSG) is a grammar formalism closely related to probabilistic context-free grammars. The difference is that the rules of a context-free grammar are of the form $T \rightarrow (T \cup N)^+$, whereas in a TSG the right-hand side of a rule may be any tree fragment. Viewed slightly differently, context-free grammar rules are tree fragments consisting of a phrasal category and its immediate children. In Section 4.2.2 we referred to these as "local trees." That is, we can view the rule S $\rightarrow$ NP VP as the tree fragment (S NP VP), and NP $\rightarrow$ DT JJ NN as (NP DT JJ NN). But while in a CFG the fragment only contains the immediate children, in a TSG it many be of arbitrary depth. For example, (VP (VB talk) (PP (IN about) NP)) would represent a TSG rule in which a VP is expanded into the verb "talk", and a prepositional phrase in which there is the preposition "about", followed by a noun-phrase. To put it another way, this one TSG rule corresponds to the application of three CFG rules: (VP VB PP), (VB talk), and (IN about).

In Chapter 4 we used the Penn Treebank to create a PCFG by reading off all of the local trees, counting how often each was used, and then getting the maximum likelihood estimate by dividing by how many local trees there were for the non-terminal category in question. This will not work for TSGs. For starters, consider a parse tree with, say, forty non-terminal symbols. Naturally, if we are using this to create a CFG, this corresponds to forty

Figure 5.3: Two ways of breaking up the parse tree for "They talk about the economy"

CFG rules, one for each local tree. How many possible TSG rules are there? The answer is close to 2^40. That is approximately 10^{12}, way to many to store. (Every computer science student should be aware that $2^{10} \approx 10^3$.)

They way to see how 40 becomes 2^{40} is to visualize a general way to break a tree into rules. In Figure 5.3 we have the parse tree for the sentence "They talk about the economy" broken up two different ways. Each non-terminal node is labeled either "B" (for "break here") or "N" (for "no break here"). We now create grammar rules in the following fashion: for every non-terminal labeled "B", create a tree fragment (i.e., a rule) consisting, of it, all of its children until you encounter another B. In the tree on the left of Figure 5.3, all of the children will be labeled B, so the fragment stops immediately with just the local children, giving us a CFG. In the tree on the right the non-terminals VP, VB, and IN are instead labeled "N", so we instead get the fragment (VP (VB talk) (PP (IN about) NP)). There are something like 2^{40} possibilities because every non-terminal node can be labeled in two ways, and most such combinations of labeling will lead to different fragments.

We emphasized the 2^{40} figure because it implies that the EM algorithm as we have been using it us not practical here. Keeping track of the expected counts over all tree fragments is too expensive.

Now consider a Gibbs sampling inference algorithm for inferring a TSG. It will start by randomly creating tree fragments over the entire tree-bank. We do this by looking at each non-terminal node in each tree (except root nodes) and assigning it either a B or N value randomly with probabilities p and $1 - p$ ($p = /frac12$) is as good as any. We then resample each non-terminal until convergence. As before, we remove the node from our counts, decide if we should label it B or N according to some distribution (to be discussed in a second) and then modify the counts to reflect the new TSG grammar.

Example 5.5: Suppose random assignment of break points gave us the left-hand-side tree in Figure 5.3, and we are resampling the node VB. Since this node is part of two current fragments (VP VB PP) and (VB talk), we decrement the counts of each of these. On the other hand, the state of the sampler had the right-hand side tree as the current assignment for the VB node, removing it from consideration would decrement the count of (VP (VB talk) PP).

The probability of a tree fragment t = (X ...) where X is non-terminal

that heads the tree, is

$$\mathrm{P}(t) = \frac{n_t(c^-) + \alpha \mathrm{P}_0(t)}{n_X + \alpha} \tag{5.9}$$

Ignoring $P_0(t)$ for the moment, this is pretty much a smoothed probability for the tree t. The first terms in the numerator and denominator would by themselves give us the maximum likelihood estimation for this fragment, the number of times the fragment occurs, divided by the number of fragment tokens in the entire treebank with the same left-hand side, X.

$P_0(t)$ is a base probability for building up any fragment out of context-free grammar rules. So the probability for building (VP (VB talk) PP) is the CFG probability for (VP VB PP), times $(1-\beta)$ as the probability of *not* splitting at VB, times the CVG probability of (VB talk), β, the probability of splitting at PP.

When we sample at a non-terminal node n, we compare the probabilities of having a single tree (unsplit at this node), versus two trees, the tree that has n as a leaf, and the one that has it as the root, and if, say, the program chooses to split, then both the parent and child tree counts are incremented by one.

The point of all this is that at no point in this algorithm is it necessary to consider all possible tree fragments. So again, consider a tree-bank tree with forty non-terminal nodes (plus the root node). When we randomly assigns splits, the number of fragments will be ≤ 40, not 2^{40}. Later when we resample, there are only three tree fragments to look at, the parent fragment, the child fragment, and the combined fragment. So the Gibbs sampling algorithm for this problem is quite feasible.

The resulting grammars tend to have multilevel fragments that capture regularities that CFGs cannot express. Our running example here — (VP (VB talk) (PP (IN about) NP)) — is an example of a *verb case frame*, how a verb combines with other constituents to form it's meaning. Here, how the verb "talk" specifies the "topic" of the talking. Some other case frames found when you run the Gibbs sampler over the Penn Treebank are:

1. (VP (VB talk) (PP TO NP)) "They talk to each other"
2. (VP (VB broke) (PP (IN into) NP)) "He broke into the building"
3. (VP (VB throw) NP (PP (IN into) NP)) "They throw the ball into the ring"
4. (VP (VB transform) NP (PP (IN into) NP)) "She can transform paper into art"

5. (VP (VB prevent) NP (PP (IN from) (S VP))) "He will prevent the water from entering"

6. (VP (VB protect) NP (PP (IN from) NP)) "She will protect the dog from harm"

While we have been discussing TSGs because they showcase an advantage of Gibbs sampling, they also relate to the more general topic of this chapter, meaning. We said language must build the meaning of the whole from the meanings of the parts. Verb case frames remind us that the pieces of language that have meaning are often not individual words (what is the meaning of "from" in the above examples?) but instead words combined in particular ways. Furthermore, the combinations that most naturally relate to meaning often seem to span multiple levels of tree structure. TSGs are one way to begin finding these combinations.

5.6 Programming assignment

The data for this assignment is in news1000.txt The format of the file is as follows:

L_d
$w_1\ w_2\ \ldots\ w_{12}$
$w_{13}\ \ldots\ w_{24}$
$\ldots\ w_{L_d}$

This is repeated 1000 times, once for each news article d. The words are all of the words in the articles that (a) appear at least 5 times and in two different articles, and (b) do not appear in a stopword list of common non-topical words.

Your assignment is to create a Gibbs sampling model with 50 topics. To keep things simple, done't worry to much about the value of α. A value of 0.5 should work well. As for the number if iterations C, 10 should be a reasonably good value.

To demonstrate that your program works it should output the following facts:

1. The log likelihood of the data at convergence. (We will define convergence as the point where the log-likelihood changes by less than 1% between one iteration and the next.)

2. the probabilities of topics for article 17 (as a cross check, it has 367 words).

3. the most probable 15 words w for each topic t according to the following formula:

$$\widetilde{\mathrm{P}}(t \mid w) = \frac{n_{t,w} + \theta}{n_{\circ,w} + N\theta}$$

Set $\theta = 5$.

5.7 Exercises

Exercise 5.1: In Equation 5.1 we ignored the issue of how many words were in each document (because is it unrelated to the question of topic). Give a version that explicitly models this aspect of the corpus.

Exercise 5.2: In our PLSA algorithm we did not pick all of the document/topic probabilities equal (e.g., $\frac{1}{50}$), but instead randomly made them slightly larger or smaller. Could we make them all equal and instead randomize the word/topic probabilities? Explain.

Exercise 5.3: Explain the logic of displaying words that score highly for a topic according to Equation 5.3. In particular, discuss how the words displayed change as θ goes from 0 to some very large number.

Exercise 5.4: We saw that topics with higher values for Equation 5.8 look better than those with low values. This poses the question, why bother with topic modeling at all, and simply write programs that maximize that equation. Unfortunately this measure is very easy to "game" — to produce meaninglessly high scores for topic collections that are useless. Suppose you have implemented either of our models, as well as written the code for Equation 5.8. Suggest some small (three lines of code or so) that will dramatically increase the average coherence, but make the model a "joke."