600.465 — Natural Language Processing
Assignment 2: Language Modeling

Prof. J. Eisner — Fall 2010
Due date: Wednesday 6 October, 2 pm

This assignment will try to convince you that statistical models—even simplistic and linguistically stupid ones like n-gram models—can be very useful, provided their parameters are estimated carefully. Almost all speech recognition systems use some form of trigram model. While trigram models can be enhanced in various ways, it's surprisingly hard to improve much on their performance. (And alternative approaches that *don't* look at the trigrams just do worse.)

In addition, you will get some experience in running corpus experiments over training, development, and test sets.

Why is this assignment absurdly long? Because the assignments are really your primary reading for the class. They're shorter and more interactive than a textbook. :-) The textbook readings are usually quite helpful, and you should have at least skimmed the readings on smoothing by now, but it is not mandatory that you know them in full detail.

Programming language: You may work in any language that you like. However, we will give you some useful code as a starting point.[1] This code is provided in several programming languages for your convenience. If you want to use another language, then feel free to translate (or ignore?) our short code before continuing with the assignment. Please send your translation to the course staff so that we can make it available to the whole class.

On getting programming help: Since this is a 400-level NLP class, not a programming class, I don't want you wasting time on low-level issues like how to handle I/O or hash tables of arrays. If you are doing so, then by all means seek help from someone who knows the language better! Your responsibility is the NLP stuff—you do have to design, write, and debug the interesting code and data structures **on your own**. But I don't consider it cheating if another hacker (or the TA) helps you with your I/O routines or language syntax or compiler warning messages. These aren't Interesting$^{\text{TM}}$.

How to hand in your work: Same procedure as assignment 1. You must test that your programs run with no problems on the `ugrad` machines (named `ugrad1`–`ugrad24`)

[1] It counts word n-grams in a corpus, using hash tables, and uses the counts to calculate simple probability estimates.

before submitting them. You probably want to develop them there in the first place, since that's where the corpora are stored. (However, in principle you could copy the corpora elsewhere for your convenience.)

Again, besides the comments you embed in your source files, put all other notes, documentation, and answers to questions in a `README` file. The file should be editable so that we can insert comments and give it back to you. For this reason, we strongly prefer a plain ASCII file `README`, or a LaTeX file `README.tex` (in which case please also submit `README.pdf`). If you must use a word processor, please save as `README.rtf` in the portable, non-proprietary RTF format.

If your programs are in some language other than the ones we used, or if we need to know something special about how to compile or run them, please explain in a plain ASCII file `HOW-TO`.

Your source files, the `README` file, the `HOW-TO` file, and anything else you are submitting will all need to be placed in a single submission directory.

Notation: When you are writing the `README` file, you will need some way of typing mathematical symbols. If your file is just plain ASCII text, please use one of the following three notations and stick to it in your assignment. (If you need some additional notation not described here, just describe it clearly and use it.) Use parentheses as needed to disambiguate division and other operators.

	Text	Picts	LaTeX
$p(x \mid y)$	`p(x | y)`	`p(x | y)`	`p(x \mid y)`
$\neg x$	`NOT x`	`~x`	`\neg x`
$\bar{x}$ (set complement)	`COMPL(x)`	`\x`	`\bar{x}`
$x \subseteq y$	`x SUBSET y`	`x {= y`	`x \subseteq y`
$x \supseteq y$	`x SUPERSET y`	`x }= y`	`x \supseteq y`
$x \cup y$	`x UNION y`	`x U y`	`x \cup y`
$x \cap y$	`x INTERSECT y`	`x ^ y`	`x \cap y`
$x \geq y$	`x GREATEREQ y`	`x >= y`	`x \geq y`
$x \leq y$	`x LESSEQ y`	`x <= y`	`x \leq y`
$\emptyset$ (empty set)	`NULL`	`0`	`\emptyset`
$\mathcal{E}$ (event space)	`E`	`E`	`E`

---

1. These short problems will help you get the hang of manipulating probabilities. Let $\mathcal{E} \neq \emptyset$ denote the event space (it's just a set, also known as the sample space), and p be a function that assigns a real number in $[0, 1]$ to any subset of $\mathcal{E}$. This number is called the probability of the subset.

You are told that p satisfies the following two axioms: $p(\mathcal{E}) = 1$. $p(X \cup Y) = p(X) + p(Y)$ provided that $X \cap Y = \emptyset$.[2]

As a matter of notation, remember that the **conditional probability** $p(X \mid Z) \stackrel{\text{def}}{=} \frac{p(X \cap Z)}{p(Z)}$. For example, singing in the rain is one of my favorite rainy-day activities: so my ratio $p(\text{singing} \mid \text{rainy}) = \frac{p(\text{singing AND rainy})}{p(\text{rainy})}$ is high. Here the predicate "singing" picks out the set of singing events in $\mathcal{E}$, "rainy" picks out the set of rainy events, and the conjoined predicate "singing AND rainy" picks out the intersection of these two sets—that is, all events that are both singing AND rainy.

(a) Prove from the axioms that if $Y \subseteq Z$, then $p(Y) \leq p(Z)$.
You may use any and all set manipulations you like. Remember that $p(A) = 0$ does not imply that $A = \emptyset$ (why not?), and similarly, that $p(B) = p(C)$ does not imply that $B = C$ (even if $B \subseteq C$).

(b) Use the above fact to prove that conditional probabilities $p(X \mid Z)$, just like ordinary probabilities, always fall in the range $[0, 1]$.

(c) Prove from the axioms that $p(\emptyset) = 0$.

(d) Let $\bar{X}$ denote $\mathcal{E} - X$. Prove from the axioms that $p(X) = 1 - p(\bar{X})$. For example, $p(\text{singing}) = 1 - p(\text{NOT singing})$.

(e) Prove from the axioms that $p(\text{singing AND rainy} \mid \text{rainy}) = p(\text{singing} \mid \text{rainy})$.

(f) Prove from the axioms that $p(X \mid Y) = 1 - p(\bar{X} \mid Y)$. For example, $p(\text{singing} \mid \text{rainy}) = 1 - p(\text{NOT singing} \mid \text{rainy})$. This is a generalization of 1d.

(g) Simplify: $(p(X \mid Y) \cdot p(Y) + p(X \mid \bar{Y}) \cdot p(\bar{Y})) \cdot p(\bar{Z} \mid X)/p(\bar{Z})$

(h) Under what conditions is it true that $p(\text{singing OR rainy}) = p(\text{singing}) + p(\text{rainy})$?

(i) Under what conditions is it true that $p(\text{singing AND rainy}) = p(\text{singing}) \cdot p(\text{rainy})$?

(j) Suppose you know that $p(X \mid Y) = 0$. Prove that $p(X \mid Y, Z) = 0$.

(k) Suppose you know that $p(W \mid Y) = 1$. Prove that $p(W \mid Y, Z) = 1$.

2. All cars are either red or blue. The witness claimed the car that hit the pedestrian was blue. Witnesses are believed to be about 80% reliable in reporting car color (regardless of the actual car color). But only 10% of all cars are blue.

(a) Write an equation relating the following quantities and perhaps other quantities:

$$p(\textit{true} = \text{blue})$$
$$p(\textit{true} = \text{blue} \mid \textit{claimed} = \text{blue})$$
$$p(\textit{claimed} = \text{blue} \mid \textit{true} = \text{blue})$$

[2] In fact, probability functions p are also required to satisfy a generalization of this second axiom: if $X_1, X_2, X_3, \ldots$ is an infinite sequence of disjoint sets, then $p(\bigcup_{i=1}^{\infty} X_i) = \sum_{i=1}^{\infty} p(X_i)$. But you don't need this for this assignment.

Reminder: Here, *claimed* and *true* are *random variables*, which means that they are functions over some outcome space. For example, the probability that *claimed* = blue really means the probability of getting an outcome x such that *claimed*(x) = blue. We are implicitly assuming that the space of outcomes x is something like the set of witnessed car accidents.

(b) Match the three probabilities above with the following terms: *prior probability*, *likelihood of the evidence*, *posterior probability*.

(c) Give the values of all three probabilities. (Hint: Use Bayes' Theorem.) Which probability should the judge care about?

(d) Let's suppose the numbers 80% and 10% are specific to Baltimore. So in the previous problem, you were implicitly using the following more general version of Bayes' Theorem:

$$p(A \mid B, Y) = \frac{p(B \mid A, Y) \cdot p(A \mid Y)}{p(B \mid Y)}$$

where Y is *city* = Baltimore. Just as 1f generalized 1d, by adding a "background" condition Y, this version generalizes Bayes' Theorem. Carefully prove it.

(e) Now prove the more detailed version

$$p(A \mid B, Y) = \frac{p(B \mid A, Y) \cdot p(A \mid Y)}{p(B \mid A, Y) \cdot p(A \mid Y) + p(B \mid \bar{A}, Y) \cdot p(\bar{A} \mid Y)}$$

which gives a practical way of finding the denominator in the question 2d.

(f) Write out the equation given in question 2e with A, B, and Y replaced by specific propositions from the red-and-blue car problem. For example, Y is "*city* = Baltimore" (or just "Baltimore" for short). Now replace the probabilities with actual numbers from the problem, such as 0.8.

Yeah, it's a mickeymouse problem, but I promise that writing out a real case of this important formula won't kill you, and may even be good for you (like, on an exam).

3. Beavers can make three cries, which they use to communicate. `bwa` and `bwee` usually mean something like "come" and "go" respectively, and are used during dam maintenance. `kiki` means "watch out!" The following **conditional probability table** shows the probability of the various cries in different situations.

$p(cry \mid situation)$	Predator!	Timber!	I need help!
`bwa`	0	0.1	0.8
`bwee`	0	0.6	0.1
`kiki`	1.0	0.3	0.1

(a) Notice that each column of the above table sums to 1. Write an equation stating this, in the form $\sum_{variable} p(\cdots) = 1$.

(b) A certain colony of beavers has already cut down all the trees around their dam. As there are no more to chew, $p(timber) = 0$. Getting rid of the trees has also reduced $p(predator)$ to 0.2. These facts are shown in the following **joint probability table**. Fill in the rest of the table, using the previous table and the laws of probability. (Note that the meaning of each table is given in its top left cell.)

$p(cry, situation)$	Predator!	Timber!	I need help!	TOTAL
`bwa`				
`bwee`				
`kiki`				
TOTAL	0.2	0		

(c) A beaver in this colony cries `kiki`. Given this cry, other beavers try to figure out the probability that there is a predator.

i. This probability is written as: $p(\_\_\_\_\_)$

ii. It can be rewritten without the | symbol as: _____

iii. Using the above tables, its value is: _____

iv. Alternatively, Bayes' Theorem allows you to express this probability as:

$$\frac{p(\_\_\_\_\_) \cdot p(\_\_\_\_\_)}{p(\_\_\_\_\_) \cdot p(\_\_\_\_\_) + p(\_\_\_\_\_) \cdot p(\_\_\_\_\_) + p(\_\_\_\_\_) \cdot p(\_\_\_\_\_)}$$

v. Using the above tables, the value of this is:

$$\frac{\_\_\_\_\_ \cdot \_\_\_\_\_}{\_\_\_\_\_ \cdot \_\_\_\_\_ + \_\_\_\_\_ \cdot \_\_\_\_\_ + \_\_\_\_\_ \cdot \_\_\_\_\_}$$

This should give the same result as in part iii., and it should be clear that they are really the same computation—by constructing table (b) and doing part iii., you were *implicitly* using Bayes' Theorem. (I told you it was a trivial theorem!)

4. (a) $p(\neg\text{shoe} \mid \neg\text{nail}) = 1$ *For want of a nail the shoe was lost,*
(b) $p(\neg\text{horse} \mid \neg\text{shoe}) = 1$ *For want of a shoe the horse was lost,*
(c) $p(\neg\text{race} \mid \neg\text{horse}) = 1$ *For want of a horse the race was lost,*
(d) $p(\neg\text{fortune} \mid \neg\text{race}) = 1$ *For want of a race the fortune was lost,*
(e) $p(\neg\text{fortune} \mid \neg\text{nail}) = 1$ *And all for the want of a horseshoe nail.*

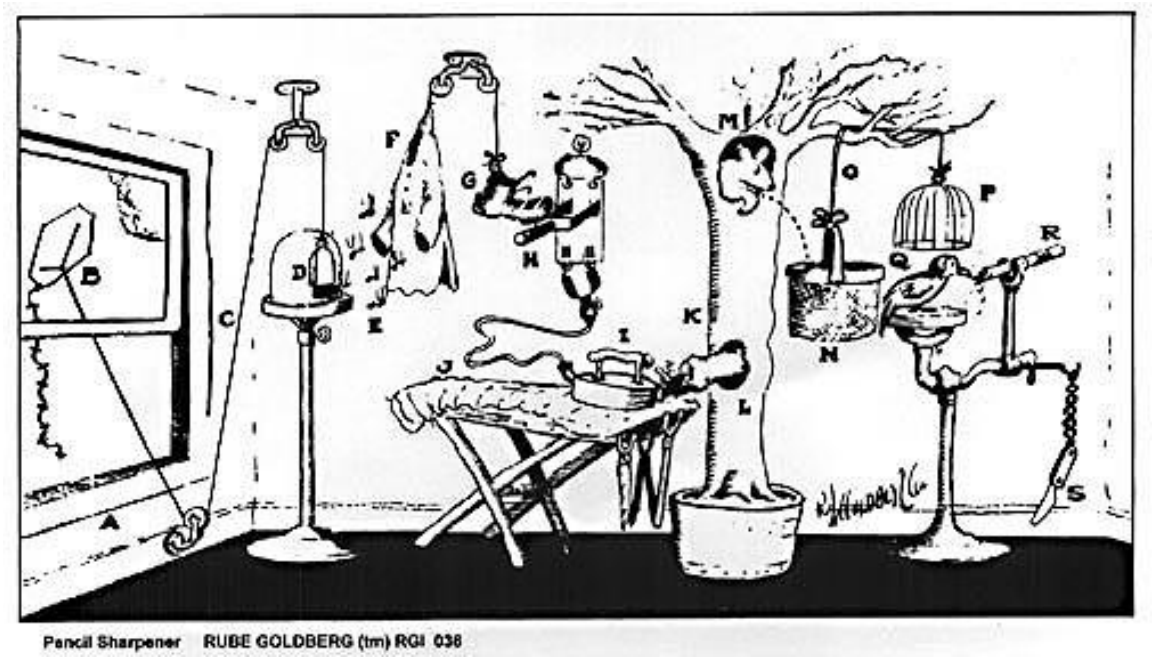

Figure 1: RUBE GOLDBERG GETS HIS THINK-TANK WORKING AND EVOLVES THE SIMPLIFIED PENCIL-SHARPENER. Open window (A) and fly kite (B). String (C) lifts small door (D) allowing moths (E) to escape and eat red flannel shirt (F). As weight of shirt becomes less, shoe (G) steps on switch (H) which heats electric iron (I) and burns hole in pants (J). Smoke (K) enters hole in tree (L), smoking out opossum (M) which jumps into basket (N), pulling rope (O) and lifting cage (P), allowing woodpecker (Q) to chew wood from pencil (R), exposing lead. Emergency knife (S) is always handy in case opossum or the woodpecker gets sick and can't work.

Show carefully that (e) follows from (a)–(d). *Hint:* Consider

$$p(\neg\text{fortune}, \neg\text{race}, \neg\text{horse}, \neg\text{shoe} \mid \neg\text{nail}),$$

as well as the "chain rule" and problems 1a, 1b, and 1k.

Note: The $\neg$ symbol denotes the boolean operator NOT.

Note: This problem is supposed to convince you that logic is just a special case of probability theory.

Note: Be glad I didn't ask you to prove the correct operation of Figure 1!

5. A **language model** is a probability function p that assigns probabilities to word sequences such as $\vec{w} = (\texttt{i}, \texttt{love}, \texttt{new}, \texttt{york})$. Think of $p(\vec{w})$ as the probability that if you turned on a radio at an arbitrary moment, its next four words would be "i love

new york"—perhaps in the middle of a longer sentence such as "the latest bumper sticker says, i love new york more than ever." We often want to consider $p(\vec{w})$ to decide whether we like $\vec{w}$ better than an alternative sequence.[3]

Suppose $\vec{w} = w_1 w_2 \cdots w_n$ (a sequence of n words). A **trigram language model** defines

$$p(\vec{w}) \stackrel{\text{def}}{=} p(w_1) \cdot p(w_2 \mid w_1) \cdot p(w_3 \mid w_1, w_2) \cdot p(w_4 \mid w_2, w_3) \cdots p(w_n \mid w_{n-2}, w_{n-1})$$

on the assumption that the sequence was generated in the order $w_1, w_2, w_3 \ldots$ ("from left to right") with each word chosen in a way dependent on the previous two words. (But the first word w_1 is not dependent on anything, since we turned on the radio at a arbitrary moment.)

(a) Expand the above definition of $p(\vec{w})$ using naive estimates of the parameters, such as

$$p(w_4 \mid w_2, w_3) \stackrel{\text{def}}{=} \frac{c(w_2 w_3 w_4)}{c(w_2 w_3)}$$

where $c(w_2 w_3 w_4)$ denotes the count of times the trigram $w_2 w_3 w_4$ was observed in a training corpus.

Remark: Naive parameter estimates of this sort are called "maximum-likelihood estimates" (MLE). They have the advantage that they maximize the probability (equivalently, minimize the perplexity) of the training data. But they will generally perform badly on test data, unless the training data were so abundant as to include all possible trigrams many times. This is why we must smooth these estimates in practice.

(b) One could also define a kind of reversed trigram language model $p_{reversed}$ that instead assumed the words were generated in reverse order ("from right to left"):

$$\begin{aligned} p_{reversed}(\vec{w}) \stackrel{\text{def}}{=} \; & p(w_n) \cdot p(w_{n-1} \mid w_n) \cdot p(w_{n-2} \mid w_{n-1}, w_n) \cdot p(w_{n-3} \mid w_{n-2}, w_{n-1}) \\ & \cdots p(w_2 \mid w_3, w_4) \cdot p(w_1 \mid w_2, w_3) \end{aligned}$$

By manipulating the notation, show that the two models are identical (i.e., $p(\vec{w}) = p_{reversed}(\vec{w})$ for any $\vec{w}$) provided that both models use MLE parameters estimated from the same training data (see problem 5a).

(c) In the data you will use in questions 6 and 14, sentences are delimited by `<s>` at the start and `</s>` at the end. For example, the following data set consists of a sequence of 3 sentences:

[3] Formally, each element $W \in \mathcal{E}$ of the underlying event space is a possible value of the *infinite* sequence of words that will come out of the radio after you turn it on. $p(\vec{w})$ is really an abbreviation for $p(\texttt{prefix}(W, |\vec{w}|) = \vec{w})$, where $|\vec{w}|$ denotes the length of the sequence $\vec{w}$. Thus, $p(\texttt{i}, \texttt{love}, \texttt{new}, \texttt{york})$ is the total probability of all infinite word sequences W that begin "i love new york ...."

`<s> do you think so </s> <s> yes </s> <s> at least i thought so </s>`

Given English training data, the probability of

`<s> do you think the </s>`

should be extremely low under any good language model. Why? In the case of the trigram model, which parameter or parameters are responsible for making this probability low?

(d) You turn on the radio as it is broadcasting an interview. Assuming a trigram model, match up expressions (A), (B), (C) with descriptions (1), (2), (3):

The expression

(A) $p(\texttt{Do}) \cdot p(\texttt{you}|\texttt{Do}) \cdot p(\texttt{think}|\texttt{Do},\texttt{you})$

(B) $p(\texttt{Do}|\texttt{<s>}) \cdot p(\texttt{you}|\texttt{<s>},\texttt{Do}) \cdot p(\texttt{think}|\texttt{Do},\texttt{you}) \cdot p(\texttt{</s>}|\texttt{you},\texttt{think})$

(C) $p(\texttt{Do}|\texttt{<s>}) \cdot p(\texttt{you}|\texttt{<s>},\texttt{Do}) \cdot p(\texttt{think}|\texttt{Do},\texttt{you})$

represents the probability that

(1) the first complete sentence you hear is `Do you think` (as in, "D'ya think?")

(2) the first 3 words you hear are `Do you think`

(3) the first complete sentence you hear starts with `Do you think`

Explain your answers briefly. Which quantity is $p(\vec{w})$? *Remark:* The distinctions matter because "Do" is more probable at the start of an English sentence than in the middle, and because (3) describes a larger event set than (1) does.

Now you are ready to build and use some n-gram language models! You will experiment with different types of smoothing. **Your starting point is the sample program `fileprob`.**

All the smoothing methods assume a finite vocabulary, so that they can easily allocate probability to all the words. But is this assumption justified? Aren't there infinitely many potential words of English that might show up in a test corpus (like "xyzzy" and "JacrobinsteinIndustries" and "fruitylicious")? Yes ... so we will *force* the vocabulary to be finite by a standard trick. We'll assemble a finite vocabulary by some means (in our case, collecting all the words we saw in the training corpus), and add one special symbol OOV (which stands for "out of vocabulary") that represents all other words. You should regard new words that you see in test data as nothing more than various spellings of the OOV symbol. For example, when you are considering the test sentence

`i saw snuffleupagus on the tv`

what you will actually compute is the probability of

`i saw` OOV `on the tv`

which is really the *total* probability of *all* sentences of the form

`i saw` [some out-of-vocabulary word] `on the tv`

Admittedly, this total probability is higher than the probability of the *particular* sentence involving `snuffleupagus`. But in most of this assignment, we only wish to compare the probability of the snuffleupagus sentence under different models. Replacing `snuffleupagus` with OOV raises the sentence's probability under all the models at once, so it need not invalidate the comparison.[4]

We do have to make sure that if `snuffleupagus` is regarded as OOV by one model, then it is regarded as OOV by all the other models, too. It's not appropriate to compare $p_{\text{model1}}(\texttt{i saw OOV on the tv})$ with $p_{\text{model2}}(\texttt{i saw snuffleupagus on the tv})$, since the former is actually the total probability of many sentences, and so will tend to be larger. So all the models must have the *same* finite vocabulary![5] V denotes the size of this vocabulary (including OOV).

Though the context "OOV `on`" never appeared in the training corpus, the smoothing method is required to give a reasonable value anyway to $p(\texttt{the} \mid \text{OOV}, \texttt{on})$, for example by backing off to $p(\texttt{the} \mid \texttt{on})$. Similarly, the smoothing method must give a reasonable (non-zero) probability to $p(\text{OOV} \mid \texttt{i}, \texttt{saw})$. Because we're merging all out-of-vocabulary words into a single word OOV, we avoid having to decide how to split this probability among them.

Your code doesn't have to detect OOV words. It doesn't need to do anything special to look up the count of a trigram such as "`i saw` OOV ." Just look up the count of "`I saw snuffleupagus`" in the usual way, and you'll get the same answer, zero, since "`snuffleupagus`" never appeared at all in the training corpus. (He's shy.)

To help you understand and debug your programs, we have grafted brackets onto all out-of-vocabulary words in *one* of the datasets (the `speech` directory). This lets you identify the OOV words at a glance. In this particular dataset, you will actually see the test sentence

`i saw [snuffleupagus] on the tv`

and therefore look up the count of the trigram "`i saw [snuffleupagus]`," which is 0.[6] You can experiment on that dataset, starting with the following problem.

---

[4] Problem 17 explores a prettier approach that may also work better for text categorization.

[5] The vocabulary of a system could be *any* list of words that you pick. It is *not* necessarily "words that appeared > 0 times." OOV is a special symbol meaning "anything else" (i.e., "not in the chosen vocabulary"). It does not mean "appeared 0 times."

[6] Your program does not have to do any special handling for the brackets. Although as it happens, the count of "`i saw snuffleupagus`" must also be 0, as both `snuffleupagus` and `[snuffleupagus]` are OOV.

6. The sample program is on the ugrad machines in the directory

   `/usr/local/data/cs465/hw2/code`

   There are subdirectories corresponding to different programming languages. Choose one as you like. Or, port to a new language (see page 1 of the assignment).

   Each language-specific subdirectory contains an `INSTRUCTIONS` file explaining how to get the program running. Those instructions will let you automatically compute the $\log_2$-probability of three sample files (`sample1`, `sample2`, `sample3`). Try it![7]

   Next, you should spend a little while looking at those sample files yourself, and in general, browsing around the `/usr/local/data/cs465/hw2` directory to see what's there. There are corpora for three tasks: language identification, spam detection, and speech recognition. Each corpus has already been divided into training, development ("held-out"), and test data, and also has a `README` file that you should look at.

   If a language model is built from the `switchboard-small` corpus, using add-0.01 smoothing, what is the model's *perplexity per word* on each of the three sample files? (You can compute this from the $\log_2$-probability that `fileprob` prints out, as discussed in class and in your textbook. Use the command `wc -w` on a file to find out how many words it contains.)

   What happens to the $\log_2$-probabilities and perplexities if you train instead on the larger `switchboard` corpus? Why?

7. Modify `fileprob` to obtain a new program `textcat` that does text categorization. See the `INSTRUCTIONS` file for programming-language-specific directions about which files to copy, alter, and submit for this problem.

   `textcat` should be run from the command line exactly like `fileprob`, except that the command line should specify *two* training corpora rather than 1: *train1* and *train2*.

   `textcat` should classify each file f listed on the command line: that is, it should print the name of the training corpus (*train1* or *train2*) that yields the higher value of $p(f)$. Finally, it should summarize by printing the percentage of files classified each way.

   Sample input (please allow this format; `gen` and `spam` are the training corpora, corresponding to "genuine" and spam emails):

[7] There is one little problem with our setup. To simplify the command-line syntax, I've assumed that the training corpus comes in a *single* big file. That means that there is only one `<s>` and one `</s>` in the whole training corpus—whereas `<s>` and one `</s>` are much more common in test data. This is a case of *domain mismatch*, where the training data is somewhat unlike the test data. Domain mismatch is a common source of errors in applied NLP. In this case, the only practical implication is that your language models won't do a very good job of modeling what tends to happen at the very start or very end of a file.

```
textcat add1 gen spam foo.txt bar.txt baz.txt
```

Sample output (please use this format; send any tracing output to the stderr stream):

```
spam    foo.txt
spam    bar.txt
gen     baz.txt
1 looked more like gen (33.33%)
2 looked more like spam (66.67%)
```

Use add1 smoothing as shown above.

As discussed earlier, both language models built by `textcat` should use the same finite vocabulary. Define this vocabulary to consist of all words that appeared in *either* training corpus (i.e., the union of two sets), plus OOV. Your model doesn't actually need to store the set of words in the vocabulary, but it does need to know its size V, because the add-1 smoothing method estimates $p(z \mid xy)$ as $\frac{c(xyz)+1}{c(xy)+V}$. We've provided code to find V for you—see the `INSTRUCTIONS` file for details.

8. In this question, you will evaluate your `textcat` program on ONE of two problems. You can do either language identification (the `english_spanish` directory) or else spam detection (the `gen_spam` directory). Have a look at the development data in both directories to see which one floats your boat. **(Don't peek at the test data!)**

   (It may be convenient to use symbolic links to avoid typing long filenames. E.g., `ln -s /usr/local/data/cs465/hw2/english_spanish/train ~/estrain` will create a subdirectory `estrain` under your home directory; this subdirectory is really just a shortcut to the official training directory.)

   Run `textcat` on all the development data for your chosen problem:

   - For the language ID problem, classify the files `english_spanish/dev/english/*/*` using the training corpora `en.1K` and `sp.1K`.
     Then classify `english_spanish/dev/spanish/*/*` similarly. Note that for this corpus, the "words" are actually letters.
   - Or, for the spam detection problem, classify the files `gen_spam/dev/gen/*` using the training corpora `gen` and `spam`.
     Then classify `gen_spam/dev/spam/*` similarly.

   From the results, you should be able to compute a total error rate for the technique: that is, what percentage of the test files were classified incorrectly?

   Now try add-λ smoothing for $\lambda \neq 1$. Experiment by hand with different values of $\lambda > 0$. (You'll be asked to discuss in 9b why $\lambda = 0$ probably won't work well.)

(a) What is the lowest error rate you could achieve on development data?

(b) What value of λ gave you that rate? Call this λ_0: for simplicity, you will use $\lambda = \lambda_0$ throughout the rest of this assignment.

(c) Using add-λ_0 smoothing, what is the error rate on test data? (Before now, you should not have done anything with the test data!)

(d) Each of the development and test files has a length. For language ID, the length in characters is given by the directory name and is also embedded in the filename (as the first number). For spam detection, the length in words is embedded in the filename (as the first number).

Using your results from problem 8a, come up with some way to quantify or graph the relation between development file length and classification accuracy. (Feel free to use the class mailing list to discuss how to do this.) Write up your results.

You may find the `xgraph` utility useful; it is very easy to use. Type `man xgraph` for documentation. To include a graph in your writeup, just give us instructions about how to see it on the ugrad machines (e.g., `xgraph < mydatafile` or `gv mygraph.ps`). A more powerful choice would be gnuplot: see http://comp.ling.utexas.edu/wiki/doku.php/tips_and_tricks#graphing.

(e) Now try increasing the amount of *training* data. Compute the overall error rate on development data for training sets of different sizes. Graph the training size versus classification accuracy.

- For the language ID problem, use training corpora of 6 different sizes: `en.1K` vs. `sp.1K` (1000 characters each); `en.2K` vs. `sp.2K` (2000 characters each); and similarly for `5K`, `10K`, `20K`, and `50K`.
- Or, for the spam detection problem, use training corpora of 4 different sizes: `gen` vs. `spam`; `gen-times2` vs. `spam-times2` (twice as much training data); and similarly for `...-times4` and `...-times8`.

Here are the smoothing techniques we'll consider:

uniform distribution (UNIFORM) $p(z \mid xy)$ is the same for every xyz; namely,

$$p(z \mid xy) = 1/V$$

where V is the size of the vocabulary *including* OOV.

add-λ (ADDL) Add a constant $\lambda \geq 0$ to every trigram count $c(xyz)$:

$$p(z \mid xy) = \frac{c(xyz) + \lambda}{c(xy) + \lambda V}$$

where V is defined as above. (Observe that $\lambda = 1$ gives the add-one estimate. And $\lambda = 0$ gives the naive historical estimate $c(xyz)/c(xy)$.)

add-λ backoff (BACKOFF_ADDL) Suppose both z and z' have rarely been seen in context xy. These small trigram counts are unreliable, so we'd like to rely largely on backed-off bigram estimates to distinguish z from z':

$$p(z \mid xy) = \frac{c(xyz) + \lambda V \cdot p(z \mid y)}{c(xy) + \lambda V}$$

where $p(z \mid y)$ is a backed-off bigram estimate, which is estimated recursively by a similar formula. (If $p(z \mid y)$ were the UNIFORM estimate $1/V$ instead, this scheme would be identical to ADDL.)

So the formula for $p(z \mid xy)$ backs off to $p(z \mid y)$, whose formula backs off to $p(z)$, whose formula backs off to ... what??

Witten-Bell backoff (BACKOFF_WB) As mentioned in class, this is a backoff scheme where we explicitly reduce ("discount") the probabilities of things we've seen, and divide up the resulting probability mass among only the things we haven't seen.

$$\begin{aligned}
p(z \mid xy) &= \begin{cases} p_{\text{disc}}(z \mid xy) & \text{if } c(xyz) > 0 \\ \alpha(xy)p(z \mid y) & \text{otherwise} \end{cases} \\
p(z \mid y) &= \begin{cases} p_{\text{disc}}(z \mid y) & \text{if } c(yz) > 0 \\ \alpha(y)p(z) & \text{otherwise} \end{cases} \\
p(z) &= \begin{cases} p_{\text{disc}}(z) & \text{if } c(z) > 0 \\ \alpha() & \text{otherwise} \end{cases}
\end{aligned}$$

Some new notation appears in the above formulas. The *discounted probabilities* p_{disc} are defined by using the Witten-Bell discounting technique:

$$\begin{aligned}
p_{\text{disc}}(z \mid xy) &= \frac{c(xyz)}{c(xy) + T(xy)} \\
p_{\text{disc}}(z \mid y) &= \frac{c(yz)}{c(y) + T(y)} \\
p_{\text{disc}}(z) &= \frac{c(z)}{c() + T()}
\end{aligned}$$

where

- $T(xy)$ is the number of different word types z that have been observed to follow the context xy
- $T(y)$ is the number of different word types z that have been observed to follow the context y
- $T()$ is the number of different word types z that have been observed at all (this is the same as V except that it doesn't include OOV)
- $c()$ is the number of tokens in the training corpus, otherwise known as N.

Given all the above definitions, the values $\alpha(xy)$, $\alpha(y)$, and $\alpha()$ will be chosen so as to ensure that $\sum_z p(z \mid xy) = 1$, $\sum_z p(z \mid y) = 1$, and $\sum_z p(z) = 1$, respectively.

Note: Numerous other smoothing schemes exist. In past years, for example, our course assignments have used Katz backoff with Good-Turing discounting. (We discussed Good-Turing in class: it is conceptually beautiful but a bit tricky in practice.) Another beautiful approach is MacKay-Peto smoothing. The most popular scheme nowadays is something called modified Kneser-Ney, or a more principled Bayesian formulation of it based on the Pitman-Yor process.

---

9. (a) Above, V is carefully defined to include OOV. So if you saw 19,999 different word types in training data, then $V = 20,000$. What would go wrong with the UNIFORM estimate if you mistakenly took $V = 19,999$? What would go wrong with the ADDL estimate?

   (b) What would go wrong with the ADDL estimate if we set $\lambda = 0$? (Remark: This naive historical estimate is commonly called the *maximum-likelihood estimate*, because it maximizes the probability of the training corpus.)

   (c) If $c(xyz) = c(xyz') = 0$, than what are the BACKOFF_ADDL estimates of $p(z \mid xy)$ and $p(z' \mid xy)$? What are they if $c(xyz) = c(xyz') = 1$?

   (d) In the BACKOFF_ADDL scheme, how does increasing λ affect the probability estimates? (Think about your answer to the previous question.)

10. The code provided to you implements some smoothing techniques, but others are left for you to implement – currently they just trigger error messages. Let's fix that!

    (a) Implement add-λ smoothing with backoff. See the `INSTRUCTIONS` file for language-specific instructions about which files to modify and submit.

    This should be just a few lines of code. You will only need to understand how to look up counts in the hash tables. Just study how the existing methods do it.

    Hint: So $p(z \mid xy)$ should back off to $p(z \mid y)$, which should back off to $p(z)$, which backs off to ... what???

(b) How does this smoothing method (with $\lambda = \lambda_0$) affect your error rate when you repeat the text categorization test of problem 8c?

(c) *Extra credit:* Use development data to find a λ_1 that works better than λ_0 with this new smoothing method. How much better does it do on test data? Is λ_1 bigger or smaller than λ_0? Why?

11. Now we turn to Witten-Bell backoff. It's conceptually pretty simple. The tricky part will be finding the right α values to make the probabilities sum to 1.

(a) Witten-Bell discounting will discount some probabilities more than others. When is $p_{\text{disc}}(z \mid xy)$ very close to the naive historical estimate $c(xyz)/c(xy)$? When is it far less (i.e., heavily discounted)? Give a practical justification for this policy.

(b) What if we changed the Witten-Bell discounting formulas to make all T values be zero? What would happen to the discounted estimates? What would the α values have to be, in order to make the distributions sum to 1?

(c) Observe that the set of zero-count words $\{z : c(z) = 0\}$ has size $V - T()$.[8] What is the simple formula for $\alpha()$?

(d) Now let's consider $\alpha(xy)$. Let $Z(xy)$ be the set $\{z : c(xyz) > 0\}$. Observe that

$$\begin{aligned}\sum_z p(z \mid xy) &= \left(\sum_{z \in Z(xy)} p_{\text{disc}}(z \mid xy)\right) + \left(\alpha(xy) \cdot \sum_{z \notin Z(xy)} p(z \mid y)\right) \\ &= \left(\sum_{z \in Z(xy)} p_{\text{disc}}(z \mid xy)\right) + \alpha(xy) \cdot \left(1 - \sum_{z \in Z(xy)} p(z \mid y)\right)\end{aligned}$$

To make $\sum_z p(z \mid xy) = 1$, solving the equation shows that you will need[9]

$$\alpha(xy) = \frac{1 - \sum_{z \in Z(xy)} p_{\text{disc}}(z \mid xy)}{1 - \sum_{z \in Z(xy)} p(z \mid y)}$$

Got that? Now, the first step in the derivation above assumed that $\sum_z p(z \mid y) = 1$. Give a formula for $\alpha(y)$ that ensures this. The formula will be analogous to the one we just derived for $\alpha(xy)$. (*Hint:* Start by defining the set $Z(y)$.)

[8] You might think that this set is just {OOV}, but that depends on how the finite vocabulary was chosen. There might be other zero-count words as well: this is true for your `gen` and `spam` (or `english` and `spanish`) models, since the vocabulary is taken from the union of both corpora. Conversely, it is possible for $c(\text{OOV}) > 0$, since in general one might decide to omit rarely observed words from one's vocabulary, treating them as OOV when they appear in training.

[9] Should we worry about division by 0 (in which case the equation has no solution)? Since $p(z \mid y)$ is smoothed to be > 0 for all z, this problem occurs if and only if *every* z in the vocabulary, including OOV, has appeared following xy. Fortunately, you defined the vocabulary to include all words that were actually observed, so no OOV words can ever have appeared following xy. So the problem cannot occur for you.

(e) Finally, we should figure out how the above formula for $\alpha(xy)$ can be *computed efficiently*. Smoothing code can take up a lot of the execution time. It's always important to look for ways to speed up critical code—in this case by cleverly rearranging the formula. The slow part is those two summations ...

i. Simplify the subexpression $\sum_{z \in Z(xy)} p_{\text{disc}}(z \mid xy)$ in the numerator, by using the definition of p_{disc} and any facts you know about $c(xy)$ and $c(xyz)$. You should be able to eliminate the $\sum$ sign altogether.

ii. Now consider the $\sum$ sign in the denominator. Argue that $c(yz) > 0$ for every $z \in Z(xy)$. That allows the following simplification: $\sum_{z \in Z(xy)} p(z \mid y) = \sum_{z \in Z(xy)} p_{\text{disc}}(z \mid y) = \frac{\sum_{z \in Z(xy)} c(yz)}{c(y)+T(y)}$.
(*Warning*: You can't use this simplification when it leads to 0/0. But in that special case, what can you say about the context xy? What follows about $\alpha(xy)$?)

iii. The above simplification still leaves you with a sum in the denominator. But you can compute this sum efficiently in advance.
Write a few lines of pseudocode that show how to compute $\sum_{z \in Z(xy)} c(yz)$ for every observed bigram xy. You can compute and store these sums immediately *after* you finish reading in the training corpus. At that point, you will have a list of trigrams xyz that have actually been observed (the provided code helpfully accumulates such a list for you), and you will know $c(yz)$ for each such trigram.
Armed with these sums, you will be able to compute $\alpha(xy)$ in $O(1)$ time when you need it during testing. You should not have to do any summation during testing.

> *Remark:* Of course, another way to avoid summation during testing would be for training to precompute $p(z \mid xy)$ for all possible trigrams xyz. However, since there are V^3 possible trigrams, that would take a lot of time and memory. Instead, you'd like training for an n-gram model to only take time about proportional to the number of *tokens* N in the training data (which is usually far less than V^n, and does not grow as you increase n), and memory that is about proportional to the number of n-gram *types* that are actually observed in training (which is even smaller than N). That's what you just achieved by rearranging the computation.

(f) Explain how to compute the formula for $\alpha(y)$ efficiently. Just use the same techniques as you did for $\alpha(xy)$ above. This is easy, but it's helpful to write out the solution before you start coding.

12. Now implement Witten-Bell backoff using the techniques above. How does this smoothing method (which does not use any λ) affect your error rate when you repeat the text categorization test in 8c?

The `INSTRUCTIONS` file gives language-specific instructions about which files to modify and submit.

> *Hint:* Here are two techniques to check that you are computing the α values correctly:
>
> - Write a loop that checks that $\sum_z p(z \mid xy) = 1$ for all x, y. (This check will slow things down since it takes $O(V^3)$ time, so only use it for testing and debugging.)
> - Use a tiny 5-word training corpus. Then you will be able to check your smoothed probabilities by hand.

13. Suppose you expect *a priori* that $\frac{1}{3}$ of your test emails will be spam. (In fact this is true for the test and development data!)

    Or, in the language ID task, assume all the documents you view are in either Spanish or English, and you expect *a priori* that $\frac{1}{3}$ of them will be in Spanish.

    How should this affect how `textcat.c` does its classification, and why? (Just give a formula, don't implement it.) *Hint:* Bayes' Theorem.

    Do you need to know the number $\frac{1}{3}$ when you train the language model? Or is it only used at test time, so that each individual user could adjust it at runtime (without having to retrain) to match the fraction of spam that they *actually* currently get?

    Extra credit: Implement this change and measure how it affects performance on the spam detection task with various smoothing methods (since, in fact, $\frac{1}{3}$ of the test data for that task *are* spam, making the *a priori* expectation correct). Does it help? Why or why not? What happens for values other than $\frac{1}{3}$ (perhaps try adjusting the prior gradually from 0 to 1)?

14. We now turn to speech recognition. Here, instead of choosing the best model for a given string, you will choose the best string for a given model.

    The data are in the `speech` subdirectory. As usual, a development set and a test set are available to you; you may experiment on the development set before getting your final results from the test set. You should use the `switchboard` corpus as your training.

    Here is a sample file (`dev/easy/easy025`):

```
8       i found that to be %hesitation very helpful
0.375   -3524.81656881726       8       <s> i found that the uh it's very helpful </s>
0.250   -3517.43670278477       9       <s> i i found that to be a very helpful </s>
0.125   -3517.19721540798       8       <s> i found that to be a very helpful </s>
0.375   -3524.07213817617       9       <s> oh i found out to be a very helpful </s>
0.375   -3521.50317920669       9       <s> i i've found out to be a very helpful </s>
0.375   -3525.89570470785       9       <s> but i found out to be a very helpful </s>
```

```
0.250   -3515.75259677371       8       <s> i've found that to be a very helpful </s>
0.125   -3517.19721540798       8       <s> i found that to be a very helpful </s>
0.500   -3513.58278343221       7       <s> i've found that's be a very helpful </s>
```

Each file has 10 lines and represents a single audio-recorded utterance U. The first line of the file is the correct transcription, preceded by its length in words. The remaining 9 lines are some of the possible transcriptions that were considered by a speech recognition system—including the one that the system actually chose to output. You will similarly write a program that chooses among those 9 candidates.

Consider the last line of the sample file. The line shows a 7-word transcription $\vec{w}$ surrounded by `<s>`...`</s>` and preceded by its length, namely 7. The number -3513.58 was the speech recognizer's estimate of $\log_2 p(U \mid \vec{w})$: that is, if someone really were trying to say $\vec{w}$, what is the log-probability that it would have come out of their mouth sounding like U?[10] Finally, $0.500 = \frac{4}{8}$ is the **word error rate** of this transcription, which had 4 errors against the 8-word true transcription on the first line of the file.[11]

(a) According to Bayes' Theorem, how should you choose among the 9 candidates? That is, what quantity are you trying to maximize, and how should you compute it?

(*Hint:* You want to pick a candidate that both looks like English and looks like the audio utterance U. Your trigram model tells you about the former, and -3513.58 is an estimate of the latter.)

(b) Modify `fileprob` to obtain a new program `speechrec` that chooses this best candidate. As usual, see `INSTRUCTIONS` for details.

The program should look at each utterance file listed on the command line, choose one of the 9 transcriptions according to Bayes' Theorem, and report the word error rate of that transcription (as given in the first column). Finally, it should summarize the overall word error rate over all the utterances—the *total* number of errors divided by the *total* number of words in the correct transcriptions.

Of course, the program is not allowed to cheat: when choosing the transcription, it must ignore each file's first row and first column!

Sample input (please allow this format; `switchboard` is the training corpus):

[10] Actually, the real estimate was 15 times as large. Speech recognizers are really rather bad at estimating $\log p(U \mid \vec{w})$, so they all use a horrible hack of dividing this value by about 15 to prevent it from influencing the choice of transcription too much! But for the sake of this question, just pretend that no hack was necessary and -3513.58 was the actual value of $\log_2 p(U \mid \vec{w})$ as stated above.

[11] The word error rate of each transcription has already been computed by a scoring program. The correct transcription on the first line sometimes contains special notation that the scorer paid attention to. For example, `%hesitation` on the first line told the scorer to count either `uh` or `um` as correct.

```
speechrec add1 switchboard easy025 easy034
```

Sample output (please use this format—but you are not required to get the same numbers):

```
0.125   easy025
0.037   easy034
0.057   OVERALL
```

Notice that the overall error rate 0.057 is not an equal average of 0.125 and 0.037; this is because `easy034` is a longer utterance and counts more heavily.

Hints about how to read the file:

- For all lines but the first, you should read a few numbers, and then as many words as the integer told you to read. (Alternatively, you could read the whole line at once and break it up into an array of whitespace-delimited strings.)
- For the first line, you should read the initial integer, then read the rest of the line. The rest of the line is only there for your interest, so you can throw it away. The scorer has already considered the first line when computing the scores that start each remaining line.
  Warning: For the first line, the notational conventions are bizarre, so in this case the initial integer *does not necessarily tell you* how many whitespace-delimited words are on the line. Be sure to throw away the rest of the line anyway! (If necessary, read and discard characters up through the end-of-line symbol `\n`.)

(c) What is your program's overall error rate on the carefully chosen utterances in `test/easy`? How about on the random sample of utterances in `test/unrestricted`? Answer for 3-gram, 2-gram, and 1-gram models.

To get your answer, you need to choose a smoothing method, so pick one that seems to work well on the development data `dev/easy` and `dev/unrestricted`. Be sure to tell us which method you picked and why! What would be an *unfair* way to choose a smoothing method?

Hint: Some options for handling the 2-gram and 1-gram models:

- You'll already have a `probs`(x, y, z) function. You could add `probs`(y, z) and `probs`(z).
- You could give `probs`(x, y, z) an extra argument that controls which kind of model it computes. For example, for a 2-gram model, it would ignore x.
- Or you could make life easy for yourself, and just call the existing `probs()` function with arguments that will make it return a bigram or unigram probability. For example, if you choose a backoff smoothing method, then $p(z \mid \text{OOV}, y)$ will back off completely to $p(z \mid y)$, which is the bigram probability that you want!

15. *Extra credit:* Problem 11a asked you to justify Witten-Bell discounting. Suppose you redefined $T(xy)$ in Witten-Bell discounting to be the number of word types z that have been observed *exactly once* following xy in the training corpus. What is the intuition behind this change? Why might it help (or hurt, or not matter much)? If you dare, try it out and report how it affects your results.

16. *Extra credit:* In our text categorization setup, a problem with all the smoothing methods is that they treat OOV just like any other zero-count word. Let's focus here on Witten-Bell backoff.

    For example, suppose the only zero-count words in the `spam` model are `snuffleupagus`, `grover`, and OOV. (The first two are zero-count because they appeared in the `gen` corpus only.) Then these three words will share the probability mass equally: $p(\texttt{snuffleupagus}) = p(\texttt{grover}) = p(\text{OOV}) = \alpha()$. But OOV represents the total class of *all* out-of-vocabulary words, so surely it should have much higher probability than one measly `snuffleupagus` or `grover`.

    If the `gen` corpus increased in size to add three more zero-count words to the vocabulary, then the `spam` model's estimates of $p(\texttt{snuffleupagus})$, $p(\texttt{grover})$, and $p(\text{OOV})$ would all have to be cut in half, since the same probability mass reserved for zero-count words would be shared equally among 6 words instead of 3. Such huge changes in the `spam` model's probability estimates are not warranted here. Surely it would be more accurate if $p(\text{OOV})$ had almost all of this probability mass to start with, and gave up tiny bits of it as additional words entered the vocabulary and left the OOV class.

    The following fix might help substantially, and I'd be very interested to know whether it improves the Witten-Bell text categorization results! The insight is that to estimate the relative probabilities in future spam of the 0-count words `snuffleupagus` (which appeared in `gen`) and OOV (which did not appear in `gen`), we can look at the relative probabilities in `spam` of *other* words that did and didn't appear in `gen`. In other words, we should measure the overlap of the two corpora.

    We are building a model for spam. Suppose there are b word types that appeared once in the `spam` training corpus, and a of those also appeared (at least once) in the `gen` training corpus. Since all 1-count types get equal unigram probability under Witten-Bell, we see that the total probability mass for the 1-count spam words is divided, with a/b of it shared equally among words that appeared in `gen`, and $1 - a/b$ of it shared equally among words that did not.[12] Assume that this ratio measured on 1-count spam words is also correct for 0-count spam words. Therefore, a/b of the total

[12] One could extend this idea to determine how the mass is divided *three* ways, among words that appeared 0, 1, ≥ 2 times in `gen`.

probability mass reserved for 0-count types in future spam (namely $T()/(c() + T())$; see question 11c) should be shared equally among the $(V - 1) - T()$ 0-count spam words that were specifically seen in `gen`, such as `snuffleupagus` and `grover`. The remaining $1 - a/b$ of this mass should go to the 0-count spam words that were never seen in `gen` either, which are collectively represented by the OOV symbol.

Here we are using the observed behavior of 1-count types to predict the unobserved behavior of 0-count types, on the grounds that rare words behave alike. There are entire smoothing methods (such as Good-Turing smoothing and Katz backoff) that are based on that idea, using the observed rate of 1-count types in a context to predict the true rate of 0-count types in that context. You can think of Witten-Bell as an approximation to those smoothing methods—in place of the number of 1-count types in a context xy, it just uses the total number of types $T(xy)$ in that context, which in practice is not much greater since most observed types only appear once.

If you find it easier, you can use the same kind of approximation for this problem: you can approximate the fraction a/b of `spam` 1-count types whose types also appeared in `gen` by the fraction $I/T()$ of *all* `spam` types that also appeared in `gen`. Here $T()$ is the total number of types in `spam`, as usual, and I denotes the total number of types in the intersection `gen` $\cap$ `spam`. To implement this method, you will have to compute I. You will also have to maintain a table of words in the vocabulary so that when you see a 0-count word in testing, you know whether it is in-vocabulary or OOV. Both can be accomplished by tinkering with the code that computes V.

17. *Extra credit:* We have been assuming a finite vocabulary by replacing all unknown words with a special OOV symbol. But notice that if the *alphabet* is finite, you could predict the probability of an unknown word by using ... you got it, a letter n-gram model! Such a prediction is sensitive to the spelling and length of the unknown word. As longer words will generally receive lower probabilities, it is possible for the probabilities of all unknown words to sum to 1, even though there are infinitely many of them. (Just as $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \cdots = 1$.)

    Devise a sensible way to estimate the word trigram probability $p(z \mid xy)$ by backing off to a letter n-gram model of z if z is an unknown word. Also describe how you would train the letter n-gram model.

    Just give the formulas for your estimate—you don't have to implement and test your idea, although that would be nice too!

    Notes:

    - x and/or y and/or z may be unknown; be sure you make sensible estimates of $p(z \mid xy)$ in all these cases
    - be sure that $\sum_z p(z \mid xy) = 1$