Official answers to the spring 2016 NLP midterm.

1a. increase, decreasing

Backoff always decreases variance while increasing bias.
"Looking where the light is better" will give you a reliable
(low-variance) estimate of the wrong (biased) probability.

In the trigram case, you really want p(z | xy), but you settle for
p(z | y), which collects more data but the data is not quite what
you want.

1b. increase, decreasing

Again, you are collecting more data but the data is not quite
what you want.

1c. development data
1d. uniform
1e. Concatenating the corpora and using add-one smoothing would
estimate p(z | xy) as

$$(c_1(xyz)+c_2(xyz)+1) / (c_1(xy)+c_2(xy)+V)$$

You could weight the two corpora evenly by scaling them to be the
same size:

$$(c_1(xyz) + (N_1/N_2)\, c_2(xyz) + 1) / (c_1(xy) + (N_1/N_2)\, c_2(xy) + V)$$

To weight them unevenly, bring in a factor of alpha:

$$(\alpha\, c_1(xyz) + (1-\alpha)\, (N_1/N_2)\, c_2(xyz) + 1) / (c_1(xy) + (N_1/N_2)\, c_2(xy) + V)$$

(Note: After these reweightings, the effective size of the
combined corpus has now been reduced to N_1 rather than N_1+N_2.
So perhaps you shouldn't do add-1 smoothing in this case, but
add-(N_1/(N_1+N_2)) smoothing. In practice, the best option
is still to do add-lambda smoothing and tune lambda on dev data.)

1f. You could use a log-linear model.

The features might include binary features that fire on
particular unigrams z, bigrams yz, and trigrams xyz --
as well as versions of all these features that fire only
on text where Speaker=doctor.

So, some of these features are specific to helping you model
hospital data, but others are like "backoff features" that
are SHARED between hospital data and chat data. Formally,
there is no difference between backing off from

```
xyz=="to the patient && Speaker==doctor
```

to

```
yz=="the patient && Speaker==doctor
```

and to

```
xyz=="to the patient"
```

For example, here are plausible weights for 6 features:

```
 +0.5    yz=="the patient"
 +2.0    yz=="the patient"
 +2.0   xyz=="at the time"
 +0.0   xyz=="at the time" && Speaker==doctor
 -0.1   xyz=="showed gross total"
 +1.8   xyz=="showed gross total" && Speaker==doctor
```

This indicates:

p(patient | <anything> the) is moderately common in the general

population but much higher for doctors.

p(time | at the) is high for everyone and no different for doctors.

p(total | showed gross) is high for doctors, even though it is not high in the general population.

This will probably work quite nicely. To understand how this approach interacts with regularization, see the paper "Frustratingly Easy Domain Adaptation."

Another approach would be to do ordinary backoff smoothing, but to back off from

```
p(z | xy=="to the" && Speaker==doctor)
```

to a LINEAR COMBINATION of

```
p(z | y=="the" && Speaker==doctor)
```

and

```
p(z | xy=="to the")
```

2a. "THERE is another way ..."
2b. "The rephrased version often seems EASIER to understand."

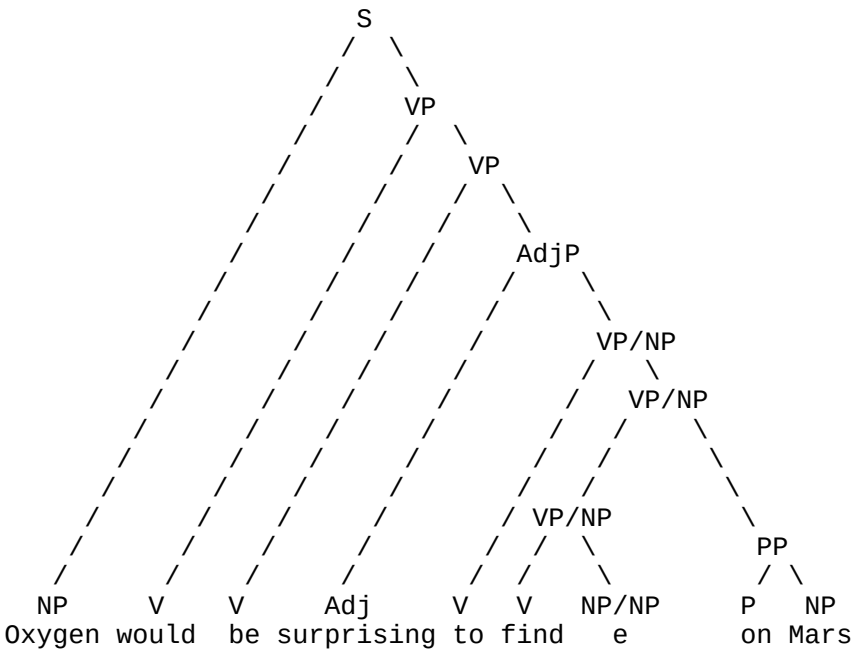
2c. The crucial point: chew, find, attempt, and veto are all transitive verbs. In the example sentences, however, their direct objects do not appear in the usual position. Rather, they have "moved" to the subject position of the sentence.

"Movement" is generally handled by slashed categories. So you should expect to see slashed categories in this answer.

The basic idea of the following tree is that an Adj like "tough" or "surprising" can take a VP/NP as its argument. Recall that a VP/NP is a verb phrase with a missing object.

So, "surprising to find on Mars" is now an AdjP that can be predicated of any NP, such as "oxygen." Notice that it is essentially interchangeable with other AdjPs, like the AdjP "scarce" in "Oxygen is scarce." This is how we know it is AdjP and not AdjP/NP.

"Scarce" and "surprising to find on Mars" are both predicates that are being used to describe "oxygen". In the latter case, the semantics of "surprising to find on Mars" will have to be an interesting lambda-term. It will end up delivering "oxygen" into the "findee" role that would ordinarily be filled by the direct object of "find". This lambda-term must be constructed by applying an interesting lambda-term for "surprising" to the standard lambda-term for the VP/NP "to find on Mars."



The attributes on the VP nodes, from the top down, should be:
   VP[tense=cond,head=be] : would be surprising to find e on Mars
   VP[tense=stem,head=be] : be surprising to find e on Mars
   VP/NP[tense=inf,head=find] : to find e on Mars
   VP/NP[tense=stem,head=find] : find e on Mars
   VP/NP[tense=stem,head=find] : find e

Notice that there are two main verbs here: "be" and "find".
Sometimes "be" is used as an auxiliary verb ("you are finding"),
but not here. It is just a verb of existence.

Notice also that "surprising" is not a present participle
(as in "you are finding"). We've already established that it
is acting as an adjective like "tough": "the surprising fact".

(It is true, however, that this adjective is originally derived
from the verb "surprise", which can take a plain VP argument:
"It surprised us to find oxygen on Mars.")

Discussion: So what's going on with the various "transformations"?

Remember that both of these sentences are grammatical:
    (1) Oxygen would be [surprising to find on Mars].
    (2) It would be [surprising to find oxygen on Mars].

What do we make of the second version? There, "find" has its
object back. So "surprising" must be able to combine not only
with a VP/NP, but also with a plain VP. (Just as the verb
"surprise" can -- see above.)

However, the resulting AdjP must have a different (and unusual)
semantic type. It doesn't apply to an NP as most AdjPs do:
    *Nitrogen would be [surprising to find oxygen on Mars].

Thus, [surprising to find oxygen on Mars] is not an ordinary AdjP.
Semantically, it is like a complete though untensed sentence, that
is, a predicate on events. And as a result, it must have a
syntactic attribute that says it can't combine with a subject like
"nitrogen", but ONLY with the dummy subject "it":
    It was [surprising to find oxygen on Mars].
Note that the tense information is added by the linking verbs, such as

"would be" or "was."

The start of the question pointed out another English construction
that allows the adjective "surprising" to get its VP argument:
   (3) To find oxygen on Mars would be surprising.
Semantically, this must mean the same thing as (2):
   (2) It would be [surprising to find oxygen on Mars].

But it is not so strange that both forms (2) and (3) exist.
After all, ordinary adjectives that apply to NPs
can also get their NP argument in two ways:
    This is a heavy skillet. (like "It is surprising to find oxygen")
    This skillet is heavy.   (like "To find oxygen is surprising")
In the first case, the Adj combines syntactically with its argument.
In the second case, a linking verb combines syntactically with
the Adj and then with its argument (which is the subject of the
linking verb). The semantics of the linking verb then ensure
that the Adj is applied to its argument.

Another remark:

When we combine Adj "surprising" with VP/NP "to find e on Mars,"
the VP/NP is not only missing its object. Like all VPs, it is
also missing its subject.

But actually, it is possible to supply a subject in
tough-movement, too: "Oxygen would be surprising FOR ME to find on
Mars." This implies that "surprising" can alternatively take a
CP/NP as its argument, where "for" is the complementizer. That
would have to be handled by another grammar rule.

3a. 3, so it can reward "played the bugle"
3b. 2, so it can punish "the beguile"
3c. A PCFG with head attributes. Then there can be a rule like
        VP[head=played] -> V[head=played] NP[head=bugle]

A simple rule like
   VP -> V NP
misses the correlation between "played" and "bugle", because
the V and NP would rewrite *independently*.

3d. "beguile", since "... the beguile at ..." probably has no good parse.

4a. for k := 1 to n :
      for all rules X -> w_k in the grammar such that w_k is in W_k :
         chart[X,k-1,k] max= p(X -> w_k | X)

 Note that in the last line, we changed := to max=, which keeps
 a running maximum.
 For example, when k=7, we will get several competing N rules:
    N -> farm
    N -> firm
    N -> forum
    N -> frame
 As usual, we will eliminate duplicate "N" entries in the same
 cell of the chart. We set chart[N,6,7] to be the *maximum* of
 these rule probabilities.

4b. As usual, when an item's value is defined by max=, we use
backpointers to remember how the maximum value was constructed.

Usually in CKY, we only do this for cells of width >= 2,
by storing backpointers to the two narrower items that
combined to give this item.

But now the width-1 cell chart[X,k-1,k] is also created by max=.
Thus, we need to store a backpointer to the single word in W_k
(such as firm) that provided the maximum-probability derivation.

4c. No, <=.

The answer is <= because sentences are ambiguous. There may be several trees for the same sentence. The probability of a sentence is the total probability of all its trees, so each individual tree's probability is <= the sentence's probability.

As a result, the answer to the first part is "no."
It could be that p(w') > p(w) for some other sentence w'.
However, the probability mass p(w') is split evenly among many trees, while the probability mass p(w) is all on the tree shown here.
As a result, none of the w' trees are INDIVIDUALLY as probable as the tree shown here, which is why the parser found the tree shown here.

5. Note: This approach is called "direct prediction" of the answer we want.

Problem 4 basically defined a source model p(w) (the PCFG) and a noisy channel p(x | w) (deterministic vowel deletion) that corrupts w into x. Then it ran this process backwards using Bayes' Theorem via a parser.

However, problem 5 just deines p(w | x) in the first place. This is also called "discriminative modeling," because we will train the parameters of the model specifically to distinguish among the reconstructions of an injured sentence, rather than to generate a normal English sentence.

5a. There are 164640 "legal" candidates, as mentioned in the previous problem. These are versions of x with vowels added to make actual English words.

If you like, you could also consider candidates w that are not "legal" -- either because they are not consistent with x, or because their words are not in the English lexicon. However, then your answer to part (e) should have additional features to ensure that these illegal candidates have probability of 0 or close to 0. (Such features would presumably have weight -infinity or very negative weight.)

5b. Some (x,w) pairs, where x is an injured sentence and w is the original version that we are trying to restore.

You will presumably want to train the model on more than one pair!
So a reasonable notation would be (x_1,w_1), (x_2,w_2), ..., (x_n,w_n).

5c. Take a lot of normal English sentences w and injure them yourself!
It is easy enough to get large corpora of normal English text, e.g., by crawling the web.

5d. sum_i log p_theta(w_i | x_i).

This is called the log-likelihood of theta. It is the log of \prod_i p_theta(w_i | x_i), which is the likelihood of theta. (That is, the probability of generating the data if theta is correct.)

5e. One option, of course, is to say that u(x,w) is simply p(w) under a PCFG language model. This gets back exactly the solution in problem 4. It is even possible to regard this as a kind of log-linear model, though I won't give the details here.

However, if you don't want to run a parser, then you can use simpler features. We'll assume the standard form

u(x,w) = exp theta . f(x,w)

where f extracts a feature vector.

The crucial part of the answer is that you need features that evaluate whether w is a plausible English sentence (just as in problem 4). For example, you could write

u(x,w) = sum_i theta_{w_i}

where there is a separate feature weight for each word type. Thus, for the sentence in question 4(c),

u(x,w) = theta_the + theta_nicest + theta_linguist + ...

Note that theta_the gets added twice for this sentence; in general, you can think of f_the(w,x) = 2, the number of occurrences of "the" in w.

Those are unigram features. You could similarly add bigram features, e.g.,

u(x,w) = sum_i theta_{w_i} + theta_{w_{i-1},w_i}

Or your unigram and bigram features could be based on embeddings, e.g.,

u(x,w) = sum_i c . v(w_i) + v(w_{i-1})^T M v(w_i)

where theta consists of the entries of the vector c and the matrix M.

You don't really need any features that look at x, because the set of candidates w is already restricted to sentences that are legal given x (see part (a)).

6. (c) This is the definition of Good Turing.

7a. p(English=e | French=f)
= p(English=e) * p(French=f | English=e) / p(French=f),
where the denominator is a constant.

7b. Prior probability: p(English=e)
Posterior probability: p(English=e | French=f)
Likelihood: p(French=f | English=e)

7c. Noisy channel modeling.

8a. the baby on ...

You can see this by looking at the cases where the dot was successfully able to advance over a terminal symbol.

8b. 1 N -> baby .
0 NP -> DET N .
0 S -> NP . VP
0 NP -> NP . PP
2 VP -> . V NP
2 VP -> . VP PP
2 PP -> . P NP
2 V -> . ate
2 V -> . slept
2 V -> . pooped
2 P -> . of
2 P -> . on
2 P -> . with

8c. In column 0, the next word is "the." We will explicitly block all of the lexical rules except for DET -> . the and NP -> . the Holy Grail.

In column 1, the next word is "baby". We will explicitly block NP -> DET . ADJ N, since "baby" cannot start an ADJ. (At least not in this grammar -- as you can tell from the fact that ADJ -> . baby was not added. So forget about "baby carrots.")

This means that ADJ -> . big and ADJ -> . little never get created.

We also explicitly block N -> . adult, N -> . child, N -> . teenager, and NP -> the . Holy Grail. In all cases, the rule cannot continue if the next word is "baby" (since "baby" is not a left corner of the symbol after the dot).

In column 2, the next word is "on". We will explicitly block S -> NP . VP, since "of" cannot start a VP. This means that the VP and V rules in this column never get created. We also explicitly block P -> . of and P -> .with.

Official answers to the 2003 NLP midterm.

1. There are two events, win and lose:
   c(win) = 47
   c(lose) = 0
   Adding 1 to both, we have smoothed counts of 48 and 1,
   so the probability of winning is 48/49:

   p(win) = (47+1)/((47+1)+(0+1)) = 48/49.

   If you want to write out p(win | Billy) in terms of conditional
   probabilities, think of Billy as the context word. Billy has
   appeared 47 times, so c(Billy)=47. There are two possible words
   that could follow Billy's appearance, namely "win" and "lose."
   So we have
   p(win | Billy) = (c(Billy, win)+1) / (c(Billy)+V)
   = (47+1) / (47+2) = 48/49.

   (Mama's "odds of 47 to 1" is another way of describing a
   probability of 47/48, not 48/49. Mama's smoothing technique is
   apparently to change 0 counts to 1. However, that means that
   0-count events and 1-count events will end up with the same
   probability. So adding 1 to all counts seems to make a little more
   sense.)

2. (a) p(x | y,z) > p(x | y), by the defintion of conditional probability.
   This is POSSIBLE, since backing off can either raise or lower a
   conditional probability.

   (b) x is conditionally independent of z, given y.

3. We discussed a number of such sentences in class, in the
   psycholinguistics lecture and when dealing with parsing.
   For example:

   Put the apple on the towel ... in the box.
   The lawyer examined ... by the judge was tall.
   The horse raced past the barn ... fell.
   The government plans to raise income tax ... are unpopular.

   Each of these sentences is unambiguous when read in full. But the
   first part is ambiguous. Up until the "...", a reader is likely to
   get the wrong interpretation. The part after the "..." forces the
   reader to revise the earlier part of the parse.

4. One should lower the probability of
   VP[head=look] -> VP[head=look] PP[head=with]

   (or if you prefer attaching PP to S,
   as in the "time flies like an arrow" slides,
   lower the probability of
   S[head=look] -> S[head=look] PP[head=with])

   Some of you wanted to lower the probability of the competing rule,
   NP[head=dog] -> NP[head=dog] PP[head=with]
   But that would reinforce Jack's misunderstanding
   instead of correcting it. *Raising* the probability of this rule
   would work, though!

5. (a) This is TRUE.

   p(s1) = ... * p(they're | believe) * p(dog | they're) * ...
   p(s2) = ... * p(their | believe) * p(dog | their) * ...

   where the ... parts are identical between p(s1) and p(s2).

So p(s1) > p(s2) if and only if

  (they're | believe) * p(dog | they're)
      > p(their | believe) * p(dog | their)

which is equivalent to the inequality given on the test.

(b) The problem is that p(s3) is very large, since Hello! is a common utterance. Fortunately, p(s3) is just the prior probability of the utterance. A speech recognizer had better also pay attention to the acoustic information that shows up through its microphone!

The speech recognizer makes its decision according to the posterior probability
p(s3 | acoustics) = p(s3) * p(acoustics | s3) / p(acoustics)
(using Bayes' Theorem). That's what you did in homework 2.

So the correct answer is (C).

(A) is wrong because p(s3) is bigger than the others.
(B) is wrong because a large p(s3) would tend to make the recognizer choose s3, not reject it.
(D) is wrong because smoothing will not affect this problem.
(E) is wrong. We wouldn't bother building n-gram speech recognizers if they couldn't even tell the difference between "Hello!" and "I think they're dog tired." Anyway, a model that was conditioned on the social context might be a good thing in general, but in this example it would make the wrong answer ("Hello!") appear more likely, not less.

6. (a) Change NP -> Det Adj N to NP -> Det Adj+N and Adj+N -> Adj N.
Change VP -> V NP NP to VP -> V NP+NP and NP+NP -> NP NP.

(b) The slashed rules are derived from the original unslashed rules, by passing the /NP down from the parent to any one of the children that might contain an NP.

S/NP -> NP/NP VP
S/NP -> NP VP/NP
NP/NP -> <e> (where <e> denotes the empty string)
VP/NP -> V NP/NP
VP/NP -> V NP/NP NP (e.g., "feed <e> the donuts")
VP/NP -> V NP NP/NP (e.g., "feed Homer <e>")

7. (a) True. We have c(see the baby) / (c(see the) + T(see the))
= 0.5 * c(see the baby) / c(see the).
Therefore c(see the) = T(see the).

(b) True. Witten-Bell smoothing discounts p(z | x y) by a factor that depends only on xy and not on z. So p(kitty | see the) will be discounted by the same factor as p(baby | see the), namely 1/2.

Another way of getting the answer: Part (a) implies that "see the kitty" and "see the baby" have identical counts, so they will behave the same.

(c) False. But close.

Since all the observed trigrams of the form "see the x" will be discounted by half, we have extra probability mass 0.5 to redistribute in proportion to the backoff probabilities.

If we distributed the 0.5 mass among ALL words, then p(corpse | see the) would get 0.5 * its backoff probability, as stated. But in fact p(corpse | see the) will get more than that, because the 0.5 mass is only shared among words that were not seen in the context "see the."

8.

	head	number	tense
NP1	we	singular	N/A
NP2	trip	singular	N/A
VP1	win	singular	pres
VP2	win	N/A	perf
VP3	win	N/A	perf
V1	have	singular	pres
V3	win	N/A	perf

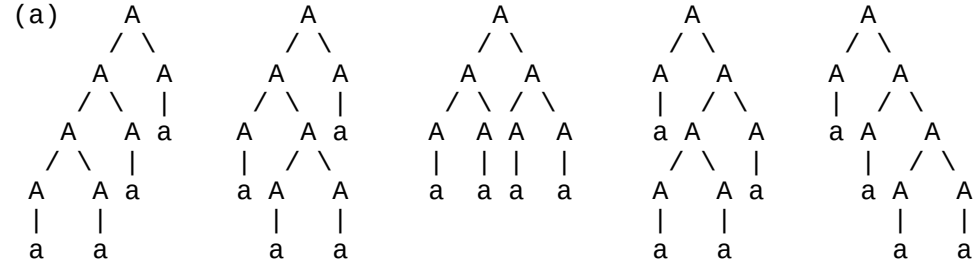
Note that the head of a word or phrase is its base form ("we," "have," "win" -- sometimes called the root or lemma), not the inflected form ("us", "has," "won"):

```
      V[head=have, num=singular, person=3rd, tense=pres] -> has
             ^^^^                                           ^^^
```

Although "us" is plural, "each of us" is singular, as you can tell by its agreement with the verb "have."

[Brock (the TA) asked me during grading: "How do you feel about 'I' being the head, instead of 'we'?" I said sure, he could take over the class!]

9. (a)



(b)

	1	2	3	4
0	A,1/2	A,1/8	A,1/32	A,1/128
1		A,1/2	A,1/8	A,1/32
2			A,1/2	A,1/8
3				A,1/2

Check: Each of the parses in part (a) uses 7 rules, so they each have probability (1/2)^7 = 128. This checks with the upper right corner of the chart.

A common mistake was forgetting to multiply in the probability of the rule.

A few of you treated 1/2 as a weight (i.e., negative log probability) to be added, instead of a probability to be multiplied. We didn't penalize heavily for this.

(c) No. We have 5 trees for this sentence. Each has the same probability 1/128 of being generated by randsent. So the inside probability is 5/128.

This isn't 1, because the inside probability is the probability that the sentence will be generated by randsent. Some probability has to be left over for sentences other than a a a a.

Some of you interpreted "the total probability of all parses" to mean the sum of p(parse | a a a a) over all parses. That

would indeed be 1. But that is not what the inside algorithm computes. In general, p(x) is the probability of tree x under the PCFG -- the probability that a run of randsent would generate tree x. The inside algorithm sums up p(x) where x ranges over all the parses of this sentence -- not over all trees in the world!

To put it more formally, we are looking at
sum_{x is a parse of a a a a} p(Tree=x),
not at
sum_{x is a parse of a a a a} p(Tree=x | sentence=a a a a).

Some other reasons you should suspect that the answer is "no":
- "If the inside algorithm always computed 1, we wouldn't have to run it."
- "If the input sentence didn't have a parse, the total probability of its parses couldn't be 1."

(d) The duplicates are
1 A -> A A .
0 A -> A A .
4 A -> . A A
4 A -> . a

The first two can be formed by different attachments.
For example, A -> A A . from 1 to 4 can be formed in two ways:
by extending A -> A . A from 1 to 2 with A -> A A . from 2 to 4
or by extending A -> A . A from 1 to 3 with A -> a . from 3 to 4.
(Doesn't that remind you of CKY?)
Similarly, A -> A A . can be formed in three ways.

The last two are PREDICTed by all the rules in the column of the form "i A -> A . A". Every time there's an A after the dot, you get A -> . A A and A -> . a, starting in that column.

10. (a) The cross-entropy is 25, so the cross-entropy per word is 25/5 = 5, so the perplexity per word is 2^5 or 32.

(b) To get substantial extra credit on this problem, you had to EXPLAIN the paradox. It's not enough to demonstrate that p(C) will in fact be higher under model B. You should recognize that this fact is deeply weird. It suggests that we should train our models on SMALLER corpora from now on, like B; but that's obviously the wrong conclusion, so you have to explain why it's not justified.

The key idea we were looking for: model A and model B have a different idea of what OOV is, so the two estimates of p(C) are not even considering the same set of events!

Some of you just pointed out that model B assigns more probability to OOVs than model A does. That is true but irrelevant, since only model B thinks the test words are OOVs. (The relevant comparison is whether model B assigns more probability to OOVs than model A assigns to real words.)

You also said that model B would therefore be better at dealing with unexpected stuff. So what? Only model B thinks the test data is unexpected. That's why we'd expect model A to do better. (The relevant comparison is whether model B is better at dealing with unexpected stuff than model A is at dealing with expected stuff.)

No, the problem is that fileprob does not really compute
p(i saw snuffleupagus on tv)

This was explained in homework 2:

The model trained on A will compute
   p(i saw OOV on tv)
which is the sum of p(i saw w on tv) for every word w that is outside that model's large vocabulary.

The model trained on B will compute
   p(OOV OOV OOV OOV OOV)
which is the sum of p(u v w x y) for all words u,v,w,x,y that are outside that model's small vocabulary.

So when trained on B, fileprob is actually printing the total probability of a much larger number of possible sentences! The probability is higher because it is the probability of a LARGER event set! The two models' estimates of p(C) are NOT COMPARABLE. That is why your homework 2 textcat program, which built two models from different corpora, was required to use the same definition of OOV for both models -- it wasn't ok to run fileprob twice.

Talking about the numbers didn't get full credit, but if you're interested:

Under add-1 smoothing, p(OOV) = 1/(N+V), where N is the number of training tokens and V is the size of the vocabulary (including OOV). If the training corpus is B, then N=10, V=9, so p(OOV)=1/19, which is extremely large. [I'm ignoring EOC here.]

The model trained on B will therefore assign probability 1/19 to every word in C, because it thinks they're all OOV. The model trained on A has a larger vocabulary, so it will assign small probabilities to all the words in C, both in-vocabulary words and "snuffleupagus."