Natural Language Processing (JHU 601.465/665)
Answers to "Probability Models" practice problems

1. a. Uniform distribution, meaning that all outcomes are equally probable.

Remark: You can think of this as a 0-gram model! In general, in an n-gram model, the probability of the ith character text[i-1:i] depends on the final n-gram text[i-n:i]. In a 0-gram model, it doesn't depend on anything at all, not even the ith character itself.

b. 4. Usually the answer would depend on the Tablish text. Some Tablish strings have higher surprisal per letter than others. But this particular model assigns exactly -log(1/16) = 4 bits to each letter, so it doesn't matter what the text is. (In information theory, logs are base 2 by convention.)

c. 2^4 = 16. Yes, this is related to the 1/16 probability.

d. infinity.
That's because p(an English text) = 0 under your model. Your model doesn't allow the letter "x", for instance.

(We say that -log(0) = infinity because infinity is the limit of -log(x) as x -> 0.)

e. infinity.

2. False.
But almost true! It's true of cross-entropy, not perplexity.

crossentropy/word = 6 * crossentropy/letter, so
perplexity/word = (2^(crossentropy/word))
= 2^(6*(crossentropy/letter))
= (2^(crossentropy/letter))^6
= (perplexity/letter)^6

For example, if you are 10-ways perplexed about what each letter is, then you will be a million-ways perplexed (10^6) about what each word is.

But wait, I wrote above:
crossentropy/word = 6*(crossentropy/letter).
Why is that true?

Well, suppose there are N words in the corpus.
Then there must be 6N letters (so that each word is 6 letters on average).
So total cross-entropy of the corpus can be expressed in 2 different ways:
(crossentropy/word) * N words = (crossentropy/letter) * 6N letters
Dividing both sides by N gives
(crossentropy/word) = 6 * (crossentropy/letter)?

More intuitively, this is for the same reason that when you take a bus trip,
cost/week = 7 * cost/day

If you travel by bus for 100 weeks and it costs you $840 total, then the cost was $8.40/week or $1.20/day. (Maybe some days were cheaper and some days were more expensive; this is just the AVERAGE cost.)

In the same way, if the overall log-probability of your test corpus is 2^-840, giving a cost of 840 bits to encode the whole corpus, and the corpus consisted of 100 words = 600 letters, then the cross-entropy was 8.4 bits/word or 1.4 bits/letter.

Think of the log-probability as the cost of driving through the corpus using your model (your bus). You have to pay to be surprised.

Some portions of the journey cost more bits than others, because they are longer, or more surprising in context (driving the bus uphill requires more gas!).

But to get the average cost per word or per letter, you look at the total cost and divide by the total number of words or letters. This is the cross-entropy of the model.

3. (a)

	HHHH	HHHH	THHH	HTHH	THHT	HTHT
unsmoothed	1	1	3/4	3/4	1/2	1/2
add-1 smoothed	5/6	5/6	2/3	2/3	1/2	1/2

(b) 3/4 (just average the first line above)
2/3 (just average the second line above)

(c) These techniques are called "estimators" because they produce estimates of p(H).

Less bias: unsmoothed.

The unsmoothed estimator is completely unbiased. That means that ON AVERAGE, it will get the right answer.

By "on average," I don't mean the average of the 6 outcomes (alternate universes) listed in part (a), but a weighted average over ALL outcomes in proportion to their probability.

That it is unbiased is easy to see. For example, suppose the true probability p(H) = 0.7. Then in the AVERAGE universe, 4 flips will give 4*0.7 = 2.8 heads, leading to an estimate p(H) = 2.8/4 = 0.7.

(To be more precise, there are 2^4 = 16 possible outcomes for 4 flips: for example, p(HTHH) = 0.7*0.3*0.7*0.7. Each outcome gives a different unsmoothed estimate, like 1 or 3/4 or 1/2 in the table above. If you average over those estimates in proportion to their probability, you get
1*p(HHHH) + (3/4)*p(HTHH) + ... + 0*p(TTTT) (16 terms)
which after some rearrangement works out to 0.7 as claimed.)

Formally, the bias is (mean estimated value - true value). This is 0 for the unsmoothed technique.

By contrast, the smoothed estimator will be biased toward giving answers that are closer to 1/2 than the truth, because adding 1 to the H and T counts always makes the estimate closer to 1/2 than what was observed: (5/6)*p(HHHH) + (2/3)*p(HTHH) + ... + (1/6)*p(TTTT) (16 terms) This works out to 0.6333, so the bias is (0.6333-0.7) = -0.0667.

Since the smoothed estimator's bias is farther from 0, we say that it is more biased than the unsmoothed estimator.

Less variance: smoothed.

You will note that in the 6 columns above (alternate universes), the smoothed estimates are closer to one another than the unsmoothed estimates are. So our estimate varies less from universe to universe.

(Formally, the variance is the average over all outcomes (universes) of (estimated value - mean estimated value)^2. If p(H)=0.7, it works out to 0.02778 for the unsmoothed estimator but only 0.02333 for the smoothed estimator.)

Which AVERAGE is closer to the truth? That's the same as asking which estimator has bias closer to zero. It's the unsmoothed estimator, whose bias is actually 0.

But who cares that the unsmoothed estimate is right on AVERAGE? The SINGLE estimate in YOUR universe could be wildly wrong, due to variance among universes.

If you're bear hunting, you don't want to shoot far left half the time and far right the other half. That may hit the bear on average, but it never actually hits the bear! It's not like the negative errors cancel out the positive errors: they're both bad. So you need a better way to measure the "loss" of your approach. Loss is any measure of how bad the error is; it should always be >= 0. The expected loss is a useful measure of how bad your estimator is.

That is why someone might want to use the smoothed estimator. The estimates that it produces might have lower loss on average.

Specifically, add-1 smoothing will tend to reduce loss if the TRUE p(H) is close to 0.5, because it pushes the estimates closer to 0.5. (Add-10 smoothing does so even more strongly.) This is particularly true with a small number of flips, when the unsmoothed loss may be high and smoothing strongly affects it. If the TRUE p(H) is far from 0.5, however, add-1 smoothing could increase the loss.

How do we define loss? SQUARED ERROR is a common way to measure the loss of an estimate, because it decomposes into bias^2 + variance. The average loss is then called "mean squared error" (MSE).

Squared error may not be the loss that you care about. In NLP, we often want the surprisal of test events to be low. That suggests instead using LOG-LOSS -- defined as the average surprisal of a test event.

For example, in the first two columns, you get unsmoothed estimates p(H)=1 and p(T)=0, which would put you in a world of hurt when you eventually see a T (which you will, unless p(H)=1). As a result, your average loss over all universes will also be infinity. Seems like a good reason to switch to a smoothed estimate!

(d) Each of the H flips costs -log(0.8) bits under your model, and each of the T flips costs -log(0.2) bits.
All logs are base 2 as usual.

So the total cost per flip is
  (-4*log(0.8) - 2*log(0.2))/6
or equivalently
  -(2/3)*log(0.8) - (1/3)*log(0.2)

Note: If you want to get really fancy, you can notice that
    log(0.8) = log(0.2*4) = log(0.2)+log(4) = log(0.2)+2
so the answer becomes
  - log(0.2) - 4/3
or equivalently

log(5) - 4/3
where log(5) is a little more than log(4)=2; so the answer is about 1 bit.
(With a calculator, the exact number turns out to be about 0.99 bit.)

(e) The problem is that 6 flips is not really enough to see the true behavior of the coin. The easiest solution is to flip (say) 1000 times instead of 6 times. A 1000-flip sequence will give a lower-variance estimate of cross-entropy, because one 1000-flip sequence of this coin is much like another.

(Since the coin has p(H)=3/4, very close to 3/4 of the 1000 flips will be heads. By contrast, we got a 6-flip sequence HHHHTT in which only 2/3 of the flips were heads, and this is far from unusual.)

Remark: The question asked for an easy way to improve the estimate. What would a hard way be? Well, you could take a mass specrometer and get a 3-D scan of the coin, then put it through your physics simulator and find out how often the simulated coin comes up heads. Again you'd want to flip the simulated coin many times, but maybe that's cheaper or faster than flipping the real coin. Or you could drop the random sampling altogether, and try to do a lot of integrals to analyze what the physics simulator would do with this coin. Note that the imperfectness of the scanning and the physics model will translate into bias. (Also note that taking those integrals will be hard, and some methods for doing it numerically actually still involve random sampling!)

4. The correct answers are a, c, d.

(a) should look familiar from an exercise on homework 2.
It can be obtained by backing off from this use of the chain rule:

```
p(w_0,w_1,w_2,w_3,w_4,w_5,w_6)
 = p(w_6)
   * p(w_5 | w_6)
   * p(w_4 | w_5, w_6)
   * p(w_3 | w_4, w_5, w_6)
   * p(w_2 | w_3, w_4, w_5, w_6)
   * p(w_1 | w_2, w_3, w_4, w_5, w_6)
   * p(w_0 | w_1, w_2, w_3, w_4, w_5, w_6)
```

(c) can be obtained by backing off from this use of the chain rule:

```
p(w_0,w_1,w_2,w_3,w_4,w_5,w_6)
 = p(w_0)
   * p(w_1 | w_0)
   * p(w_2 | w_0, w_1)
   * p(w_3 | w_0, w_1, w_2)
   * p(w_4 | w_0, w_1, w_2, w_3)
   * p(w_5 | w_0, w_1, w_2, w_3, w_4)
   * p(w_6 | w_0, w_1, w_2, w_3, w_4, w_5)
```

The same is true of (d). If you're not convinced, notice that to back off, we always generalize the condition to the right of the bar. Let's do this formally. Let W_0, W_1, W_2, ... be random variables denoting the words, and U_i be a random variable denoting the number of uncommon words in (W_0,W_1,...W_i). The notation

```
    p(w_3 | w_0, w_1, w_2)
```

is really just shorthand for

```
    p(W_3 = w_3 | W_0 = w_0, W_1 = w_1, W_2 = w_2)
  = p(W_3 = song | W_0 = BOS, W_1 = Sing, W_2 = a)
  = p(W_3 = song | W_0 = BOS, W_1 = Sing, W_2 = a, U_2 = 1)
```

We can back off from this to
  = p(W_3 = song | U_2 = 1)
as claimed.

However, (b) is NOT valid. The problem is that, for example, W_2 is generated dependent on W_3 and also vice-versa. This is circular.

In any valid use of the chain rule, you introduce variables one at a time, in some fixed order, with each variable's value being conditioned on only the values already chosen for the previous variables -- or, with backoff, on some subset of those values. But (b) cannot have been obtained this way, because of the circularity noted above. (W_2 must have been introduced after W_3, and also vice-versa, which is impossible.)

5a. Answer 1: Because we have no training data for Synglish.

(Although it is possible to create such training data. Just simulate the plagiarism process by sampling sentences from C and randomly replacing their words with synonyms. This will give you a sample of Synglish, at least if our model of the plagiarism process is correct. Now you can indeed train a log-linear model to distinguish between English and Synglish.)

Answer 2: Even if we have training data, it is not clear what features to use. Whether the log-linear model performs well depends on whether it has adequate features. The method in the rest of the problem doesn't require any feature design -- it simply draws the appropriate conclusions from the language models.

5b. Bayes' Theorem says that the two probabilities in the odds ratio are respectively proportional to p(Synglish)*p(W | Synglish) and p(English)*p(W | English). Therefore, their ratio is

$$\frac{p(\text{Synglish})\ p(W \mid \text{Synglish})}{p(\text{English})\ p(W \mid \text{English})}$$

Note that p(Synglish) is our prior probability that a student decides to plagiarize. p(W | Synglish) is given by a model of how text is generated in this case. The problem says that we have such a model, and it is called p_Synglish. So the final answer is

$$\frac{p(\text{Synglish})\ p\_\text{Synglish}(W)}{p(\text{English})\ p\_\text{English}(W)}$$

Remark: If you are feeling insecure about the odds ratio above, here is a more explicit use of Bayes' Theorem to get it:

$$p(\text{Synglish} \mid W) = p(\text{Synglish}) * p(W \mid \text{Synglish}) / p(W)$$
$$p(\text{English} \mid W) = p(\text{English}) * p(W \mid \text{English}) / p(W)$$

When we take the odds ratio of these two, the identical denominators cancel out. This is very convenient because the denominator has the slightly complicated form

$$p(W) = p(\text{Synglish}) * p(W \mid \text{Synglish}) + p(\text{English}) * p(W \mid \text{English})$$

In general, when using Bayes' Theorem, it is sometimes easier to think about RELATIVE probabilities of different outcomes (odds ratios) without bothering to convert them to absolute probabilities.

5c. Here is one reasonable answer:

p_Synglish(W) = sum_V p(V) * p(w_1 | v_1) * ... * p(w_n | v_n)

This assumes that for each word, the student makes an INDEPENDENT decision about whether and how to replace it.

Notice that this formula sums over the many ways of getting
   W = In the senior sunshines, this was easy.
We don't know whether V was
   V = In the old days, this was easy.
   V = Inside the old days, this was simple.
   V = In the super sunshines, this was simple.
   ...

So we have to sum over all the possibilities. What we have really done is to build a joint model of p(V,W), and then we obtain p(W) = sum_V p(V,W).

Now, we have to fill in the details. First of all, what is p(V)? This is the English sentence that the cheating student started with. There are at least three reasonable models that we could use:

(A) p(V) = 1/|C| if V is in C
     p(V) = 0   otherwise

(B) p(V) = p_English(V)

(C) p(V) = p_trigram(V)
            where the trigram probabilities are estimated from English

Second, what is p(w_i | v_i)? Well, the student doesn't replace all the words. A simple model is that the student has a 90% chance of leaving v_i alone (so w_i = v_i), and a 10% chance of replacing it with an element of syn(v_i), where all elements including v_i itself are equally likely to be chosen. Under this model,

p(w_i | v_i) = 0.9 + 0.1 / syn(v_i) if w_i = v_i
p(w_i | v_i) = 0.1 / syn(v_i)   if w_i is in syn(v_i) but w_i != v_i
p(w_i | v_i) = 0   otherwise

(Remark: A more complex model might say that if v_i is a function word, then the student has a 98% chance of leaving it alone, but if v_i is a content word, then the student has only an 80% chance of leaving it alone.)

(These numbers such as 0.9, 0.98, 0.8 don't have to be hard-coded. They could be learned parameters of the model. But we would need Synglish data to learn them from.)

Note that are other possible models. For example, another model is that the student first chooses k words from the n word sentence to replace, and then replaces those. This means that replacing one word makes the student less likely to replace other words. That is different from making an independent decision for each word.

5d. Model (A) is like a 100-gram model that is trained on C, and just memorizes C.

Model (B) is a 3-gram model that is trained on C, so it may generalize to sentences outside of C.

In general, reducing the value of n in an n-gram model will increase the bias (bad) but reduce the variance (good).

i. Model (B) suffers from greater bias because it pretends that English is a trigram model. Model (B) cannot ever learn properties of English like "a sentence almost always has exactly one main verb."

Model (A) is not biased at all, because ON AVERAGE it will give you the correct distribution of English. (If you average over many random training sets C.)

ii. Model (A) suffers from greater variance because the estimates are highly sensitive to the particular training set C. In particular, model (A) estimates sentences that are in C to have probability $1/|C|$, but estimates sentences that are not in C to have probabability 0, which is worse by a factor of infinity!

Model (B) smooths the training data somewhat simply by using a trigram model, so that it can assign positive probability to sentences outside C. But model (A) doesn't smooth the training data at all.

iii. There are a lot of options between a 3-gram model and a 100-gram model. How about a 4-gram model? Better yet, how about a 100-gram model estimated with backoff smoothing?

Another possibility is to take a linear interpolation of the two models:

$$p(V) = \lambda * p_A(V) + (1-\lambda) * p_B(V)$$

But this is probably not as clever as using backoff smoothing, which factors p(V) into a product of contextual probabilities, each of which can be interpolated to a *different* degree depending on how well its context was observed in C.

5e. i. We want V that maximizes $p(V) * p(W \mid V)$.
(Note that in question 4c, we summed over the possible source sentences V, but here we are looking for the single most likely one.)

ii. Decoding.

(We have a noisy channel model. V is the source sentence, and synonym substitution is the noisy channel that messes up V into W. We observe W and want to "decode" this message to recover V.)