HUGH HAN, PEIYI ZHENG

600.465 Natural Language Processing **Fall 2016**
Homework #3 **Due:** 27 September 2016, 5:00pm

1. The log probability for each sample corpus, training on the `switchboard-small` corpus, is calculated as follows.

```
  -12111.3   speech/sample1
  -7388.84   speech/sample2
  -7468.29   speech/sample3
```

To calculate the perplexity of each corpus, we can simply raise 2 to the power of each *negated* log probability. That is, we can simply do the following.

$$\begin{array}{lll} 2^{12111.3} & \text{for} & \texttt{speech/sample1} \\ 2^{7388.84} & \text{for} & \texttt{speech/sample2} \\ 2^{7468.29} & \text{for} & \texttt{speech/sample3} \end{array}$$

To calculate the perplexity per word, we first need to calculate the cross-entropy by dividing each negated log proability by the number of words in its respective corpus.

The number of words in each corpus is calculated as follows.

```
  1686   speech/sample1
   978   speech/sample2
   985   speech/sample3
```

Then we see that the perplexity per word of each file is the following.

$$\begin{array}{lll} 2^{\frac{12111.3}{1686}} & \text{for} & \texttt{speech/sample1} \\ 2^{\frac{7388.84}{978}} & \text{for} & \texttt{speech/sample2} \\ 2^{\frac{7468.29}{985}} & \text{for} & \texttt{speech/sample3} \end{array}$$

We obtain the following approximated perplexity-per-word values.

$$\begin{array}{lll} 145.36 & \text{for} & \texttt{speech/sample1} \\ 188.06 & \text{for} & \texttt{speech/sample2} \\ 191.61 & \text{for} & \texttt{speech/sample3} \end{array}$$

Training on the larger `switchboard` corpus, we get the following log probabilities for the following sample corpuses, respectively:

```
  -12561.5   speech/sample1
  -7538.27   speech/sample2
  -7938.95   speech/sample3
```

Note that each of the log-probabilities grew more negative for its respective sample corpus. Thus, the each perplexity must also be smaller for its respective sample corpus. The reason for this is because if we increase the size of our training data, then we introduce a greater number of possible sentences. That is, the probability that the sentences in our specific test files appear would be much smaller, in that there is a greater variety of possibilities to choose from.

2. IMPLEMENTATION PROBLEM

3. We chose to do spam detection.

   When classifying the sample files containing genuine messages, we received the following output:

```
178 looked more like gen_spam/train/gen (98.89%)
2 looked more like gen_spam/train/spam (1.11%)
```

   When classifying the sample files containing spam messages, we received the following output:

```
66 looked more like gen_spam/train/gen (73.33%)
24 looked more like gen_spam/train/spam (26.67%)
```

   That is, we had the following error rates for the different data sets.

$$\begin{aligned} \texttt{gen} &: \quad \text{error-rate} = 1.11\% \\ \texttt{spam} &: \quad \text{error-rate} = 73.33\% \end{aligned}$$

   (a) Each classification decision is based on probability. That is, whichever training corpus yields a higher cross-entropy with each test file is the training corpus to which that test file will be classified.

   However, we cannot necessarily deduce that the lowest cross-entropy occurs with the lowest error rate. With a low error rate, it could just be that the opposite incorrect corpus has a higher cross-entropy than the correct corpus, instead of the correct corpus having the lowest possible cross-entropy.

   Thus, we are forced to do some type of guess-and-check to find the value of the lowest cross-entropy. We proceeded by doing a human version of binary search.

   Using our binary search, we find the following two values of the approximate smallest log-probabilities.

$$\begin{aligned} \texttt{gen} &: \quad \text{log-probability} \approx -423681 \\ \texttt{spam} &: \quad \text{log-probability} \approx -280362 \end{aligned}$$

   Next, we can count the number of words in all of the gen dev files and the number of words in all of the spam dev files using the following two commands, respectively.

```
$ find gen_spam/dev/gen/ -name '*.txt' | xargs wc -w
$ find gen_spam/dev/spam/ -name '*.txt' | xargs wc -w
```

We went up getting the values of 48198 and 39284 for the `gen` and `spam` dev files, respectively.

Using these values, we can then obtain the approximate smallest cross-entropies.

$$\begin{aligned} \texttt{gen} &: \text{cross-entropy} \approx 8.79 \\ \texttt{spam} &: \text{cross-entropy} \approx 7.14 \end{aligned}$$

The λ values that were used in finding these minimum cross-entropies were as follows.

$$\begin{aligned} \lambda_{\texttt{gen}} &= 0.01357 \\ \lambda_{\texttt{spam}} &= 0.0083 \end{aligned}$$

(b) We can do binary search again, but let's think of how we can be a little smarter this time. We already have

$$\begin{aligned} \lambda_{\texttt{gen}} &= 0.01357 \\ \lambda_{\texttt{spam}} &= 0.0083 \end{aligned}$$

so we can use them as critical points. We see that:

(i) all $\lambda < \lambda_{\texttt{spam}}$ cause increasing log-probabilities

(ii) all $\lambda > \lambda_{\texttt{gen}}$ cause increasing log-probabilities

Then the following must be true.

$$\lambda_{\texttt{spam}} \leq \lambda^* \leq \lambda_{\texttt{gen}}$$

So now we can do a smarter guess-and-check binary search, and we find the following value of λ^*.

$$\lambda^* \approx 0.0104$$

(c) First, let's try to classify the test files containing genuine messages.

```
     347 looked more like gen_spam/train/gen (96.39%)
     13 looked more like gen_spam/train/spam (3.61%)
```

Next, let's try to classify the test files containing spam messages.

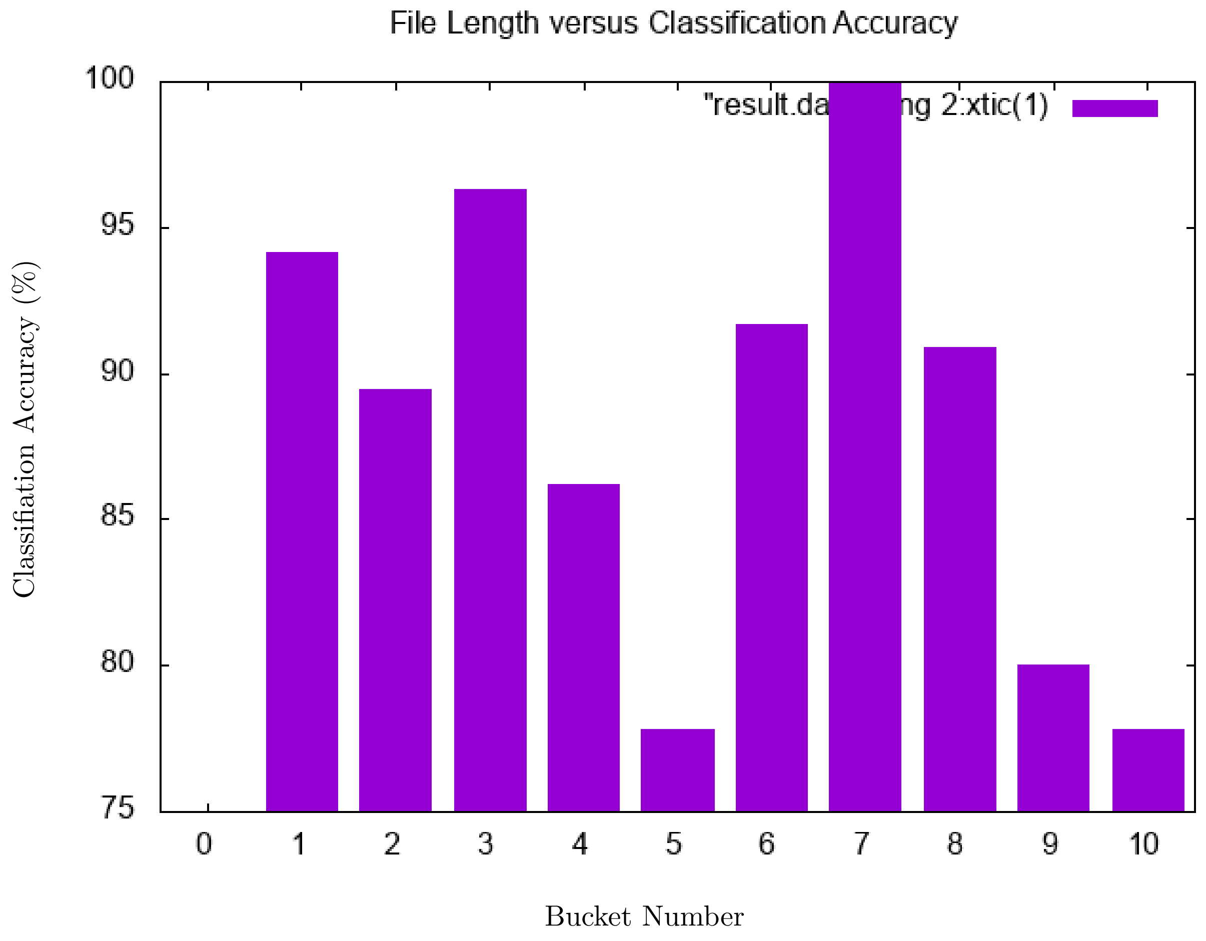
```
     50 looked more like gen_spam/train/gen (27.78%)
     130 looked more like gen_spam/train/spam (72.22%)
```

So then we get the following error rates.

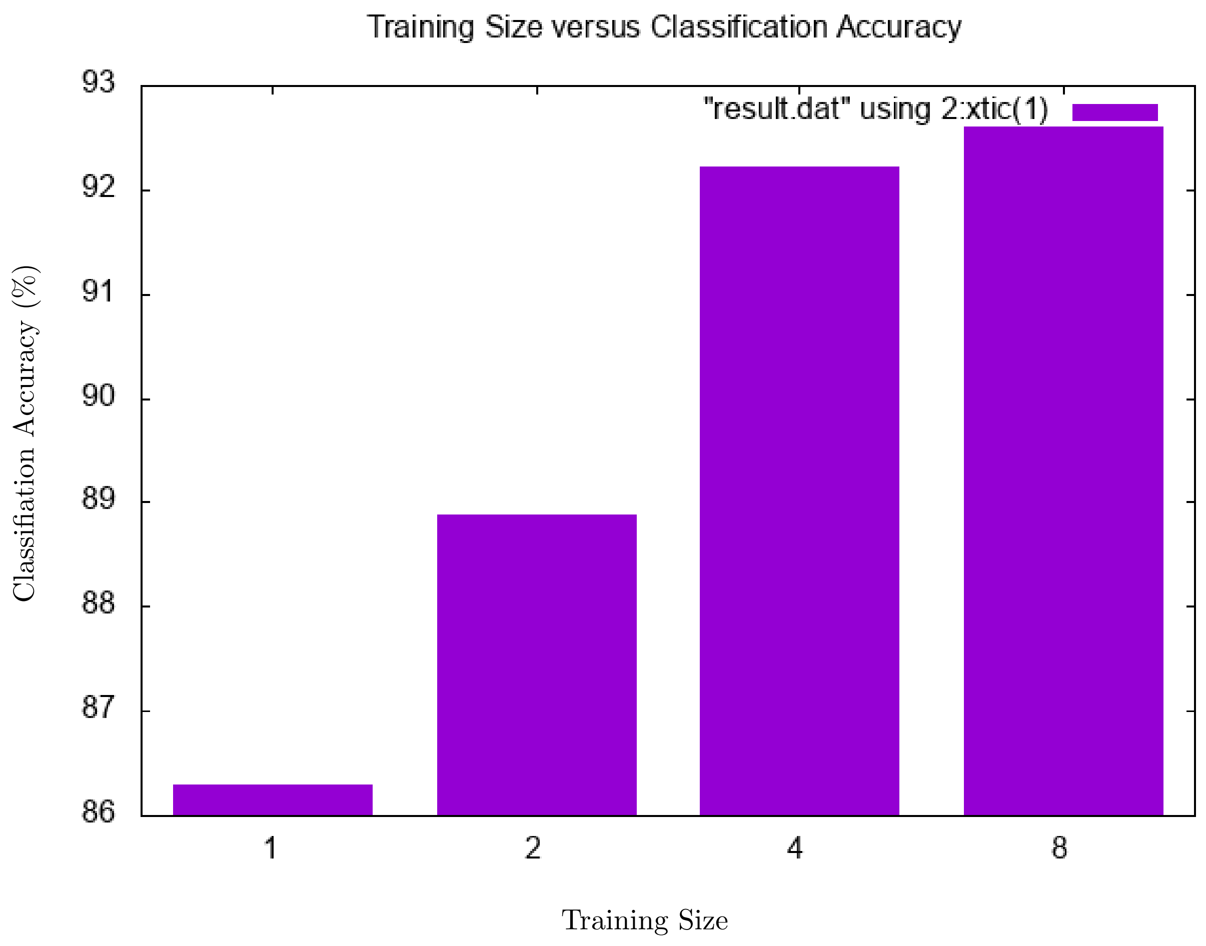
$$\begin{aligned} \texttt{gen} &: \text{error-rate} = 3.61\% \\ \texttt{spam} &: \text{error-rate} = 27.78\% \end{aligned}$$

(d) For this problem, we can all of the possible file lengths into 10 buckets of file length intervals. For instance, the first bucket would contain the classification results from the files with lengths in the range [0, 40), the second bucket would contain those in the range [40, 80), and so on. We can reserve our last bucket as a “special” bucket. That is, this special bucket will contain the classificatioon results of all of the files with lengths greater than 400. Now note that the first bucket can serve as a baseline, which represents the lowest accuracy of 75%. The reason for this is that small files containing less information, so they could be more difficult to classify.

The plot is shown below.



(e) It is easy to see that as the training data size increases, the classification accuracy also increases. But increasing the size of the training set does not necessarily solve all classification accuracy issues. At some point, it will level off. For example, it was noted that when we switched from `gen-times4` to `gen-times8`, the classification accuracy was improved only slightly.



4. (a) Let's say that we take $V = 19{,}999$.

   i. Using the UNIFORM estimate, if we see a word that is out of the vocabulary, we would have no choice but to assign that OOV word a probability of 0, due to it being the 20,000th word of a vocabulary of size 19,999.
   The reason for this is that the probabilities of all other words must sum to 1, and we cannot have a probability greater than 1.

$$\sum_{i=1}^{V} \frac{1}{V} = 1, \quad V = 19{,}999$$

   From the above equation, it is clear that we cannot assign OOV a probability of any value other than 0, if we are not to exceed a total probability of 1.
   However, assigning OOV a probability of 0 is a bad estimation, because no word should have a probability of 0, regardless of whether that word was observed in the training data.

   ii. Using the ADDL estimate, a somewhat similar situation occurs. The ADDL probability is defined as follows.

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

   Now if we were to sum the probabilities $\hat{p}(z \mid xy)$ for all $z \in$ vocabulary, we would take a summation over 20,000 elements. However, if $V = 19{,}999$, we would end up getting a sum that is greater than 1, which canot be true due to the laws of probability. Hence, taking $V = 19{,}999$ is problematic.

(b) If we let $\lambda = 0$, then we are not modifying our probability at all. That is, we get the following from our probabiliy estimate.

$$\begin{aligned}\hat{p}(z \mid xy) &= \frac{c(xyz) + \lambda}{c(xy) + \lambda V} \\ &= \frac{c(xyz) + 0}{c(xy) + 0} \\ &= \frac{c(xyz)}{c(xy)}\end{aligned}$$

This ends up being the naive estimate of $\hat{p}(z \mid xy)$. If no smoothing occurs, then the probabilitiy given to any trigram that did not appear in the training corpus is 0, which is not well represntative of natural language.

(c) i. First, let's think about the case when $c(xyz) = c(xyz') = 0$.

We see that

$$\hat{p}(z \mid xy) \quad = \quad \frac{0 + \lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

Then the difference between $\hat{p}(z \mid xy)$ and $\hat{p}(z' \mid xy)$ is the possible differing values of $\hat{p}(z \mid y)$ and $\hat{p}(z' \mid y)$. And we cannot guarantee that $\hat{p}(z \mid y) = \hat{p}(z' \mid y)$.

For example, we might see neither of the phrases "kiss the tree" nor "kiss the Jupiter". However, when we backoff, it is possible that we would have seen both of the phrases "the tree" and "the Jupiter" (which are both much more common).

And in English, it is much more likely that the phrase "the tree" is more common than "the Jupiter", which demonstrates a case in which $\hat{p}(z \mid y) \neq \hat{p}(z' \mid y)$.

$\implies \hat{p}(z \mid xy) \neq \hat{p}(z' \mid xy)$.

ii. Now let's think about the case when $c(xyz) = c(xyz') = 1$.

We see that

$$\hat{p}(z \mid xy) \quad = \quad \frac{1 + \lambda V \cdot \hat{p}(z \mid y)}{c(xy) + \lambda V}$$

Again, the difference between $\hat{p}(z \mid xy)$ and $\hat{p}(z' \mid xy)$ is the possible differing values of $\hat{p}(z \mid y)$ and $\hat{p}(z' \mid y)$. And again, we cannot guarantee that $\hat{p}(z \mid y) = \hat{p}(z' \mid y)$.

For example, consider the phrases "kick the Spongebob" and "kick the man", both of which have been counted exaclty once in the training corpora. But when we backoff, it is possible that we would have seen the "the man" much more frequently than "the Spongebob". Again, this is another case in which $\hat{p}(z \mid y) \neq \hat{p}(z' \mid y)$.

$\implies \hat{p}(z \mid xy) \neq \hat{p}(z' \mid xy)$.

(d) Again, we can write out the formula for the probability of a particular trigram using BACKOFF_ADDL smoothing.

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

If we were to increase the value of λ, we would be smoothing at a greater rate. That is, we would be more heavily weighting the value of $\hat{p}(z \mid y)$ into our calculation of $\hat{p}(z \mid xy)$. So increasing the value of λ would push our probabilities $\hat{p}(z \mid xy)$ closer to the probabilities $\hat{p}(z \mid y)$.

Now note that $\hat{p}(z \mid y)$ backs off onto $\hat{p}(z)$, which is calculated using the naive estimate. Also ote that the calculation of $\hat{p}(z \mid y)$ is *also* influenced by λ (i.e. larger values of λ push it towards $\hat{p}(z)$). So if we increase λ to be large enough, we would eventually push the probabilities $\hat{p}(z \mid xy)$ towards $1/V$, or the uniform distribution.

5. (a) `IMPLEMENTATION PROBLEM`

   (b) Using $\lambda^* = 0.0104$, as calculated in problem **3**(b), we can calculate the cross-entropies for the switchboard corpora and analyze the text categorization error rates for `gen/spam`. The following generalizations are made with the assumption that $\lambda^* = 0.0104$.

cross-entropies for the switchboard corpora

In problem **1**, we calculated the log-probabilities using ADDL using a λ value of 0.01. Because $\lambda^* = 0.0104$, we need to recalculate the log-probabilities.

Below contains log-probabilities using ADDL for switchboard corpora.

```
-12572.8   speech/sample1
-7544.45   speech/sample2
-7944.22   speech/sample3
```

Below contains log-probabilities using BACKOFF_ADDL for switchboard corpora.

```
-10040.1   speech/sample1
-6039.31   speech/sample2
-6413.95   speech/sample3
```

Now to calculate the cross-entropties of the sample files, we must count the number of words in each sample file. We can do so using the following command.

```
$ find speech -name 'sample*' | xargs wc -w
```

```
1686   speech/sample1
978    speech/sample2
985    speech/sample3
```

Then we can find the cross-entropies of each sample file by dividing the negative log-probability of each sample file by its corresponding word count.

Below contains the cross-entropies of the switchboard corpora for each of the three samples, using ADDL and BACKOFF_ADDL smoothing.

	ADDL	BACKOFF_ADDL
`speech/sample1`	7.457	5.955
`speech/sample2`	7.714	6.175
`speech/sample3`	8.065	6.512

For each of the three speech samples, we see that switching from ADDL smoothing to BACKOFF_ADDL smoothing decreases the cross-entropy greatly, against the training switchboard corpus.

text categorization error rates for gen/spam

Below contains output using BACKOFF_ADDL for `gen` test files.

```
337 looked more like gen_spam/train/gen (93.61%)
23 looked more like gen_spam/train/spam (6.39%)
```

Below contains output using BACKOFF_ADDL for `spam` test files.

```
17 looked more like gen_spam/train/gen (9.44%)
163 looked more like gen_spam/train/spam (90.56%)
```

From these outputs, we can see that the error rates using BACKOFF_ADDL were *significantly* lower than the error rates using ADDL as in **3**(c) for `spam` test files.

At the same time, the error rates using BACKOFF_ADDL were *slightly* higher than the error rates using ADDL as in **3**(c) for `gen` test files.

We can conclude that switching from ADDL to BACKOFF_ADDL causes the error rates for classifying spam files to decrease extraordinarily, but at the cost of slightly increasing error rates for classifying genuine files. It seems to balance out the classification correctness rates a quite a bit.

(c) *Extra credit:* We were able to find $\lambda_1 = 0.1$, which works a bit better than λ^* in a sense. By using λ_1, our classification results on test data are exactly the same, but the cross-entropies are much smaller.

6. (a) `IMPLEMENTATION PROBLEM`

(b) `IMPLEMENTATION PROBLEM`

(c)

```
epoch 1:  F=-3.274666
epoch 2:  F=-3.160848
epoch 3:  F=-3.114404
epoch 4:  F=-3.087500
epoch 5:  F=-3.070043
epoch 6:  F=-3.058030
epoch 7:  F=-3.049404
epoch 8:  F=-3.042993
epoch 9:  F=-3.038087
epoch 10: F=-3.034236
```

(d) First, we can run some experiments on the development files. Because we will need the word counts of the development files, we can get those easily using the following two commands.

```
wc -w english_spanish/dev/english/*/*
```

```
wc -w english_spanish/dev/spanish/*/*
```

For `english`, we have total word count of 16820.
For `spanish`, we have total word count of 17069.

After getting the word counts of the files, we can then calculate the negative log probabilities across each particular group of files (either `english` or `spanish`).
To calculate the cross-entropy, we simply divide the negative log probabilities by their corresponding word counts.
The data below contain results using development files.

ADDL

C = 1

	log-probability	cross-entropy
`english`	−72686.8	4.321
`spanish`	−67682.5	3.965

C = 0.05

	log-probability	cross-entropy
`english`	−70801.5	4.209
`spanish`	−67773	3.971

C = 3

	log-probability	cross-entropy
`english`	−77439.3	4.604
`spanish`	−72267.5	4.234

Generally speaking, it would seem that increasing the value of C increases the cross-entropy when using the ADDL smoothing function.

BACKOFF_ADDL

C = 1

	log-probability	cross-entropy
`english`	−62616.4	3.723
`spanish`	−61372.4	3.596

C = 0.05

	log-probability	cross-entropy
`english`	−71948	4.278
`spanish`	−72578.1	4.252

C = 3

	log-probability	cross-entropy
`english`	−65199.2	3.876
`spanish`	−63375	3.713

Generally speaking, it would seem that there is an optimal value of C that minimizes the cross-entropy when using the BACKOFF_ADDL. This optimal value most likely occurs somewhere in the range $(0.05, 3)$.

LOGLINEAR

C = 1

	log-probability	cross-entropy
`english`	−75830.3	4.443
`spanish`	−74669.3	4.375

C = 0.05

	log-probability	cross-entropy
`english`	−75876	4.445
`spanish`	−74640.2	4.373

C = 3

	log-probability	cross-entropy
`english`	−75872	4.511
`spanish`	−74811.4	4.383

Generally speaking, there does not seem to be a significant change in the cross-entropy for any C in the range $[0.05, 1]$ when using the LOGLINEAR smoothing function. However, increaseing the value of C past 1 seems to increase the cross-entropy, but only slightly.

After analyzing the results of changing C around in relation to the effect on cross-entropy for the ADDL, BACKOFF_ADDL, and LOGLINEAR functions, it would probably make sense to take a look at how changing C around affects classification error rates of each model.

ADDL

	$C = 0.0104$	$C = 0.0500$	$C = 1.0000$	$C = 3.0000$
`english`	12.50%	10.83%	8.33%	6.67%
`spanish`	5.04%	6.72%	6.72%	6.72%

Generally speaking, it would seem that increasing the value of C decreases the classification error rate when using the ADDL smoothing function for our interval range $C \in [0.0104, 3]$.

BACKOFF_ADDL

	$C = 0.0104$	$C = 0.0500$	$C = 1.0000$	$C = 3.0000$
`english`	5.00%	5.00%	5.00%	7.50%
`spanish`	26.89%	28.57%	26.05%	23.53%

Generally speaking, it would seem that there is an optimal value of C that minimizes the classification error rate when using the BACKOFF_ADDL. This optimal value seems to be different for `english` and `spanish`.

LOGLINEAR

	$C = 0.0104$	$C = 0.0500$	$C = 1.0000$	$C = 3.0000$
`english`	21.67%	21.67%	20.83 %	20.00%
`spanish`	41.18%	41.18%	41.18 %	42.02%

Generally speaking, all values of C using the LOGLINEAR smoothing function caused high error rates for both `english` and `spanish` development files when compared with BACKOFF_ADDL smoothing. It seems that $C = 0.0104$, on average, minimized the error rates. For the sake of time, we decided to take $C^* = 0.0104$. Note that these values were calculated using $\gamma_0 = 0.01$.

Compared to BACKOFF_ADDL smoothing, LOGLINEAR performed much worse. For all values of C that were tested, we saw that LOGLINEAR smoothing causes higher classification error rates in both the `english` and `spanish` development files.

After confirming this, we performed some tests with BACKOFF_ADDL and LOGLINEAR using the `english` and `spanish` test files.

The results are recorded below.

BACKOFF_ADDL

	$C = 0.0104$	$C = 0.0500$	$C = 1.0000$	$C = 3.0000$
`english`	5.95%	6.22%	6.76%	8.11%
`spanish`	23.58%	24.66%	24.93%	23.85%

Generally speaking, it would seem that there is an optimal value of C that minimizes the classification error rate when using the BACKOFF_ADDL. This optimal value seems to be different for `english` and `spanish`.

LOGLINEAR

	$C = 0.0104$	$C = 0.0500$	$C = 1.0000$	$C = 3.0000$
`english`	19.73%	20.00%	19.73 %	19.46%
`spanish`	36.04%	36.04%	35.77 %	36.86%

Generally speaking, all values of C using the LOGLINEAR smoothing function caused high error rates for both `english` and `spanish` development files. It seems that $C = 0.0104$, on average, minimized the error rates. For the sake of time, we decided to take $C^* = 0.0104$.
Note that these values were calculated using $\gamma_0 = 0.01$.

(e) *Extra credit:* no answer.

(f) After implementing the new feature, we played around with the values of α for a bit and concluded that $\alpha = 0.25$ generally yielded better results for classification error rates.

We compared the results with our old LOGLINEAR model using $C = 0.0104$.

After running the new model, we found that the `english` test files produced an error rate of 12.70%, and the `spanish` test files produced an error rate of 18.73%. Both of these error rates are significantly lower than before the new feature was introduced.

For `en.1K`, our program learned $\beta = 3.289$, and found a log-probability of -152135 for the `english` test files. With a word count of 35240, we can compute the total cross-entropy to be approximately 4.317. This is much smaller than the computed 5.197 cross-entropy of the `english` test files using BACKOFF_ADDL.

For `sp.1K`, our program learned $\beta = 3.522$, and found a log-probability of -148822 for the `spanish` test files. With a word count of 35080, we can compute the total cross-entropy to be approximately 4.242. This is much smaller than the computed 5.136 cross-entropy of the `spanish` test files using BACKOFF_ADDL.

(g) iv. After implementing the new feature and running the tests, we found that the `english` and `spanish` test files had log-probabilities of approximately -207303 and -149003, respectively. We see that their respective cross-entropies equated to approximately 5.883 and 4.248, respectively.

For the `english` test files, the cross-entropy increased tremendously while for the `spanish` test files, the cross-entropy increased only slightly from before this feature was implemented.

Below are our error-rate results for the `english` and `spanish` test files, respectively.

```
    219 looked more like en.1K (59.19%)
    151 looked more like sp.1K (40.81%)
```

```
    53 looked more like en.1K (14.36%)
    316 looked more like sp.1K (85.64%)
```

We see that our error rates are much better for `spanish`, but worse for `english`. Note that these tests were run using $\gamma_0 = 0.01$.

7. Let w denote an arbitrary word. When we want to do the classification, we may want to consider the probability $P(w \text{ is spam} \mid w)$. According to Bayes's Theorem, we can write the following equation.

$$\begin{aligned} P(w \text{ is } \texttt{spam} \mid w) &= \frac{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam})}{P(w)} \\ &= \frac{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam})}{P(w \mid w \text{ is } \texttt{spam}) \cdot P(\texttt{spam}) + P(w \mid w \text{ is } \texttt{gen}) \cdot P(\texttt{gen})} \end{aligned}$$

$P(w \text{ is } \texttt{spam} \mid w)$ and $P(w \mid w \text{ is } \texttt{gen})$ are the probabilities generated separately from the two models we train using the `spam` and `gen` corpora.

Now we know the following.

$$P(\texttt{spam}) = \frac{1}{3}$$

Then the following naturally follows.

$$P(\texttt{gen}) = 1 - P(\texttt{spam}) = \frac{2}{3}$$

We can rewrite then rewrite the following.

$$P(w \text{ is } \texttt{spam} \mid w) = \frac{1}{1 + \frac{2P(w|w \text{ is } \texttt{gen})}{P(w|w \text{ is } \texttt{spam})}}$$

By doing this, we can calculate the probability $P(w \text{ is } \texttt{spam} \mid w)$. Obviously we don't need *priori* when training the model.

In order to implement this method, we need to

i. train the `gen` model and store the probabilities for all test files,
ii. train the `spam` model and store the probabilities for all test files,
iii. iterate through all $P(w \mid w \text{ is } \texttt{gen})$ and $P(w \mid w \text{ is } \texttt{spam})$, so that we can calculate $P(w \text{ is } \texttt{spam} \mid w)$.

Extra credit: We implemented this change in `problem7.py`. We tested our implementation using an extremely small value for λ (which was `add0.00001` in our case). From this specific test, the program successfully classified approximately 33.3% of the test data to be spam. That is, the result is extremely close to *priori*.

8. (a) Using Bayes Theorem, we can express the probability $P(\vec{w} \mid U)$ in the following equation.

$$P(\vec{w} \mid U) = \frac{P(U \mid \vec{w}) * P(\vec{w})}{P(U)}$$

Now in order to maximize $P(\vec{w} \mid U)$, we can think about maximizing $log_2(P(\vec{w} \mid U))$. Using the rules of logarithms, note that the following statement is true.

$$log_2(P(\vec{w} \mid U)) = log_2(P(U \mid \vec{w})) + log_2(P(\vec{w})) - log_2(U)$$

Since $log_2(U)$ remains constant for all of the possible $\vec{w}$ being considered, we can ignore the final term on the right side of the equation when maximizing $log_2(P(\vec{w} \mid U))$.

That is, we only need to worry about maximizing the two other terms, $log_2(P(U \mid \vec{w})) + log_2(P(\vec{w}))$. Luckiily, $log_2(P(U \mid \vec{w}))$ is provided to us in the input files. Then all we need to do is calculate $log_2(P(\vec{w}))$ using our trigram model, which will measure the extent to which it looks like English.

So we iterate all 9 candidates and compute $log_2(P(U \mid \vec{w})) + log_2(P(\vec{w}))$, and can thus find the candidate with the highest probability $P(\vec{w} \mid U)$.

(b) `IMPLEMENTATION PROBLEM`

(c) Some trigrams in unrestricted data are very uncommon due to the existence of words like "uh", "um", or "huh" intertwined inside of the sentences.

Therefore in order to overcome this problem we need to utilize backing off. In pratice, we set the value of λ to be 0.01 according to our experiments on development data. We can see the overall error rates in the table below.

	test/easy	test/unrestricted
3-gram model	0.141	0.380
2-gram model	0.159	0.396
1-gram model	0.210	0.408

9. *Extra credit:* no answer.

10. *Extra credit:*

(a) We see that as $T(xy)$ approaches 1, $\hat{p}(z \mid xy)$ approaches the naive estimate. However, note that $T(xy)$ must be at least 1 for the formula not containing α to be applied.

If $T(xy) = 0$, then for $\hat{p}(z \mid xy)$ to approach the naive estimate, we need to make it so that $\alpha \approx 1$. This ensures a likelihood that $\hat{p}(z \mid y)$ will back off onto the naive estimate, which will push $\hat{p}(z \mid xy)$ closer to the naive estimate.

Thus, $\hat{p}(z \mid xy)$ approaches the naive estimate when the trigram (x, y, z) is unique (i.e. there are few other words z' that follow the bigram (x, y)), or if $\alpha \approx 1$.

(b) If all of the T values were made to be 0, it doesn't really make sense. $T(xy)$ counts the number of words that follow (x, y). But if $T(xy) = 0$ for all $(x, y) \in$ corpus, then it means that there are no trigrams in the entire corpus!

However, let's assume that we redefine T and force them to be 0 anyway. If all of the T values were made to be 0, then each discounted probability would approach the naive estimate. This answer naturally follows from the answer above.

11. *Extra credit:* no answer.