600.465 – Natural Language Processing
Assignment 3: Probability and Vector Exercises

Li-Yi Lin, Jinyi Guo

February 2016

1. Sample 1:
   log_2probability: -12111.3, word count: 1686, perplexity per word: $2^{12111.3/1686} \approx 145.37$
   Sample 2:
   log_2probability: -7388.84, word count: 978, perplexity per word: $2^{7388.84/978} \approx 188.06$
   Sample 3:
   log_2probability: -7468.29, word count: 985, perplexity per word: $2^{7468.29/985} \approx 191.61$

   When switch to the larger `switchboard` corpus the log_2probabilities go slightly lower while the perplexities go up a lot for they are calculated by taking exponential . This is because typically larger corpus have more words than smaller ones, making the probabilities of words in the sample have lower probabilities to appear.

3. (a) We chose the language ID problem. The lowest error rate we can achieve is 0.933.
   (b) The value of λ we use is 2.7.
   (c) Test result for english:
   `342 looked more like en.1K (92.43%)`
   `28 looked more like sp.1K (7.57%)`

   Test result for spanish:
   `39 looked more like en.1K (10.57%)`
   `330 looked more like sp.1K (89.43%)`
   Thus the error rate is $67/739 = 0.091$.

4. (a) For the UNIFORM estimate, the probability of each word including OOV will sum up to more than one i.e. 20000/19999 if there is an OOV in the training data, which is very likely, making it inconsistent with the rule of probability. Also the estimation for every xyz will be greater than it should be.
   For the ADDL estimate, same problems in UNIFORM estimate still exist. Since V is in the denominator of the estimate equation and everything else remains the same, the estimation for each xyz is greater than it should be and the sum of all possible $\hat{p}(z \mid xy)$ is greater than 1.
   (b) This would make the estimate very sensitive to the training data, therefore cause overfitting. For example, if $c(xy) = 1$, then if the numerator $c(xyz)$ is 1, the estimate for xyz will be 1, which is too high.
   (c) If $c(xyz) = c(xyz') = 0$, then by the definition of BACKOFF_ADDL, we have:

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

Where $c()$ is the number of tokens in the training set.
Regarding the fact that $c(z) = c(z')$ and $c(yz) = c(yz')$ are not necessarily true, we have $\hat{p}(z \mid xy) = \hat{p}(z' \mid xy)$ iff $c(z) = c(z')$ and $c(yz) = c(yz')$. Otherwise, $\hat{p}(z \mid xy) \neq \hat{p}(z' \mid xy)$.
If $c(xyz) = c(xyz') = 1$, we have:

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

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

Similarly, we have $\hat{p}(z \mid xy) = \hat{p}(z' \mid xy)$ iff $c(z) = c(z')$ and $c(yz) = c(yz')$. Otherwise, $\hat{p}(z \mid xy) \neq \hat{p}(z' \mid xy)$.

(d) By the result above, when we increase λ the denominator grows faster than numerator, making the probability estimates lower than before. Taking the innest fraction as an example:

$$\frac{c(z') + \lambda}{c() + \lambda V}$$

if we increase λ, the denominator is growing faster than numerator by a factor of V, thus the fraction converges to zero when λ is increasing. Same rule applies to the whole fraction, so the probability estimates just become lower.

5. (a) $\hat{p}(z)$ backs off to $\hat{p}()$, which equals to $1/V$. Note this can be deduced by the fact that all probabilities sum up to 1.

(b) In question 3c we have $\lambda^* = 2.7$. Here is the cross-entropies for the switchboard cropora:

```
9766.48 speech/sample1
6029.55 speech/sample2
5987.3 speech/sample3
```

For the test categorization, we have the following result:
English:

```
337 looked more like en.1K (91.08%)
33 looked more like sp.1K (8.92%)
```

Spanish:

```
83 looked more like en.1K (22.49%)
286 looked more like sp.1K (77.51%)
```

The overall error rate is 15.7%. So switching from ADDL to BACKOFF_ADDL acutally makes the performance worse, with a smaller cross-entropy.

(c) By taking a very small λ (e.g. $\lambda = 0.0001$) we can have a better performance than ADDL on classfying English files. The error rate is 5.95% while we have an error rate of 7.57% in the ADDL model. However, no matter what value for λ we take, we can't get an error rate of less than 10% for identifying spanish.

Here the λ_1 we take is much less than λ because for this particular problem and given data, we have to rely mainly on the ADDL model. This means the trigram count based on the training data works well, most trigrams on the test data also appear in the training data.

6. (c) We trained the log-linear model on lexicon chars-10.txt and training corpora en.1k and sp.1k with $\gamma_0 = 0.01$. The objective function values are shown below:
Training from corpus en.1k
epoch 1: F=-2998.765906
epoch 2: F=-2923.936118

epoch 3: F=-2884.881779
epoch 4: F=-2861.087717
epoch 5: F=-2845.191350
epoch 6: F=-2833.850381
epoch 7: F=-2825.369426
epoch 8: F=-2818.801938
epoch 9: F=-2813.577367
epoch 10: F=-2809.330760

Training from corpus sp.1k
epoch 1: F=-2843.774148
epoch 2: F=-2793.530913
epoch 3: F=-2765.064986
epoch 4: F=-2746.394647
epoch 5: F=-2732.984146
epoch 6: F=-2722.782219
epoch 7: F=-2714.721207
epoch 8: F=-2708.179284
epoch 9: F=-2702.764254
epoch 10: F=-2698.213419

(d) For cross-entropy, we tested several english dev files with different length (from 10 to 500), here is the result:

C	Training file	length-10	length-20	length-50	length-100	length-200	length-500
0.05	en.1K	5.2135	5.0921	4.1851	5.8113	4.5030	4.3008
0.1	en.1K	5.2114	5.0891	4.1851	5.8059	4.5014	4.2996
0.5	en.1K	5.1959	5.0666	4.1866	5.7649	4.4899	4.2910
1	en.1K	5.1790	5.041	4.1890	5.7188	4.4775	4.2820
2	en.1K	5.1515	4.9972	4.1956	5.64	4.4583	4.2683

By testing different value of C on language identification task with training set en.1K and sp.1K we have the following result:

C	Error Rate
0.05	24.70%
0.1	24.70%
0.5	24.28%
1	23.02%
2	23.02%
5	22.60%
7	20.92%
8	20.08%
9	20.08%
10	20.50%
11	20.50%
12	20.91%
19	20.49%
20	20.49%
21	20.49%
22	20.08%
23	20.50%
24	20.50%
25	20.50%
50	23.01%

From the result we found that $C^* = 8$, using this value, we experimented lexicons with different dimensions and different training files.

dimension/training file	1K	2K	5K	10K
10	20.97%	27.61%	16.65%	17.03%
20	17.59%	35.99%	9.20%	10.41%
40	19.75%	39.64%	9.20%	5.844%

(e)

(f) Taking $C = 8$ as before, we have $\beta = 1.074$ learned from `en.1K`.

This new feature helps the model produce a slightly lower cross-entropy, here is some result when setting $C = 1$ and $C = 2$ for comparison with the original model:

C	Training file	length-10	length-20	length-50	length-100	length-200	length-500
1	en.1K	5.0659	4.9469	4.1089	5.2517	4.2979	4.1329
2	en.1K	5.0654	4.9208	4.1153	5.1946	4.2701	4.1169

For the perofomance of language identification, we also have a slight improvement on most cases. Here is a list of error rate with $C = 8$:

dimension/training file	1K	2K	5K	10K
10	20.56%	29.36%	14.34%	10%
20	16.91%	37.48%	9.60%	10.69%

(g) We chose to implement the first improvement here.

At first we tried adding a binary feature f_w for each word in the vocabulary, by testing the model on language identification task again we can see a further improvement on the accuracy:

dimension/training file	1K	2K	5K	10K
10	20.43%	25.43%	14.88%	10.82%

Then we tried to add a binary feature for each bigram and trigram that appears at least 3 times in the training data. Here is some of the test result when taking $C = 8$:

dimension/training file	1K	2K	5K	10K
10	21.24%	17.05%	%	%

We can tell from the result above that generally this new model works better than all other models above. This is based on the fact that we didn't change C for the model. When we go back to dev file to tune the C for this particular model again, we found that this model can even have a better performance by taking a smaller C like 0.001. But due to time limit, we didn't test all data here.

7. Originally we assume the priori of a word being English or Spanish ar e same so we classify the words directly by calculating their likelihood. Now we have $p(lang = English) = 2/3$ and $p(lang = Spanish) = 1/3$ as our priori.

$$p(lang = English \mid word = W) = \frac{p(word = W \mid lang = English)}{p(word = W)} \cdot p(lang = English)$$
$$p(lang = Spanish \mid word = W) = \frac{p(word = W \mid lang = Spanish)}{p(word = W)} \cdot p(lang = Spanish)$$

By the above equations, we can compare the posteriori of English and Spanish by multiply the likelihood of English by 2 while keep the likelihood of Spanish the same, then comparethe two values to determine the language of this word. Simply we have:

$$\frac{p(lang = English \mid word = W)}{p(lang = Spanish \mid word = W)} = \frac{2p(word = W \mid lang = English)}{p(word = W \mid lang = Spanish)}$$

We don't need to know the priori of the document being in English or Spanish, because when we train the model, we are actually trying to maximize the likelihood of the training data. E.g. when training on English data, we want to maximize $p(word = W \mid lang = English)$, which has nothing to do with the priori.

By implementing this change we found a slight improvement on the performance on the language identification task. Here are the result on various smoothing methods:

ADDL:
English:
`113 looked more like en.1K (94.17%)`
`7 looked more like sp.1K (5.83%)`

Spanish:
`11 looked more like en.1K (9.24%)`
`108 looked more like sp.1K (90.76%)` The overall error rate is 7.54%, while the one for unmodified version is 9.1%.

BACKOFF_ADDL:
English:
`112 looked more like en.1K (93.33%)`
`8 looked more like sp.1K (6.67%)`

Spanish:
`31 looked more like en.1K (26.05%) 88 looked more like sp.1K (73.95%)` The overall error rate is 16.36%, while the original one is 15.7%. This only outperformed the former one on classifying English.

LOGLIN (with $C = 8$):
English:
`98 looked more like en.1K (81.67%)`
`22 looked more like sp.1K (18.33%)`

Spanish:
`30 looked more like en.1K (25.21%)`
`89 looked more like sp.1K (74.79%)` The overall error rate is 21.77% which is close to the original one.

8. (a) We want to pick one sentence that has highest probability given the utterance, $p(\vec{w} \mid U)$, from the 9 candidates. By Bayes's Theorem, we can compute $p(\vec{w} \mid U) \propto p(U \mid \vec{w}) \times p(\vec{w})$. Since we already have $\log_2 p(U \mid \vec{w})$, we can compute $\log_2 p(\vec{w} \mid U) = \log_2\{p(U \mid \vec{w}) \times p(\vec{w})\} = \log_2 p(U \mid \vec{w}) + \log_2 p(\vec{w})$.

   (c) **Using test/easy with smoother backoff_add0.01 and 3-gram model**

0.100 easy061	0.143 easy083	0.250 easy105	0.083 easy127	0.231 easy149
0.364 easy062	0.111 easy084	0.143 easy106	0.167 easy128	0.125 easy150
0.143 easy063	0.167 easy085	0.167 easy107	0.111 easy129	0.100 easy151
0.062 easy064	0.200 easy086	0.125 easy108	0.179 easy130	0.091 easy152
0.100 easy065	0.111 easy087	0.267 easy109	0.059 easy131	0.231 easy153
0.000 easy066	0.000 easy088	0.111 easy110	0.167 easy132	0.167 easy154
0.105 easy067	0.125 easy089	0.333 easy111	0.091 easy133	0.167 easy155
0.125 easy068	0.118 easy090	0.167 easy112	0.154 easy134	0.167 easy156
0.125 easy069	0.350 easy091	0.294 easy113	0.167 easy135	0.182 easy157
0.100 easy070	0.000 easy092	0.091 easy114	0.095 easy136	0.000 easy158
0.143 easy071	0.167 easy093	0.000 easy115	0.053 easy137	0.167 easy159
0.200 easy072	0.100 easy094	0.167 easy116	0.091 easy138	0.136 easy160
0.083 easy073	0.182 easy095	0.077 easy117	0.158 easy139	0.167 easy161
0.250 easy074	0.059 easy096	0.143 easy118	0.071 easy140	0.167 easy162
0.167 easy075	0.095 easy097	0.143 easy119	0.111 easy141	0.125 easy163
0.077 easy076	0.000 easy098	0.182 easy120	0.048 easy142	0.111 easy164
0.143 easy077	0.182 easy099	0.286 easy121	0.250 easy143	0.091 easy165
0.062 easy078	0.136 easy100	0.143 easy122	0.429 easy144	0.167 easy166
0.133 easy079	0.211 easy101	0.182 easy123	0.143 easy145	
0.167 easy080	0.125 easy102	0.194 easy124	0.143 easy146	
0.182 easy081	0.154 easy103	0.083 easy125	0.182 easy147	
0.059 easy082	0.100 easy104	0.182 easy126	0.333 easy148	

0.141 OVERALL

Using test/easy with smoother backoff_add0.01 and 2-gram model

0.100 easy061	0.308 easy076	0.350 easy091	0.143 easy106	0.286 easy121
0.364 easy062	0.143 easy077	0.000 easy092	0.167 easy107	0.429 easy122
0.143 easy063	0.062 easy078	0.167 easy093	0.125 easy108	0.182 easy123
0.125 easy064	0.133 easy079	0.200 easy094	0.267 easy109	0.194 easy124
0.100 easy065	0.333 easy080	0.200 easy095	0.111 easy110	0.167 easy125
0.000 easy066	0.182 easy081	0.059 easy096	0.333 easy111	0.182 easy126
0.105 easy067	0.059 easy082	0.095 easy097	0.167 easy112	0.083 easy127
0.125 easy068	0.143 easy083	0.000 easy098	0.294 easy113	0.167 easy128
0.125 easy069	0.111 easy084	0.182 easy099	0.091 easy114	0.000 easy129
0.167 easy070	0.167 easy085	0.136 easy100	0.091 easy115	0.179 easy130
0.429 easy071	0.200 easy086	0.211 easy101	0.167 easy116	0.059 easy131
0.267 easy072	0.111 easy087	0.125 easy102	0.077 easy117	0.250 easy132
0.125 easy073	0.133 easy088	0.077 easy103	0.286 easy118	0.091 easy133
0.250 easy074	0.187 easy089	0.100 easy104	0.143 easy119	0.154 easy134
0.167 easy075	0.118 easy090	0.250 easy105	0.182 easy120	0.333 easy135

0.238 easy136	0.250 easy143	0.125 easy150	0.273 easy157	0.111 easy164
0.158 easy137	0.143 easy144	0.100 easy151	0.000 easy158	0.091 easy165
0.091 easy138	0.143 easy145	0.182 easy152	0.167 easy159	0.167 easy166
0.158 easy139	0.143 easy146	0.154 easy153	0.182 easy160	
0.071 easy140	0.182 easy147	0.333 easy154	0.167 easy161	
0.111 easy141	0.111 easy148	0.333 easy155	0.167 easy162	
0.048 easy142	0.154 easy149	0.167 easy156	0.125 easy163	

0.160 OVERALL

Using test/easy with smoother backoff_add0.01 and 1-gram model

0.100 easy061	0.357 easy083	0.125 easy105	0.250 easy127	0.308 easy149
0.182 easy062	0.222 easy084	0.429 easy106	0.167 easy128	0.250 easy150
0.571 easy063	0.500 easy085	0.167 easy107	0.222 easy129	0.000 easy151
0.312 easy064	0.267 easy086	0.125 easy108	0.179 easy130	0.091 easy152
0.200 easy065	0.556 easy087	0.267 easy109	0.176 easy131	0.231 easy153
0.167 easy066	0.133 easy088	0.111 easy110	0.250 easy132	0.167 easy154
0.263 easy067	0.250 easy089	0.500 easy111	0.182 easy133	0.000 easy155
0.500 easy068	0.176 easy090	0.167 easy112	0.231 easy134	0.333 easy156
0.125 easy069	0.450 easy091	0.294 easy113	0.333 easy135	0.273 easy157
0.167 easy070	0.111 easy092	0.091 easy114	0.190 easy136	0.167 easy158
0.000 easy071	0.167 easy093	0.182 easy115	0.211 easy137	0.417 easy159
0.267 easy072	0.400 easy094	0.167 easy116	0.091 easy138	0.182 easy160
0.125 easy073	0.182 easy095	0.231 easy117	0.158 easy139	0.167 easy161
0.250 easy074	0.059 easy096	0.286 easy118	0.107 easy140	0.500 easy162
0.333 easy075	0.143 easy097	0.286 easy119	0.000 easy141	0.375 easy163
0.308 easy076	0.333 easy098	0.182 easy120	0.048 easy142	0.667 easy164
0.143 easy077	0.182 easy099	0.286 easy121	0.125 easy143	0.091 easy165
0.187 easy078	0.227 easy100	0.429 easy122	0.143 easy144	0.167 easy166
0.467 easy079	0.211 easy101	0.182 easy123	0.429 easy145	
0.333 easy080	0.125 easy102	0.161 easy124	0.214 easy146	
0.182 easy081	0.231 easy103	0.167 easy125	0.273 easy147	
0.294 easy082	0.400 easy104	0.182 easy126	0.111 easy148	

0.222 OVERALL

using test/unrestricted with smoother add0.01 and 3-gram model

0.370 speech061	0.167 speech075	0.833 speech089	0.000 speech103	0.250 speech117
0.227 speech062	0.818 speech076	0.250 speech090	0.222 speech104	0.778 speech118
1.000 speech063	1.000 speech077	0.000 speech091	1.000 speech105	0.875 speech119
0.000 speech064	0.000 speech078	0.714 speech092	0.000 speech106	0.000 speech120
0.250 speech065	0.000 speech079	0.000 speech093	0.000 speech107	1.000 speech121
0.231 speech066	0.000 speech080	0.000 speech094	0.500 speech108	1.000 speech122
1.000 speech067	2.000 speech081	0.417 speech095	1.000 speech109	1.000 speech123
0.415 speech068	0.000 speech082	1.000 speech096	1.000 speech110	0.556 speech124
1.000 speech069	0.917 speech083	0.294 speech097	0.000 speech111	0.500 speech125
0.583 speech070	0.596 speech084	0.474 speech098	0.000 speech112	0.167 speech126
0.200 speech071	0.000 speech085	0.273 speech099	1.000 speech113	1.000 speech127
1.000 speech072	0.800 speech086	0.143 speech100	0.154 speech114	0.154 speech128
0.125 speech073	0.133 speech087	0.000 speech101	0.000 speech115	0.333 speech129
0.286 speech074	0.000 speech088	0.400 speech102	0.250 speech116	0.500 speech130

0.000 speech131	1.000 speech139	0.000 speech147	0.750 speech155	0.375 speech163
0.333 speech132	1.000 speech140	0.208 speech148	1.500 speech156	0.538 speech164
0.000 speech133	0.000 speech141	0.286 speech149	0.000 speech157	0.364 speech165
0.500 speech134	1.000 speech142	0.083 speech150	0.500 speech158	0.000 speech166
0.324 speech135	0.429 speech143	0.500 speech151	0.000 speech159	
0.625 speech136	0.333 speech144	2.000 speech152	1.000 speech160	
0.467 speech137	0.267 speech145	0.500 speech153	1.000 speech161	
0.000 speech138	0.375 speech146	0.500 speech154	0.786 speech162	

0.382 OVERALL

using test/unrestricted with smoother add0.01 and 2-gram model

0.407 speech061	0.917 speech083	1.000 speech105	0.923 speech127	0.286 speech149
0.273 speech062	0.596 speech084	0.000 speech106	0.154 speech128	0.083 speech150
0.000 speech063	0.000 speech085	0.000 speech107	0.417 speech129	0.437 speech151
0.000 speech064	0.800 speech086	0.875 speech108	0.500 speech130	2.000 speech152
0.500 speech065	0.233 speech087	1.000 speech109	0.083 speech131	1.000 speech153
0.308 speech066	0.000 speech088	0.000 speech110	0.667 speech132	0.500 speech154
1.000 speech067	0.833 speech089	0.000 speech111	0.222 speech133	0.500 speech155
0.439 speech068	0.250 speech090	0.000 speech112	0.500 speech134	1.000 speech156
0.333 speech069	0.000 speech091	0.000 speech113	0.405 speech135	0.000 speech157
0.500 speech070	0.714 speech092	0.077 speech114	0.625 speech136	0.500 speech158
0.000 speech071	0.000 speech093	1.000 speech115	0.467 speech137	1.000 speech159
0.000 speech072	0.250 speech094	0.500 speech116	0.500 speech138	1.000 speech160
0.125 speech073	0.417 speech095	0.250 speech117	1.000 speech139	0.000 speech161
0.000 speech074	1.000 speech096	0.667 speech118	1.000 speech140	0.786 speech162
0.167 speech075	0.294 speech097	0.937 speech119	0.000 speech141	0.375 speech163
0.818 speech076	0.421 speech098	0.000 speech120	1.000 speech142	0.615 speech164
0.000 speech077	0.182 speech099	0.000 speech121	0.357 speech143	0.636 speech165
0.000 speech078	0.286 speech100	1.000 speech122	0.667 speech144	0.000 speech166
0.000 speech079	0.000 speech101	1.000 speech123	0.267 speech145	
1.000 speech080	0.400 speech102	0.222 speech124	0.500 speech146	
0.000 speech081	0.000 speech103	0.500 speech125	0.000 speech147	
0.000 speech082	0.444 speech104	0.167 speech126	0.292 speech148	

0.419 OVERALL

using test/unrestricted with smoother add0.01 and 1-gram model

0.333 speech061	0.571 speech074	0.233 speech087	0.286 speech100	0.000 speech113
0.273 speech062	0.167 speech075	0.000 speech088	0.000 speech101	0.231 speech114
0.000 speech063	0.818 speech076	1.167 speech089	0.600 speech102	1.000 speech115
1.000 speech064	0.000 speech077	0.250 speech090	0.067 speech103	0.500 speech116
0.500 speech065	0.000 speech078	0.000 speech091	0.222 speech104	0.250 speech117
0.308 speech066	0.000 speech079	0.714 speech092	1.000 speech105	0.778 speech118
1.000 speech067	1.000 speech080	0.500 speech093	1.000 speech106	0.812 speech119
0.439 speech068	1.000 speech081	0.250 speech094	0.000 speech107	0.000 speech120
0.333 speech069	0.400 speech082	0.417 speech095	0.500 speech108	0.000 speech121
0.417 speech070	0.917 speech083	1.000 speech096	1.000 speech109	1.000 speech122
0.200 speech071	0.615 speech084	0.529 speech097	0.000 speech110	1.000 speech123
0.000 speech072	0.000 speech085	0.421 speech098	1.500 speech111	0.222 speech124
0.125 speech073	0.800 speech086	0.364 speech099	0.000 speech112	0.500 speech125

0.167 speech126	0.405 speech135	0.667 speech144	1.000 speech153	0.786 speech162
0.923 speech127	0.625 speech136	0.267 speech145	0.500 speech154	0.375 speech163
0.154 speech128	0.667 speech137	0.375 speech146	1.000 speech155	0.462 speech164
0.250 speech129	0.500 speech138	0.000 speech147	1.500 speech156	0.364 speech165
0.500 speech130	1.000 speech139	0.208 speech148	0.500 speech157	0.000 speech166
0.083 speech131	1.000 speech140	0.571 speech149	0.500 speech158	
0.333 speech132	0.000 speech141	0.083 speech150	0.250 speech159	
0.222 speech133	1.000 speech142	0.562 speech151	1.000 speech160	
0.500 speech134	0.214 speech143	2.000 speech152	1.000 speech161	

0.438 OVERALL

We used add-λ for easy and backoff_addλ for unrestricted just because we simply picked the one that perform better under the same λ value. We didn't test loglinear smoother because it takes much longer time for training and we didn't have enough time for its result.

9.

10. (a) We have $p_{disc}(z \mid xy) = \frac{c(xyz)}{c(xy)+T(xy)}$. Thus, when $T(xy)$ is close to 0, $p_{disc}(z \mid xy)$ will be very close to the naive historical estimate $c(xyz)/c(xy)$. It means when there are few or zero word types z that have been observed to follow the context xy, $p_{disc}(z \mid xy)$ will have similar result as $c(xyz)/c(xy)$. In other words, when xy appears in the context, it will be very likely followed by one of the few word types z.

(b) When all $T()$ become zero, all $p_{disc}()$ will become naive historical estimates.
When all $T()$ become zero, α should become zero in order to make the distribution sum to 1.

(c) $\alpha() = \frac{1-\sum_{z:c(z)>0} p_{disc}(z)}{V-T()}$

(d) To make $\sum_z \hat{p}(z \mid y) = 1$, we have following equation:

$$1 = \sum_z \hat{p}(z \mid y) = \sum_{z:c(yz)>0} P_{disc}(z \mid y) + \sum_{z:c(yz)=0} \alpha(y)\hat{p}(z)$$
$$= \sum_{z:c(yz)>0} P_{disc}(z \mid y) + \alpha(y)\Big(1 - \sum_{z:c(yz)>0} \hat{p}(z)\Big)$$

By solving the equation above, we will need:

$$\alpha(y) = \frac{1 - \sum_{z:c(yz)>0} p_{disc}(z \mid y)}{1 - \sum_{z:c(yz)>0} \hat{p}(z)}$$

(e)

(f) By the given formulation, we have:

$$1 = \sum_{z:c(yz)>0} p_{disc}(z \mid y) + \alpha(y)\Big(1 - \sum_{z:c(yz)>0} p(z)\Big)$$

$$\alpha(y) = \frac{1 - \sum_{z:c(yz)>0} p_{disc}(z \mid y)}{1 - \sum_{z:c(yz)>0} \hat{p}(z)}$$

We can use the same method to simplify the denominator in the above equation:

$$\sum_{z:c(yz)>0} \hat{p}(z) = \sum_{z:c(yz)>0} p_{disc}(z) = \frac{\sum_{z:c(yz)>0} c(z)}{c() + T()}$$

(g) We implemented Witten-bell backoff in the "speechrec.py" file and be called by "textcat2.py" file. The result of the same experiment as 3(c) are:
Witten-bell backoff on English classification
343 looked more like en.1K (92.70%)
27 looked more like sp.1K (7.30%)

Witten-bell backoff on Spanish classification
67 looked more like en.1K (18.16%)
302 looked more like sp.1K (81.84%)

Thus the error rate of using Witten-Bell backoff is $94/739 = 0.127$, which is not as good as the error rate, 0.091, of ADD-λ.