ANLP Assignment 1 2018
Solution Notes and Marking Information

This document provides some notes on the solutions to Assignment 1, along with information about how we are marking it. It is based on a subset of submissions that we've looked at so far.

Individual feedback and marks will be provided once the marking is done.

General Points

Some students lost marks due to difficulty with terminology. To set things straight:

- *Relative frequency estimation* is equivalent to *Maximum Likelihood Estimation* for the models we have considered so far. (Though this is not necessarily true in general: relative frequency estimation is defined through the procedure of taking relative frequencies; whereas MLE is defined as the estimate that maximizes the likelihood. In n-gram models, HMMs, and PCFGs, these end up being equivalent.)
- Some students referred to smoothed estimates as MLE also. However, this is incorrect: once you are using smoothing, it is no longer MLE. You could get away with saying something like "we used smoothed MLE estimates" because at least that is clear you are *not* actually using straight MLE. But it's better to simply say "we used smoothed estimates", "we used add-alpha smoothing to estimate probabilities", or something similar.
- All of these (MLE, add-alpha, Good-Turing, etc.) are different estimation methods.

Task 1

Most people got full or nearly full marks on this question. We were looking for some recognition that you had considered what to do about newline characters (i.e., end of sentence, since the training data had one sentence per line). You could either insert special sentence boundary markers (as in our toy example and model-br.en), or replace the newline with a space which at least provides a special context (period followed by space) for choosing what character to begin the next sentence with. You should also explain why: if you don't, then when generating from the model (later question), there's no proper history to condition on for the first two characters. (Additional pros/cons might be mentioned as well.)

Task 2

To get full marks on this question, your answer needs to identify add-alpha smoothing as the most likely, with good justification, likely including some specific examples of trigrams you looked at. In particular:

- MLE is ruled out (i.e., some form of smoothing is used) because there are no zero probabilities.

- Backoff/interpolation are very unlikely because so many unlikely trigrams have the same probability, regardless of their final bigram. For example, the trigrams `_c.`, `_cv`, and `_cy` (where `_` represents a space) all have the same probability. This could happen under add-alpha or Good-Turing if all three are unseen in the training data (which is plausible). But using backoff or interpolation, the probabilities would be estimated by combining the trigram information with bigram information: specifically, the estimated bigram probabilities $P(\texttt{.}|\texttt{c})$, $P(\texttt{v}|\texttt{c})$, and $P(\texttt{y}|\texttt{c})$. But notice that, unlike the trigrams, these are *not* likely to all have zero counts, because it's perfectly reasonable to have a `c` before a full stop, or a `cy` in a word (probably at the end of the word), whereas `cv` even as a bigram is very bad. So using backoff or interpolation, we'd probably end up estimating $P(\texttt{\_}|\texttt{c.})$ and $P(\texttt{\_}|\texttt{cy})$ as higher than $P(\texttt{\_}|\texttt{vc})$.

- This leaves add-one (Laplace), add-alpha, Good-Turing (and Kneser-Ney) as possibilities. (Some people made the jump from saying add-alpha is possible, to concluding that add-alpha is the right answer. Please be careful with your reasoning: simply showing that all your evidence fits one hypothesis is not the same as showing that other hypotheses are ruled out.)

If your reasoning was clear and got as far as that, you will have got most of the marks.

But you can go further. For example, there are a large number of trigrams that all have the same probability of 3.333e-02 = 1/30, and these look like unlikely trigrams with unlikely histories (for example, all the trigrams that begin with a space followed by a full stop). One can speculate that probably all of these trigrams, as well as their bigram history, have counts of 0. Under those circumstances, add-alpha smoothing would indeed yield a probability of 1/30. But Good-Turing (unless combined with backoff or interpolation) would actually be undefined, because it only adjusts the trigram counts (numerator) but doesn't change the bigram counts (denominator)!

In fact, I used add-alpha smoothing with alpha=0.1.

Task 3

Most people should expect to get between 21-33 marks on this question. A really good answer would have the following characteristics:

- Clear explanation with correct use of terminology (see General Points at the top) and appropriate scientific style (e.g., no "history-telling"—see the assignment handout if you're not sure what I mean).

- Must use some form of smoothing, and give a full justification. (i.e., just saying "we used smoothing to avoid zeros") is only part of the justification: *why* do you need to avoid zeros, especially in the context of this assignment? You need to say something about making predictions or computing perplexity on a test document.

- The type of smoothing isn't critical, though may get slightly more weight for using (say) add-alpha than add-one, or at least pointing out this would be better. A very good answer will try different alpha values, but on a validation set. Or at least acknowledge that a validation set should be used.

- Acknowledgement of the weaknesses of the chosen method, and mention some alternatives. Even better if you point out that using add-alpha is actually *not* as bad with characters as it is with words, because the "vocabulary" size is small.
- Define all terms in equations, and say what they refer to in this context (that is, they refer to character trigrams/bigrams, and V is the number of distinct characters, i.e. 30).
- Probabilities for `ng` should be non-zero for all 30 possible continuations.
- Comments on both some of the high-frequency continuations (like `ng_`)—ideally giving examples of real English words with these combinations—and some of the low-frequency ones.

Most answers do not satisfy all these points. Those that cover most of them will be closer to the high end of the range, while those that are weak on several points will be closer to the low end of the range. Marks of less than 21 usually mean an incomplete implentation, severe problems with the explanation, or failure to use any smoothing at all, possibly along with problems in clarity.

Task 4

Most people had the right general idea of what to do here but some didn't implement it quite right. For those who did not, I provide below an example of the kind of output we were looking for. To create it, we first learned the trigram character model from the training corpus (or read it in from model-br.en), and then generated each character in turn by sampling from the conditional probability distribution defined by the previous two characters. The key parts to getting this correct are understanding that you need to sample from the conditional distribution, not a joint distribution, and that because the history changes at each point, so does the distribution you are sampling from. You also need to make sure you deal with beginning of sequences correctly.

A good textual description should mention the following:

- something about how begin/end of sequence are handled.
- that the 2-char history is used to determine the distribution to sample from.
- that you generate by sampling from the distribution, not taking the most probable option.

If you used pseudocode to describe what you did, it should indicate the same points, and crucially needs to be easily understandable. Typically, that means a mixture of symbols and text; pseudocode that consists entirely of symbols/variables, especially if they are not very clearly explained, is rarely easy to understand. Pseudocode that looks basically like real code also often defeats the purpose of being easier to understand than real code.

The reason for asking you to generate 300 characters was so you'd have enough to see some of the differences. A few people stopped after generating a single sequence, which isn't enough to do this. Generating everything as a single sequence (replacing the end-of-sequence markers with nothing, instead of a newline) is also not ideal, since technically these are separate sequences and looking at the difference in sequence length is important.

Example output: First, here are some samples from the English trained model using different values of alpha smoothing. Notice that the larger values of alpha do give somewhat more rubbishy sequences because they are over-smoothed, so there are quite a few sequences that never occur in training (e.g., full stops in the middle of sentences). To emphasize that, I left the begin/end of sentence marks in so it's obvious where each "sentence" is.

Sample output from English training data (using add-1 smoothing):

```
##thie0ibil theri#
##uwhisionethe ber eurepelobidteurat hat wilit inat ing eur the mosjpgvdvout imrs
come do moswturaffolat of sidelos.#
##ing the ch ince tegive thanotend iwqmtnjill ther wis is mograiling the i diregiveles.#
##i do plqterglyjzons maiqm#
##we to thinlyps.#
##fich ineressionvated forchariatit thantly i .eced.#
##to beithavelo00#
##thisheral and pors ther by is sen a in thropme ta.o my it housfecievat pend oufsh
re droessin poreprourrecautgf.i it stake rei#
##the berest the sxmg#
##lv#
##the statiount mit ission
```

Sample output from English training data (using add-0.1 smoothing):

```
##fort whichilso reat foluroaday alloylpitand mort isz thessmajord vais aninstrut lest
it to a we rent withe expresitividesidam shoulary of to by re artart onotiont as is expe
dent he sin modiftion we is to as of the th and havrlight0vich trodqarlicle no a go exh.#
##tooke dit thowe pords.#
##we an ch the unt is con whe wilow isionothrovessiation to withatzihdral he and
strieve thally inly of ity reed tort whisbod inessagerds obill to wiliammithey a mrsents
cas a the comme eit an and licusiont.#
##the pri
```

Sample output from English training data (using add-0.01 smoothing):

```
##thes satich broppolis regainvey cas.#
##ity the prele.#
##itteduse the unif 00 sueraccon.#
##it ing of ition fuld i supple regalissidepreffethicatual malstropor lemorivion the the
that se on to asuchns tabill the ares on.#
##re of ce re cal beeposicy warregioncludy areverthicy loole hould eu wis objece an
ammit intral pardeld inticutive koche the and gre ans covito behe dis men lone derach
cis the ifir ancip apprievargenable sposent wount incleal ally cauct ime wardly the the
gurefor refor mat ther thet tha
```

Now, here's some sample output from model-br.en:

```
##th your a scrabyes.#
##cant there dow cou.#
##is one.#
```

```
##now ame spook to.#
##you kno.#
##that do you go.#
##he yone ant.#
##your.#
##a likes is twou fand ged one cand do.#
##here a like hi.#
##is for liff.#
##whats thely.#
##what callo.#
##thi kit you whatchy hos the a le come youseed say.#
##wand howere.#
##se mou we bigby.#
##thaid towele.#
##a seep.#
##kin dooke thats rie one put.#
##its say therose like and book does top.#
##what ase for sed he this.#
##brus.#
##whattle thats ards a areephow wits lent ats in put down.#
##it me on.#
##right smay do gray.#
##te book
```

To get most of the marks in this question, your generated sequences should look roughly like these, you should have a clear and concise explanation of what you did to generate them, and you should have mentioned a few of the following points:

- model-br produces shorter words and shorter sentences than your trained model.
- Its words also look simpler, with more common character sequences.
- You should propose a possible explanation, which could be that we used a better smoothing method, but seems more likely to be due to a different training corpus that has different properties.

To get full marks, we would like to see a bit more thought/analysis. Ideally you might have tried generating more sequences from model-br, and if you did you might start to notice that it generates a lot of sequences like 'dog', 'doggie', 'daddy', 'hello', 'what', 'book', etc. This fact, together with the other properties noted above, suggests that the training corpus was nothing like the Europarl data. In fact we used a transcript of child-directed speech (much like the data from Lab 1, but cleaned up a bit).

Task 6

To do well here, you need to:

- Get the right perplexity values
- Say that you can guess the language of the test doc according to the training language which produced lowest perplexity value
- Include some reasonable discussion about the perpexity of a new document (see below).

Some notes on issues people had:

- Quite a few submissions have the perplexity values wrong. The right values will depend on exactly which estimation method you used, but in general the value for English will be around 8-10 and values for Spanish and German will be above 20. In other words, there is an obvious difference between the right language (English) and the other ones.

  Many people tested their perplexity function on the simple example from Q0. This is a good start, but may not be sufficient. Whenever you write code, it's a good idea to consider what range of values you expect to get in the results. In this case, you might reason as follows:

  - There are 30 different characters. A uniform distribution over these would have entropy (average negative log prob) of $\log_2 30$, or about 5. Perplexity is based on *cross-entropy*: the average negative log prob of some data under my model. A perplexity of 30 (cross-entropy of about 5) would mean that my model is about as good at predicting the sequence it sees as a random uniform model would be at predicting a sequence of random characters uniformly distributed among 30 possible characters. In other words, my model's predictions on *this* data are about as good as the *best* model's predictions could be on a completely unpredictable sequence over the same number of characters.
  - This line of reasoning should lead you immediately to to the conclusion that any reasonable model should have a perplexity (considerably) lower than 30 on data that looks similar to what it was trained on (since the data is not actually uniform random).
  - On the other hand, if the model is presented with data that are very dissimilar to what it was trained on, then its predictions could actually be as bad or worse than a random model's predictions on random data. So it wouldn't be totally surprising to get perplexities close to 30 for dissimilar data.
  - Also, suppose (as many students did) you end up with perplexities that are extremely similar for the three test models: say, 9.1, 9.2, and 9.3. These are saying that your three models' predictions are as good as a uniform random model on uniform random sequences containing (respectively) 9.1, 9.2, or 9.3 distinct characters. But the difference between 9.1 and 9.2 distinct characters is not much at all. So it should seem implausible that your models, which are trained on completely different languages, have only that much difference in perplexity. This should give you a hint that either you're computing perplexity wrong, or you estimated your models incorrectly, or both.
  - Finally, perplexity *cannot* be less than 1, because entropy cannot be less than 0. And if you're even getting values just a bit bigger than 1, you should be probably be suspicious that they're too low, and again go looking for a bug (or convince yourself somehow that you are actually correct).

- Most people realized that zero probabilities present a problem for computing perplexity. However, it is not correct to simply skip over them. Doing so is not only incorrect according to the definition of perplexity, but also presents practical problems. Consider that for a test document in a language that doesn't match the training document, there are likely to be many more unseen trigrams than there are in a test document from a matching language. If you simply skip over them, you are losing a huge amount of information regarding the fact that your test document doesn't look like your training document. You could actually end up concluding that the non-matching test document has better perplexity than the matching document, simply because you are ignoring so much of the non-matching information. The zeros are telling you something: your model predicts that something *never* happens, but actually it does. So you get infinite perplexity because your model is infinitely wrong. The problem is simply that you can't make sensible comparisons between different things that are all infinitely wrong.

- Most people stated that perplexity could not be used to determine the language of a new test document because perplexity is only a relative measure. This is correct if *all* you know is the perplexity of the new test document (let's call it d_1 under your English LM. However, if you have already run your program on the test document provided with the assignment (d_0) after training on the German and Spanish data, you would also know the perplexity of that d_0 under the German and Spanish language models. As noted above, there is a pretty big difference in perplexity between the three models on d_0. Given that information, it might be possible to tell fairly accurately whether d_1 is also in English or not: is its perplexity under the English LM close to that of d_0, or is it closer to the perplexity of d_0 under the German or Spanish models, which we might take as typical "cross-linguistic" perplexity values?

  An even better answer might suggest (or even try) ways to estimate how much variability in perplexity might occur for different English test documents. You might, for example, try some new documents yourself, or you might compute the perpelxity of the given document under model-br.en. If you did, you'll notice it's a lot higher than under the model from the English training data! For that reason, using the very basic smoothing methods here, it probably *isn't* possible to correctly identify the language of a test document. But if we were using a better model/estimation method (e.g., higher order n-grams with better smoothing), *and* we knew the genre of text, we almost certainly would be able to.[1]

  Answers that just mention the first sentence above ("entropy is relative") get some credit, mainly because the question is ambiguous about what information is available. But for many answers it would be nice to see bit more consideration.

Task 6

A relatively small proportion of pairs attempted this question. Simply implementing a more complex method is not sufficient for many (if any) marks by itself: as noted in the assignment,

[1]Language identification for multi-genre text is much harder, but does use some of the ideas we've explored here and in the rest of this course. For example, see *langid.py: An Off-the-shelf Language Identification Tool* (Liu and Baldwin, 2012): `www.aclweb.org/anthology/P12-3005`

you must address a specific *question*. It's even better if you propose a hypothesis about what you expect to find. I would expect to see things like exploring the effect of different orders of n-grams, more complex smoothing methods, optimizing α, and looking at other data sets. Marks are based on the difficulty of the task, a clear and correct description of the method with evidence of correct implementation, and clearly presented results and discussion.