# ANLP Assignment 1

S1866666 S1879523

October 2018

## Contents

## 1 Preprocess Line

Defined below is the function `preprocess_line`.

```
def preprocess_line(line):
  processed = ''
  for char in line.lower():
    if char in " .abcdefghijklmnopqrstuvwxyz":
      processed+=char
    elif char.isdigit():
      processed+='0'
  return "##"+processed.strip()+'#'
```

python code for `preprocess_line`

In addition to filtering the characters, our codes adds two hashes to the beginning of the line and one hash to the end. The purpose of this addition is to make it feasible that in training our trigram model we can estimate the conditional probabilities of the two characters leading a sequence and estimate the end of a sequence. For example, the probability of $w_1$ being the first character of a sequence and the probability of end-of-sequence can be obtained using maximum likelihood estimation (MLE)[1] by

$$P_{ML}(\text{<s>}|c_1) = P_{ML}(\texttt{\#\#}|c_1) = \frac{\text{Count}(\texttt{\#\#}c_1)}{\text{Count}(\texttt{\#\#})} \tag{1}$$

$$P_{ML}(c_{i-2}c_{i-1}|\text{</s>}) = P_{ML}(c_1c_2\texttt{\#}) = \frac{\text{Count}(c_1c_2\texttt{\#})}{\text{Count}(c_1c_2)} \tag{2}$$

In this, "<s>" denotes a beginning-of-sequence and "</s>" an end-of-sequence.
Presented in `cw1.py` is a slightly different version of this function which retains the exact functionality the one shown above has but generalizes to n-gram case instead of trigram only.

## 2 Model Analysis

We evaluated that estimation method used in producing the probabilities was an add-alpha smoothing method. Our evidence for this comes from observations of the trigram probabilities, particularly those $P(X|\text{qw})$, $X \in \{\text{a,b,c,...,y,z,...}\}$, all of which share a common probability (see Table 1).

| | | |
|---|---|---|
| $P(\ |\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{g}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{q}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\#|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{h}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{r}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(.|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{i}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{s}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(0|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{j}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{t}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{a}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{k}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{u}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{b}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{l}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{v}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{c}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{m}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{w}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{d}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{n}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{x}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{e}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{o}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{y}|\text{qw}) = 3.333\text{e}{-02}$ |
| $P(\text{f}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{p}|\text{qw}) = 3.333\text{e}{-02}$ | $P(\text{z}|\text{qw}) = 3.333\text{e}{-02}$ |

Table 1: `An excerpt from model-br.en showing the equality of probabilities of trigrams with history "qw".`

Due to linguistic, orthographic restrictions of the language and the rules surrounding the written /q/ and /w/, "qw" clusters are impossible in English, except perhaps in a very rare case of an unlikely proper noun. It's therefore very likely that there would have been no instances of this cluster in the data. Add-alpha smoothing compensates for this by artificially increasing the count of each triphone once to avoid problematic zero probabilities. The low yet uniform probabilities of the offending "qw$X$" clusters is evidence that this may have taken place. Interpolation and back-off smoothing can be discounted as the use of either method would result in variation in probabilities for different "qw$X$" trigrams due to the differences in the bigrams (i.e. "w$X$")and unigrams of the third letters. Kneyser-Ney would also show variation across the "qw$X$" trigrams due to differing histories counts of the third letter. Maximum Likelihood Estimation can be discounted as it would show examples of zero probabilities in the results, none of which do occur in our data.

## 3 Model Development

The method we used to estimate probabilities in our own model was add-alpha smoothing. Add-alpha smoothing involves including a pseudocount to avoid zero frequency in MLE. Add-alpha is an advancement on add-one (or Laplace) smoothing, which also adds a pseudocount though as the name suggests, the pseudocount in add-one smoothing is always equal to one. As the count is adjusted, the dominator in the equation for a psuedocount smoothing also needs to be adjusted to account for the additional increment in the vocabulary. An additional complication with add-alpha smoothing is the added challenge of optimising the alpha value of the pseudocount.[3] Details about our methods in choosing the most effective alpha value are explained in some detail below. Avoiding zero frequency occurrences like this prevents zero probabilities for unseen sequences in the test set, thus allowing us to compute perplexity effectively.[2] The issue with zero probability occurrences in a test set is a mathematical one. The nature of the calculation to establish perplexity is such that a single n-gram with a zero probability will corrupt the whole test sentence, making its entire probability zero. This is why some kind of smoothing method is necessary for us to effectively estimate these probabilities.[3] Also, it eases the potential over-fitting issue posed by MLE.[1] The probability of each character is given by

$$P(c_i|c_{i-2}c_{i-1})_{+\alpha} = \frac{\text{Count}(c_{i-2}c_{i-1}c_i) + \alpha}{\text{Count}(c_{i-2}c_{i-1}) + \alpha v} \tag{3}$$

In this, $v$ is the number of character choices given the history. In our case, $v = 30$ as there are 30 characters in our "vocabulary". (i.e. 26 lowercase alphabet, zero, space, ".", and end-of-sequence hash). Exceptionally, $v = 29$ if the history is "##", since the trigram "###" meaning "<s></s>"is not appropriate in our model. Add-alpha smoothing generalizes add-one smoothing to avoid its tendency to borrow too much probability from more frequently occurring trigrams and skew the distances in probability between low and high occurrence trigrams, so much as to offer an inaccurate and distorted portrayal of the realities of the probability distributions.

Circling back to our justification of choosing to use add-alpha smoothing and building on this, the following is a comprehensive description of how we chose the optimum alpha value to use. We employed the Monte Carlo cross-validation method to set the hyper-parameter $\alpha$ in order to avoid selection-bias. The training set was split 80:20 in each iteration, where 20% of it was used for validation. Candidate $\alpha$ were uniformly picked in the range of $(0, 1]$. Shown in Figure 1 is the result of the cross-validation

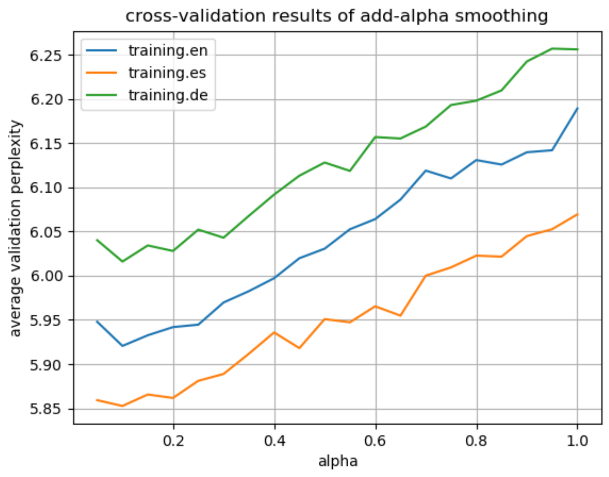

Figure 1: Plot of alpha against average perplexity of validation set over 20 iterations of Monte Carlo process for each training file

on different training files. Specifically, the lowest average validation perplexity over 20 iterations on `training.en` is given by $\alpha = 0.1$, which is therefore used to derive our model. Using the said procedure, the model trained from other two files are optimized at $\alpha = 0.1$ as well. An excerpt of our model is presented in Table 2, where we see no 0 probability as expected and there's a common probability $P = 1.302\text{e}{-04}$ assigned to trigrams very unlikely to occur, eg. "ngq", "ngz".

This is predictably contrasted by the much higher values given to trigrams with a vowel following the "ng" cluster, as these represent a natural and common environment for the "ng" cluster to appear in English. Though consistently higher than the unlikely "ngq" and "ngz" clusters, we can also see that there is variation between the probabilities of the "ng-(vowel)" clusters. The word final "ng " also has a particularly high probability of 0.8139. This is unsurprising given the amount of words in English which end in an "ng", common verbs and adjectives such as "bring" and "long", not to speak of the common "-ing" suffix. The (y—ng) cluster is a little surprising. It has the common probability for those trigrams unlikely to occur, but the cluster can't be said to be particularly alien to English or to be unlikely. Words such as "spongy" and "dingy" are certainly present in the language, but it is possible that they're uncommon enough as to not be present in the data used in establishing these probabilities. Aside from add-alpha smoothing, we also spent some time experimenting the interpolation method which turned out to give us results equivalent to the add-alpha model. Optimizing $\lambda$ using cross-validation while keeping $\alpha = 0.1$ gave us $\lambda = [0, 0, 1]$ for all three training sets, meaning that the

|  |  |  |
|---|---|---|
| $P(\ \|\text{ng}) = 8.139\text{e}{-01}$ | $P(\text{g}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{q}\|\text{ng}) = 1.302\text{e}{-04}$ |
| $P(\#\|\text{ng}) = 1.432\text{e}{-03}$ | $P(\text{h}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{r}\|\text{ng}) = 1.185\text{e}{-02}$ |
| $P(.\|\text{ng}) = 2.617\text{e}{-02}$ | $P(\text{i}\|\text{ng}) = 1.432\text{e}{-03}$ | $P(\text{s}\|\text{ng}) = 2.096\text{e}{-02}$ |
| $P(0\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{j}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{t}\|\text{ng}) = 1.315\text{e}{-02}$ |
| $P(\text{a}\|\text{ng}) = 2.734\text{e}{-03}$ | $P(\text{k}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{u}\|\text{ng}) = 2.734\text{e}{-03}$ |
| $P(\text{b}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{l}\|\text{ng}) = 2.734\text{e}{-03}$ | $P(\text{v}\|\text{ng}) = 1.302\text{e}{-04}$ |
| $P(\text{c}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{m}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{w}\|\text{ng}) = 1.302\text{e}{-04}$ |
| $P(\text{d}\|\text{ng}) = 4.036\text{e}{-03}$ | $P(\text{n}\|\text{ng}) = 1.432\text{e}{-03}$ | $P(\text{x}\|\text{ng}) = 1.302\text{e}{-04}$ |
| $P(\text{e}\|\text{ng}) = 8.737\text{e}{-02}$ | $P(\text{o}\|\text{ng}) = 6.641\text{e}{-03}$ | $P(\text{y}\|\text{ng}) = 1.302\text{e}{-04}$ |
| $P(\text{f}\|\text{ng}) = 1.432\text{e}{-03}$ | $P(\text{p}\|\text{ng}) = 1.302\text{e}{-04}$ | $P(\text{z}\|\text{ng}) = 1.302\text{e}{-04}$ |

Table 2: `An excerpt of add-alpha model estimated from training.en showing the probability of trigrams with history "ng".`

trigram probability gets weighted 1 whereas unigram and bigram get 0.

## 4 Random Sequence Generation

Provided below is the pseudo-code for `generate_from_LM`, and the random sequences generated using this process for two models: `model-br.en` and our add-alpha model trained from `training.en` with $\alpha = 0.1$. The hashes are removed for the sake of simple comparison.

```
function generate_from_LM(model):
  output <- "##"
  repeat until satisfied:
    char <- an X randomly drawn from distribution P(X'|output[-2:]) defined
        by model
    output += char
    if the generated char is "#" (i.e. output[-1]=="#"):
      start a new line in output
      output += "##"
  return output
```

`Pseudo-code for generate_from_LM`

```
whande.
i to your is cake bruck.
do peqxmh0wzcyre.
andy dognqzjbxhgue this a nim thice.
se.
pup on yealk its leedror.
a book tay.
kay.
what thelike is is of some thingry.
yes stere doing.
whe duchats that see.
is ther.
what aboon says have cat.
fing thissee nothe ch upsit.
thi.
you chat.
```

`Text generated by model-br.en`

```
the is plationclualical fortual ch ancluse trationd mix buttive thers the
    pand thiss dork.
the and terieseloyments.
rx a parep witherepropment need nommis andmembe thergiss weeport ing
    themenlithed ter bective fort speoproul.
```

```
4  thich exib zon funal an thery theres ack wht.
5  lat wely me mosioncresithat the is a per nal crampoin commis ad
6  ithey belly de on asidee the coment welsold cat the mort and theit voduce
       of accifyint of a to varest ing devent accoriburech fropeatund govis
       wevialudgembermada.
```

`Text generated by our model estimated from training.en`

The differences between the results of our model and the given model defined by `model-br.en` include that the sequences generated from the given model are much shorter than ours. We have hypothesized that this may be due to differences of formatting in the training sets. The one producing shorter sentences may be poetry, or some other format which produces short sequences, and thus the end-of-sequence occurs more frequently than in our training set. To see this, we randomly generated 10,000 characters from each model and computed unigram probabilities of "#" (which is appended only once to each sequence at the end) and space, shown in Table 3. We can also infer from the table that average word length in the training set of `model-be.en` is slightly shorter than ours. Although having more characters per space, once we take # as word boundaries, the training set of `model-be.en` has been shown to have the fewer characters per word boundary of the two.

|  | `model-br.en` | `training.en` |
|---:|---:|---:|
| $P(\#)$ | 0.05379 | 0.00820 |
| $P(\text{' '})$ | 0.14269 | 0.16008 |
| spaces per # | 2.653 | 19.522 |
| characters per # | 18.591 | 121.951 |
| characters per space | 7.008 | 6.247 |
| characters per word boundary | 5.090 | 5.942 |

Table 3: Statistics showing some differences in generated sequences from two models

Furthermore, there are increased occurrences of valid English words, or words with the appropriate morphology and letter clustering to be considered close to English or potentially valid English. An explanation for this could be that the words in its training set are shorter, thus trigram model fits the length of the words well, and therefore have more success modelling on them. It is worth further investigating whether generative trigram models can be shown to be more effective if trained from corpora with small average word length, a subject which we will discuss further in section 6: Further Investigation.

## 5 Language Perplexity

Perplexity is a metric for evaluating language models. Crucially, the higher the probabilities, the lower the perplexity. Therefore a lower perplexity means a higher test set probability according to the model.[3] The perplexity ($PP$) of a model `LM` on a test set $\vec{c}$ having $N$ trigrams is given by

$$PP_{\text{LM}}(\vec{c}) = P_{\text{LM}}(\vec{c})^{-\frac{1}{N}} = \prod_{c_i} P_{\text{LM}}(c_i|c_{i-2}c_{i-1})^{-\frac{1}{N}} \quad (4)$$

In practice, we evaluate log probability to ensure numerical accuracy in our code. The perplexity of the test document under each of our three language models is as follows:

English: 8.92
Spanish: 22.65
German: 24.25

We can use these numbers to guess at the language of the test document. As it has the lowest perplexity and therefore a higher probability[1], English is likely to be the language of the test document, however

we don't have enough information to be certain on this.
If we ran our program on a new test document and knew the perplexity under our English language model, we probably couldn't say conclusively whether the document was written in English. If it were in English, the language used (vocabulary, topic) might be very different from that with which we initially trained the language model. Combining this problem with the knowledge that English shares with German and Spanish a few linguistic similarities (affixations, shared words), we might struggle to prove in the event of a non-English document, that it is in fact not English, and not just English with a sufficiently different lexical group used. However, if we could repeatedly establish a close enough similarity to English compared with what we knew to be German and Spanish in repeated trials, we could probably gather enough evidence with enough trials to be fairly certain that such a consistent similarity is in fact meaningful.

## 6 Further Investigation

Our assumption is that n-gram models trained from corpora with shorter words tend to be more successful in random generation in terms of the amount of words in its training set being reproduced. We set up an experiment where we use n-gram models to randomly generated sequences, and count how many words in the sequences come from the training set, which we refer to as "real words" in the following content. We don't count punctuation ".", beginning- and end-of-sequence "#" as words to minimize the impact of sequence formatting. Shown in Table 4 is the experiment results collected from 100,000 characters generated by n-gram add-alpha models. By observation, we see that higher-gram

| training set | `training.en` | `training.es` | `test` | `training.de` |
|---:|---:|---:|---:|---:|
| average word length of training set | 4.736 | 4.794 | 5.231 | 5.760 |
| number of word type | 3212 | 4100 | 883 | 4242 |
| number of real word w/ unigram | 2084 | 3175 | 1343 | 1587 |
| number of real word w/ bigram | 2945 | 5608 | 1935 | 2488 |
| number of real word w/ trigram | 6465 | 7346 | 4089 | 4195 |
| number of real word w/ 4-gram | 9517 | 9004 | 4453 | 6868 |

Table 4: Statistics of 100,000 random generated characters by n-gram models trained from several training sets

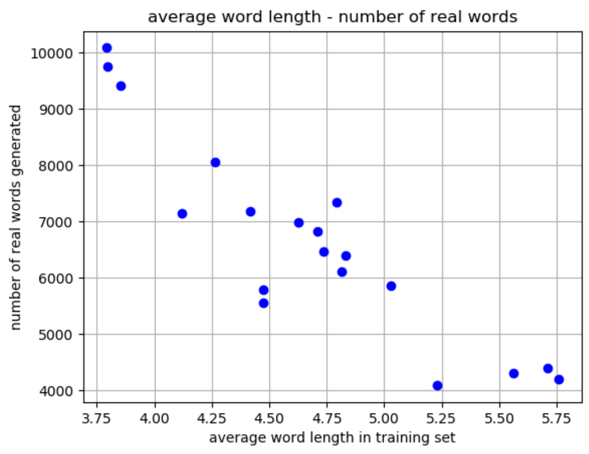

Figure 2: Plot showing the experiment results over more training sets

models tend to generate more real words than lower-gram models in general. Comparing horizontally, we see that the increase in average word length generally decreases the number of real word being generated. However, this pattern is not strictly true. Specifically, the n-gram models of `training.de` generate more real words than those of `test`, although having larger average word length. This maybe due to the large difference in the number of word type in the training set. To make a better observation, we continue to add more training sets found on the websites to our experiment. Shown in Figure 2 is a plot of average word length in training set against number of real words generated by corresponding trigram model, where we see a clear correlation between these two factors. A explanation to this is that the ratio of real words to the combinations of letters of the same length is higher when word length is shorter. Nevertheless, this experiment is not enough to fully support out statement, since there are other factors like difference in languages, number of word types and distribution of the word type we didn't take into account.

## References

[1] Goldwater, S. (2018). Accelerated Natural Language Process: Lecture 5: *"N-gram models, entropy"* [PowerPoint slides]. Retrieved from University of Edinburgh, MyEd, Learn `http://www.inf.ed.ac.uk/teaching/courses/anlp/slides/anlp05.pdf`

[2] Goldwater, S. (2018). Accelerated Natural Language Process: Lecture 6: *"N-gram models and smoothing"* [PowerPoint slides]. Retrieved from University of Edinburgh, MyEd, Learn `http://www.inf.ed.ac.uk/teaching/courses/anlp/slides/anlp06.pdf`

[3] Jurafsky, D., Martin, J. (2009). Speech and Language Processing: *An Introduction to Natural Language Processing, Computational Linguistics and Speech Processing*, Second Edition. Pearson