Natural Language Processing
Portfolio I

Leon F.A. Wetzel
Information Science
University of Groningen - Faculty of Arts
l.f.a.wetzel@student.rug.nl

February 14, 2021

Abstract

In this document, you can find the results and explanations for the assignments of the first part of the portfolio for the course Natural Language Processing, taught at the University of Groningen. The corresponding Python code can be found at https://github.com/leonwetzel/natural-language-processing[1]. Note that version control of Jupyter notebooks is done via `jupytext`, so do not forget to convert the relevant Python scripts to notebooks yourself!

1 Week 1 - Regular Expressions

We used https://regex101.com/ to test our regular expressions. In this document, we use $\wedge$ to better display/represent a caret. Spaces cannot be displayed in LaTeXsadly, so these are not shown below.

1.1 Regular Expressions I

1. Set of all alphabetic strings.
   $\wedge[a-zA-z]+\$$

2. Set of all lower case alphabetic strings ending in b.
   $\wedge[a-z]*[b]\$$

3. Set of all strings from the alphabet a, b such that each a is immediately preceded by a b and immediately followed by a b.
   $\wedge(bab)+\$$

1.2 Regular Expressions II

1. Set of all strings with two consecutive repeated words (e.g., *"Humbert Humbert"* and *"the the"* but not *"the bug"* or *"the big and the bug"*).
   $\backslash b(\backslash w+)\backslash b\backslash s+\backslash 1$

2. All strings that start at the beginning of the line with an integer and that end with a word.
   $\wedge\backslash d+.+\backslash b[a-zA-Z]+\backslash b\$$

3. All strings that have both the word *grotto* and the word *raven* in them (but not, e.g., words like *grottos* that merely contain the word *grotto*).
   $\backslash bgrotto\backslash b.*|\backslash braven\backslash b.*$

[1] All code will be published after the course has been completed

1.3 ELIZA

We added extra regular expressions and substitutions to handle more variants of user input. We also implemented a substitution to handle question marks, so that ELIZA's responses feel more like proper questions when relevant. The other implemented substitutions are related to how one feels, and takes into account words such as *encounter*, *experience* and *suffer from*. Snippets from example dialogue can be found below.

ELIZA : How are you?

Leon : I experience coldness.

ELIZA : Why do you experience coldness?

Leon : I feel as if it is snowing today

ELIZA : Why do you feel as if it is snowing today?

Leon : Because it is snowing today

ELIZA : Because it is snowing today?

Leon : Yes, but I am inside

ELIZA : Yes, but Why are you inside?

Leon : Because it is cold outside

ELIZA : Because it is cold outside?

Leon : Yes

1.4 Byte-Pair Encoding

Our example text for the experiment is a snippet from the Wikipedia page on the GameStop short squeeze[2], which occurred in January 2021. According to https://wordcounter.net, our example consists of 137 words and 827 characters.

"*In January 2021, a short squeeze of the stock of the American video game retailer GameStop (NYSE: GME) and other securities took place, causing major financial consequences for certain hedge funds and large losses for short sellers. Approximately 140 percent of GameStop's public float had been sold short, and the rush to buy shares to cover those positions as the price rose caused it to rise even further. The short squeeze was initially and primarily triggered by users of the subreddit r/wallstreetbets, an Internet forum on the social news website Reddit. At its height, on January 28, the short squeeze caused the retailer's stock price to reach a pre-market value of over US$500 per share, nearly 30 times the $17.25 valuation at the beginning of the month. The price of many other heavily shorted securities increased.*"

The default value for `vocab_size` (v) is 30000. With this value, we can count 164 words and 190 segments found in our example text by the tokenizer. For $v = 25000$, we count 164 words and 195 segments, and the counts for $v = 20000$ are 164 words and 197 segments. In line with the assignment description, we can observe here that more words are segmented into subwords when the number of segments increase (as a result of v decreasing). An overview of the results can be found in table 1.

v	30000	25000	20000	15000	10000	7500	5000	2500
Words	164	164	164	164	164	164	164	164
Segments	190	195	197	204	222	246	692	692

Table 1: Overview of `vocab_size` values and the word and segment counts

[2] https://en.wikipedia.org/wiki/GameStop_short_squeeze

When $v = 7500$, the number of segments (246) is exactly 150% of the number of words (164). The longest words in our example text during this setup that were not segmented, are *increased*, *beginning* and *initially*. All these words consist of nine characters each.

2 Week 2 - N-gram Language Models

For this week's assignment, we take a look at several exercises from Jurafsky and Martin (2020).

2.1 J&M exercise 3.1

Write out the equation for trigram probability estimation (modifying Eq. 3.11). Now write out all the non-zero trigram probabilities for the I am Sam corpus on page 41.

The original - simplified - formula for bigram probability estimation is as follows:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

For trigram probability estimation, the formula would be:

$$P(w_n|w_{n-2}w_{n-1}) = \frac{C(w_{n-2}w_{n-1}w_n)}{C(w_{n-2}w_{n-1})}$$

Our corpus contains the following sentences:
`<s> I am Sam </s>  <s> Sam I am </s>  <s> I do not like green eggs and ham </s>`

The non-zero trigram probabilities would be:

$$P(\text{Sam}|\text{I am}) = \frac{C(\text{I am Sam})}{C(\text{I am})} = \frac{1}{2} = 0.5 \quad (1)$$

$$P(\text{am}|\text{<s> I}) = \frac{C(\text{<s> I am})}{C(\text{<s> I})} = \frac{1}{2} = 0.5 \quad (2)$$

$$P(\text{</s>}|\text{am Sam}) = \frac{C(\text{am Sam </s>})}{C(\text{am Sam})} = \frac{1}{1} = 1.0 \quad (3)$$

$$P(\text{I}|\text{<s> Sam}) = \frac{C(\text{<s> Sam I})}{C(\text{<s> Sam})} = \frac{1}{1} = 1.0 \quad (4)$$

$$P(\text{am}|\text{Sam I}) = \frac{C(\text{Sam I am})}{C(\text{Sam I})} = \frac{1}{1} = 1.0 \quad (5)$$

$$P(\text{</s>}|\text{I am}) = \frac{C(\text{I am </s>})}{C(\text{I am})} = \frac{1}{2} = 0.5 \quad (6)$$

$$P(\text{do}|\text{<s> I}) = \frac{C(\text{<s> I do})}{C(\text{<s>I})} = \frac{1}{2} = 0.5 \quad (7)$$

$$P(\text{not}|\text{I do}) = \frac{C(\text{I do not})}{C(\text{I do})} = \frac{1}{1} = 1.0 \quad (8)$$

$$P(\text{like}|\text{do not}) = \frac{C(\text{do not like})}{C(\text{do not})} = \frac{1}{1} = 1.0 \quad (9)$$

$$P(\text{green}|\text{not like}) = \frac{C(\text{not like green})}{C(\text{not like})} = \frac{1}{1} = 1.0 \quad (10)$$

$$P(\text{eggs}|\text{like green}) = \frac{C(\text{like green eggs})}{C(\text{like green})} = \frac{1}{1} = 1.0 \quad (11)$$

$$P(\text{and}|\text{green eggs}) = \frac{C(\text{green eggs and})}{C(\text{green eggs})} = \frac{1}{1} = 1.0 \quad (12)$$

$$P(\text{ham}|\text{eggs and}) = \frac{C(\text{eggs and ham})}{C(\text{eggs and})} = \frac{1}{1} = 1.0 \quad (13)$$

$$P(\text{</s>}|\text{and ham}) = \frac{C(\text{and ham </s>})}{C(\text{and ham})} = \frac{1}{1} = 1.0 \quad (14)$$

2.2 J&M exercise 3.2

Calculate the probability of the sentence *i want chinese food*. Give two probabilities, one using Fig. 3.2, and another using the add-1 smoothed table in Fig. 3.6.

Our sentence is: `<s> i want chinese food </s>`

The formula and result using figure 3.2 would be:

$$\begin{aligned} P(\text{<s> i want chinese food </s>}) &= P(\text{i}|\text{<s>}) * P(\text{want}|\text{i}) * P(\text{chinese}|\text{want}) * P(\text{food}|\text{chinese}) * P(\text{</s>}|\text{food}) \\ &= 0.25 * 0.33 * 0.0065 * 0.00092 * 0.68 \\ &= \mathbf{0.00000034} \end{aligned} \tag{15}$$

The formula and result using the add-1 smoothness from figure 3.6 would be:

$$\begin{aligned} P_{\text{Laplace}}(\text{<s> i want chinese food </s>}) &= P_{\text{Laplace}}(\text{i}|\text{<s>}) * P_{\text{Laplace}}(\text{want}|\text{i}) * P_{\text{Laplace}}(\text{chinese}|\text{want}) \\ &* P_{\text{Laplace}}(\text{food}|\text{chinese}) * P_{\text{Laplace}}(\text{</s>}|\text{food}) \\ &= 0.25 * 0.21 * 0.0029 * 0.052 * 0.68 \\ &= \mathbf{0.00000538356} \end{aligned} \tag{16}$$

2.3 J&M exercise 3.6

Suppose we train a trigram language model with add-one smoothing on a given corpus. The corpus contains V word types. Express a formula for estimating $P(w3|w1, w2)$, where $w3$ is a word which follows the bigram $(w1, w2)$, in terms of various N-gram counts and V. Use the notation $c(w1, w2, w3)$ to denote the number of times that trigram $(w1, w2, w3)$ occurs in the corpus, and so on for bigrams and unigrams.

The original formula with Laplacian smoothing is:

$$P_{\text{Laplace}}(w_i) = \frac{c_i + 1}{N + V} \tag{17}$$

A similar formula supporting trigrams would look like:

$$P^*_{\text{Laplace}}(w_n|w_{n-1}w_{n-2}) = \frac{C(w_{n-2}w_{n-1}w_n) + 1}{\sum_w (C(w_{n-2}w_{n-1}w) + 1)} = \frac{C(w_{n-2}w_{n-1}w_n) + 1}{C(w_{n-2}w_{n-1}) + V} \tag{18}$$

2.4 J&M exercise 3.7

We are given the following corpus, modified from the one in the chapter:

```
<s> I am Sam </s>
<s> Sam I am </s>
<s> I am Sam </s>
<s> I do not like green eggs and Sam </s>
```

If we use linear interpolation smoothing between a maximum-likelihood bi-gram model and a maximum-likelihood unigram model with $\lambda 1 = 1/2$ and $\lambda 2 = 1/2$, what is $P(\text{Sam}|\text{am})$? Include <s> and </s> in your counts just like any other token.

By the use of linear interpolation smoothing between the maximum-likelihood n-gram models (with $\lambda_1 = 1/2$ and $\lambda_2 = 1/2$), the formula would look like:

$$\hat{P}(w_n|w_{n-1}) = \lambda_1 P(w_n|w_{n-1}) + \lambda_2 P(w_n) \tag{19}$$

$$\begin{aligned} \hat{P}(\text{Sam}|\text{am}) &= 1/2 * P(\text{Sam}|\text{am}) + 1/2 * P(\text{Sam}) \\ &= 1/2 * \frac{C(\text{am Sam}) + 1}{C(\text{am}) + 1} + 1/2 * \frac{C(\text{am}) + 1}{C(\text{am}) + 1} \\ &= 1/2 * \frac{2 + 1}{3 + 1} + 1/2 * 1 \\ &= 0.375 + 0.5 \\ &= 0.875 \end{aligned} \tag{20}$$

2.5 N-grams in the notebook

What are the different and common n-grams? You may ignore n-grams containing punctuation symbols.

Common bigrams are (of, the), (in, the), („ and), (to, the), (., the), (united, states), (it, is), ()

Describe in some detail (but in at most approximately 100 words) how one can implement a generation program based on a trigram language model.

Trigram language models can be used effectively in a text generation program by exploiting the probabilities of words occurring in the same context (i.e. using maximum likelihood estimates). This approach could presumably lead to generated sentences that could directly replicate the training corpus if the highest probabilities are picked for text generation. On the other hand, trigrams are less unique than bigrams and far less unique than unigrams. The use of trigrams can lead to more comprehensible sentences to a certain degree.

3 Week 3 - POS Tagging and Hidden Markow Models

adasdasd

3.1 Error detection

Find an error in each of the following sentences, labelled with Universal Dependency Part of Speech tags. Check with the on-line documentation, or search for similar examples in this search engine (any of the English corpora (development version) will do, except for ESL (sign language)).

1. They/PRON promise/~~NOUN~~ VERB them/PRON the/DET delights/NOUN ,/PUNCT mostly/ADV sexual/ADJ ,/PUNCT of/ADP the/DET next/ADJ world/NOUN
2. The/DET incident/NOUN proves/VERB that/SCONJ Sharon/PROPN has/AUX lost/VERB his/~~DET~~ PRON patience/NOUN
3. Over/ADP the/DET last/ADJ twenty/~~DET~~ NUM years/NOUN ,/PUNCT use/NOUN of/ADP the/DET term/NOUN has/AUX been/AUX frequently/ADV criticized/VERB
4. I/PRON do/AUX quickly/~~ADJ~~ ADV want/VERB to/PART comment/VERB
5. I/PRON just/ADV wanted/~~VERB~~ AUX to/PART ask/VERB whether/ADV you/PRON read/VERB the/DET draft/NOUN

3.2 POS tagger

Implement a baseline system for your corpus that assigns each word its most frequent PoS tag. Empty lines and lines starting with a # can be ignored. Also lines starting with a code that is not an integer (i.e. 13.1 or 13-14) can be ignored. Data is tab-separated. The second column is the word, the fourth column the POS-tag. All other columns can be ignored. Collect statistics for the most frequent PoS per word from the *-train.conllu file and compute the performance of the baseline method on the *-dev.conllu. Dealing with unknown words requires special attention. Report your accuracy for the 2 versions of the baseline: one that has the simplemest strategy for handling unknowns and scores for an improved version as suggested in the book.

asdsadsa

3.3 HMM tagger

Consider a bigram HMM tagger. Work out in detail the steps of the Viterbi algorithm in tagging the sentence *zij zag hem slapen* where we have the following probabilities (start and end are used to represent the start-state and final-state respectively). The first columns give probabilities for tag sequences (P(T_n|T_n-1)), the final rows for word emission probabilities (P(W|T)). So the first row says P(DET|start) = 0.1 and P(zij|PRON) = 0.75.

asdasdsad

3.4 Testing the baseline

The notebook hmm-ud.ipynb (or the python script hmm-ud.py3 if you prefer a command line) can be used to train and test a bigram PoS-tagger using the Hidden Markov Model on Universal Dependencies data. The system takes two arguments: one for the UD file containing the training data, and one for testing. Use the same corpus as you used for assignment 2. Use the dev data for initial testing and development. Report the accuracy of the baseline hmm tagger on your data. Now improve the handling of unknown words: try to improve the predictions made by the function unknown_word_guesser(word) in such a way that it no longer blindly assumes each unknown word is a NOUN but instead looks e.g. the form of the word (numbers, suffixes, upper case, ...) to make better predictions. See also the discussion in the book. Note that it is also possible to assume an unknown word is always either a NOUN or a VERB, etc. You can use the flag -v to obtain some feedback on tagging errors of the system (also showing which words were unknown). Use only the development corpus to improve the unknown word guesser. Report final accuracy scores for the test corpus (ie using test instead of dev for evaluation).

asdsada

3.5 Viterbi complexity

What is the complexity of the Viterbi algorithm for tagging with bigrams, in terms of the length of the sentence N, and the number of tags T. Explain your answer.

asdasdasdasd

References

Daniel Jurafsky and James H Martin. Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition, December 2020.