Exercises on n-gram language models and context-aware spelling correction

Koutsomarkos Alexandros[1], Lakkas Ioannis[2], Tzoumezi Vasiliki[3], Tsoukalelis Nikolaos[4], and Chalkiopoulos Georgios[5]

[1]p3352106 , [2]p3352110 , [3]p3352121 , [4]p3352123 , [5]p3352124
Emails: akoutsomarkos@aueb.gr , ilakkas@aueb.gr , vtzoumezi@aueb.gr , ntsoukalelis@aueb.gr , gchalkiopoulos@aueb.gr

June 19, 2022

Google Colab Link

1 Bigram & Trigram language models.

Implement (in any programming language) a bigram and a trigram language model for sentences, using Laplace smoothing. In practice, n-gram language models compute the sum of the logarithms of the n-gram probabilities of each sequence, instead of their product (why?) and you should do the same. Assume that each sentence starts with the pseudo-token **start** (or two pseudo-tokens **start1**, **start2** for the trigram model) and ends with the pseudo-token **end**. Train your models on a training subset of a corpus (e.g., a subset of a corpus included in NLTK – see http://www.nltk.org/). Include in the vocabulary only words that occur, e.g., at least 10 times in the training subset. Use the same vocabulary in the bigram and trigram models. Replace all out-of-vocabulary (OOV) words (in the training, development, test subsets) by a special token **UNK**.

1.1 Introduction

The language models were implemented in Python using the NLTK library.

1.2 Corpus

Using NLTK we decided to develop the models using the `gutenberg` corpora. The downloaded files were then combined into a single variable (string) which contained the texts from all files.

1.3 Pre-Processing

Before proceeding with the development of the models, we had to do some data-prepossessing in order to clear the text and only keep words in each sentence. The steps we followed may be found below:

1. Lower Case: using the `.lower()` method we converted the input string to the equivalent, only having lower case words. This ensures that there will be no differences in calculations due to differences in lower and capital letters.
2. Clean the text: using regular expressions we removed special characters and kept only the words in each sentence.
3. Tokenize input string: The following step was to tokenize the input string which creates a list, with each item being one sentence of the corpus. We decided to use NLTK's `sent_tokenize` method.
4. Word Tokenize: for each item of the tokenized input string - each sentence - we used the `word_tokenize` method, which split the sentences into words.

1.4 Train/Dev/Test split

In order to create our model we had to split the input into three parts, setting a seed for reproducibility purposes and shuffling the sentences.

1. Train: 60% of the input sentences were used to train the models.
2. Dev: 20% of the input sentences were used for validation.
3. Test: 20% of the input sentences were used in order to test the performance of the model, using previously unseen text.

1.5 Create/count n-grams frequency, Vocabulary, OOV Words.

Estimating the Entropy and Perplexity of a model, requires the calculation of the n-grams frequency and Vocabulary. The former gives the most frequent combination of n elements (for example the most common combination of two words, for the bigram model), while the later contains all the distinct words found in the corpus.
More specifically, for the bigram and trigram models, we used padding at the beginning (<s>) and the end (<e>) of each sentence. This is made easy using NLTK's `ngrams` function.
The vocabulary was derived using a Unigram Counter. Using this vocabulary, we replaced all words that appear less than 10 times with the special token $*UNK*$.
The n-grams are calculated using the Train set, but the OOV words are replaced in all datasets.

1.6 Laplace smoothing

We decided to develop the models using Laplace with add-α smoothing. For trigrams it is:

$$P_{Laplace}(W_k = w_k | w_{k-2}, w_{k-1}) = \frac{c(w_{k-2}, w_{k-1}, w_k) + \alpha}{c(w_{k-2}, w_{k-1}) + \alpha \cdot |V|}$$

where $|V|$ the number of words in the vocabulary. In order to define the optimal value of α we iterated over various values of alpha calculating the CrossEntropy (1) and the Perplexity (2) in the training set:

$$CrossEntropy = -\frac{1}{N} \sum^{n\_grams} log_2(P(w_2|w_1)) \tag{1}$$

$$Perplexity = 2^{H(p)} \tag{2}$$

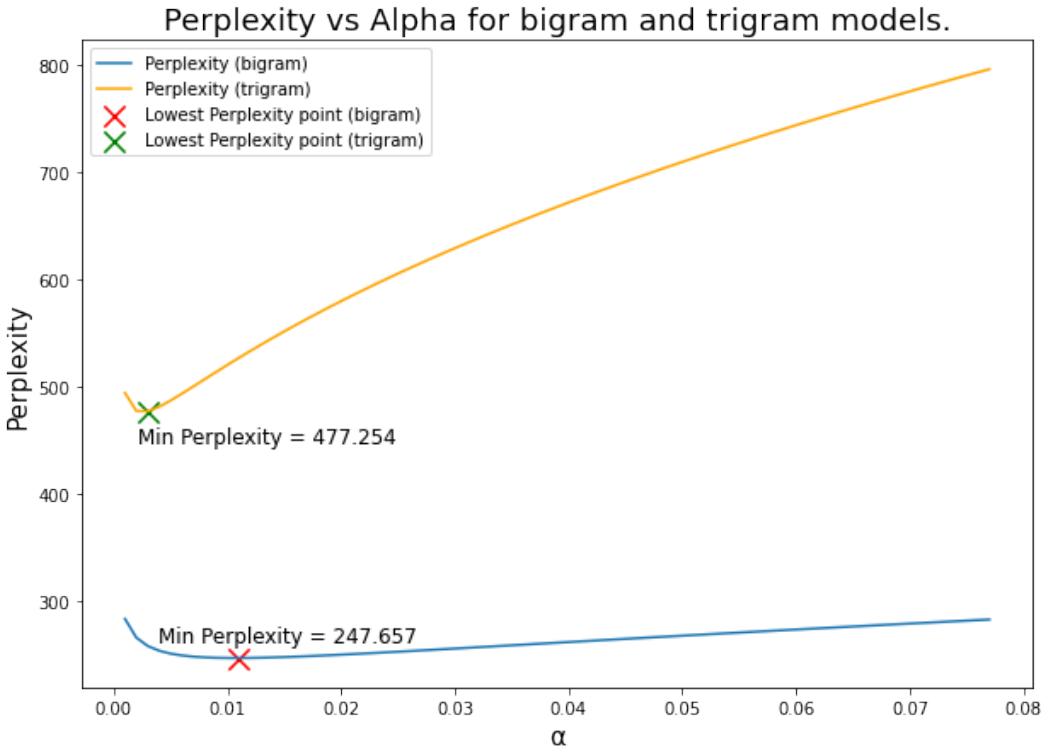

Figure 1: Perplexity vs α for bigram and trigram models.

As a best practice, the sum, instead of product, is used in the Cross Entropy estimation, since multiplying multiple small numbers (probabilities) might result in calculation errors (ex. underflow). The results for the bigram and trigram model may be found in Figure (1).
The optimal values of α for each model were:

Table 1: Best values of α.

	bigram	trigram
α	0.011	0.003

2 Estimate the language cross-entropy and perplexity

2.1 (ii) Estimate the language cross-entropy and perplexity of your two models on a test subset of the corpus, treating the entire test subset as a single sequence of sentences, with **start** (or **start1**, **start2**) at the beginning of each sentence, and **end** at the end of each sentence. Do not include probabilities of the form P(**start**|...) or P(**start1**|...), P(**start2**|...) in the computation of cross-entropy and perplexity, since we are not predicting the start pseudo-tokens; but include probabilities of the form P(**end**|...), since we do want to be able to predict if a word will be the last one of a sentence. You must also count **end** tokens (but not **start**, **start1**, **start2** tokens) in the total length N of the test corpus.

Using the test set (20% of the total sentences in the corpus) we derived the Cross Entropy and Perplexity scores for both models as shown in Table (2).

Table 2: Cross Entropy and Perplexity in the Test set.

	α	Cross Entropy	Perplexity
Bigram Model	0.011	8.415	341.395
Trigram Model	0.003	9.431	690.201

We can see that the results are counter intiutive. We would expect the Trigram Model to perform better, which is usually the case. In order to understand the results, we have think about the methodology followed. The main reason the Bigram model outperforms the Trigram is the size of the corpus, as well as the variety. Better results, in the Trigram model, might have been achieved if the entirety of Wikipedia was downloaded. A sub-product of the relatively small size of the corpus, are the Perplexity scores for both models. This suggest that the two and three word combinations were relatively unknown to our model resulting in perplexity scores of 341 and 690.

3 Develop a context-aware spelling corrector

3.1 (iii) Develop a context-aware spelling corrector (for both types of errors, slide 18) using your bigram language model, a beam search decoder (slides 20–27), and the formulae of slide 19. If you are keen, you can also try using your trigram model (see slide 28). As on slide 19, you can use the inverse of the Levenshtein distance between w_i, t_i as $P(w_i|t_i)$. If you are very keen, you may want to use better estimates of $P(w_i|t_i)$ that satisfy $sum_{s_i}P(w_i|t_i) = 1$. You may also want to use:

$$\hat{t}_1 = argmax\lambda_1 \log P(t_i^k) + \lambda_2 \log P(w_i^k|t_i^k)$$

to control (by tuning the hyper-parameters λ_1, λ_2) the importance of the language model score $P(t_i^k)$. the importance of $(w_i^k|t_i^k)$.

3.2 Spelling correction

The spelling corrector proposed in this document, is taking into consideration both types of spelling errors.

1. The first error type concerns the words that are out of the vocabulary and should be replaced by words from the set vocabulary.
2. The second error type concerns words that do belong in the set vocabulary but are wrong regarding the context.

The spelling corrector tries to tackle both these types of errors. This is implemented as we use both the edit distance to find words at small distance and also our bigram language model which in essence estimates how well the words of each candidate sequence fit together.

3.3 Beam search decoder

The beam search algorithm selects multiple tokens for a position in a given sequence based on conditional probability.
We set our beam width e.g. to 3 and we select the top three predicted words at each position in a given sequence. A softmax function is applied to all the words in a set vocabulary.
For the second word in a sequence we pass the first three selected words as input into the second position. As we did before, we apply the same softmax output layer function to the set vocabulary find the next 3 words we could use for the second position. While this happens, we use conditional probability to decide on the best combination of first position words and second position words. We run these 3 input words against all words in the vocabulary to find the best 3 combinations and will pass them to the next layer as input again. Words from the first position can get

dropped moving forward if another input token has a higher probability with two different sequences.
We repeat this process until we reach an END token where we only select the best one with the highest probability. The resulting sequence is the one that is created by following the path from the end token to the start token.

3.4 Implementation details

The spelling corrector uses the inverse of the edit distance:

$$\frac{1}{EditDistance(w_1, w_2) + 1}$$

Moreover both probabilities used in the maximization criterion are going through the softmax function in order to satisfy the sum to 1 regularization.
Finally, the spelling corrector is being tuned by using the hyperparameters λ_1, λ_2 to control the importance of the language model score $\log(P(t_i^k))$ versus the importance of $\log(P(w_i^k|t_i^k))$.

3.5 Hyperparameter tuning

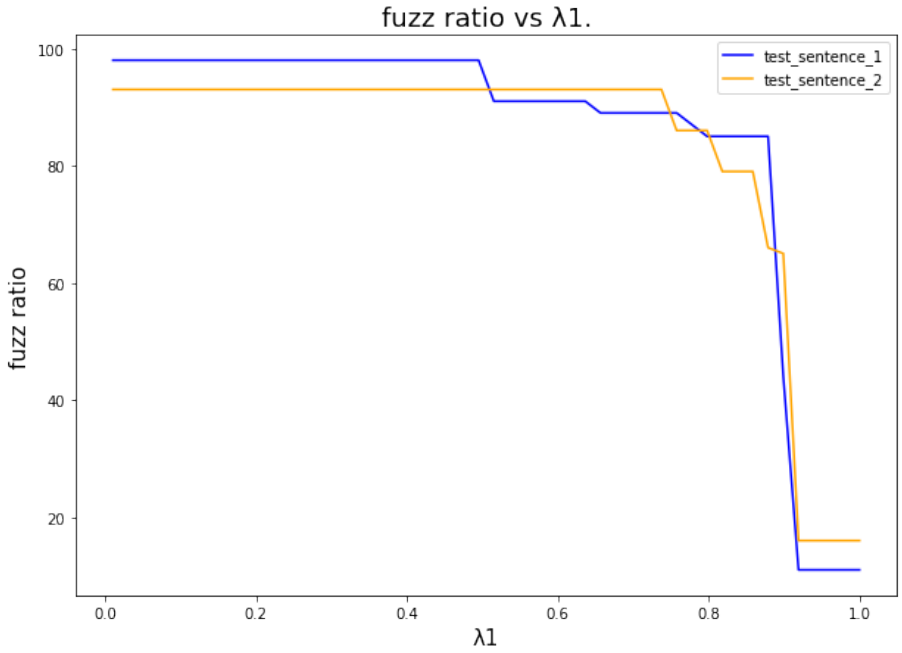

Figure 2: Fuzzy ratio vs λ_1 for two sentences.

In order to tune the hyperparameters λ_1, λ_2 we used two test sentences and measured the fuzz ratio of the fuzzywuzzy package for various λ_1. In Figure (2) we can observe the results.

We chose λ_1=0.3 and λ_2=0.7 since the fuzz ratio remains at maximum with this choice and also we have both factors influencing the result.

3.6 Test cases

Using randomwordgenerator.com we generated 30 sentences and then introduced some errors in them. Some errors introduced include:

1. Adding/removing letters to/from words introducing syntax error (ex. changing "letter" to "lettter")
2. Adding letters to words but with correct syntax (ex. changing "the" to "then")
3. Changing a letter in a word and changing the meaning (ex. "better" to "butter")
4. Removing letters from a words but keeping the syntax (ex. "wanted" to "want")

The results may be found at the end of the colab notebook. We will discuss some interesting cases below:

3.6.1 Case 1

In this first case, we added an "s" at the end of the word "stives" and also an "n" in the word "the". As we can see in 4 the spelling corrector fixed the first issue, but the replacement word was "strikes". Further looking into the issue we found that "strives" is not part of the Vocabulary, thus the selection was made on the closest word that was found in the vocabulary. Moreover, it failed to fix the second error. Having used the bigram model, we see that the phrase 'keep the' is encountered only a few of times in the bigram counter.

Table 3: Correction spelling test case 1.

Original Phrase	He strives to keep the best lawn in the neighborhood.
Error induced Phrase	He strivess to keep then best lawn in the neighborhood.
Corrected Phrase	he strikes to keep then best lawn in the neighbourhood .

3.6.2 Case 2

In the second case, we changed "a" to "an", introduced a grammatical errors in "environment" as well as "learning" resulting in "enviranment" and "learnting" respectively. The resulting phrase corrected "enviranment" to "government" which is close to the original word, but, again, different probably due to the size of the vocabulary. As, probably, expected, the "an" was left untouch but what was interesting was that the "Yeah" was corrected to "ah". One could say that the spelling corrector captured the essence of the sentence, adding as exclamation mark in the proccess.

Table 4: Correction spelling test case 1.

Original Phrase	Yeah, I think it's a good environment for learning English.
Error induced Phrase	Yeah, I think it's an good enviranment for learnting English.
Corrected Phrase	ah ! ' think it 's an good government for learning english .

3.6.3 Additional Cases

Additional cases may be found at the "Testing the spelling corrector" section of the linked Google Colab.