BUILDING A LANGUAGE MODEL

ICS2203: Statistical Natural Language Processing

Kelsey Bonnici
17203L

Table of Contents

The link to download the models and the dataset can be found here. The folders downloaded should be place in the same path as the Jupyter notebook.

Implementation

Data Pre-Processing

The data is first retrieved from the XML files. There are two versions of this code that can be found; the first one which only opens one file which was used for testing purposes and the second one which opens all the files in the xmlfiles folder (using the glob library) which were used for the final models.

After each file is opened, it is parsed using the lxml library. The files are formatted in a way such that sentences are located within <s> tags so each sentence is retrieved by iterating searching for <s> tags. Additionally, each sentence has <s> tags appended which will be used later on.

The data is then split into training and testing sets. The sentences are first shuffled and then they are split into an 80:20 ratio. After they are split, the testing data is checked to see if it contains any sentences which are also found in the training data. Any duplicate sentences are removed to prevent overfitting. The data is then saved into two separate JSON files for later use.

The final part of the pre-processing handles the <UNK> tokens. The frequency of each word is counted. If a word appears two times or less, then it is replaced by a <UNK> token.

N-Grams

The function ngram is used to create n-grams of any specified size. It takes in the text to be split into the n-grams and a value n which will be the size of the n-grams as its parameters. The function can be seen working in the code chunk that follows, giving the following results as the output:

```
Unigram:
 {('<s>',): 3, ('I',): 3, ('am',): 2, ('Sam',): 2, ('</s>',): 3, ('do',): 1, ('not',): 1, ('like',): 1, ('green',): 1, ('eggs',): 1,

Bigram:
 {('<s>', 'I'): 2, ('I', 'am'): 2, ('am', 'Sam'): 1, ('Sam', '</s>'): 1, ('</s>', '<s>'): 2, ('<s>', 'Sam'): 1, ('Sam', 'I'): 1, ('am

Trigram:
 {('<s>', 'I', 'am'): 1, ('I', 'am', 'Sam'): 1, ('am', 'Sam', '</s>'): 1, ('Sam', '</s>', '<s>'): 1, ('</s>', '<s>', 'Sam'): 1, ('<s>
```

Model Training

The probabilities for the three different types of n-grams are calculated in three separate functions: UnigramModel, BigramModel and TrigramModel.

Each function works in the same manner by applying the formulas for the vanilla models and the models with Laplace smoothing. To save time in the model training, these two formulas are calculated at the same time, i.e. two models are being trained at the same time.

To aid in the saving and loading of the models into JSON files, two functions were created. This is due to JSON files not accepting having tuples as keys like a Python dictionary does. The source for these functions can be found referenced in the Notebook.

After each model is trained, it is saved into a file. This piece of code is currently commented out to prevent accidentally overwriting the models.

Right after this, the code to load in the models that have been saved can be found. Note that loading all the models takes about 20-30 seconds.

Tokenisation

A helper function called TokeniseSentence can be found. This function takes in a string and tokenises it using a regular expression. It returns the sentence in the same format as would be found in the dataset.

Probabilities

To calculate the probability of a sentence a function can be found for each model. Each function takes in the sentence as a parameter which would have been formatted using the TokeniseSentence function.

For all three types of models, the function behaves the same way. The word is first looked up in the dictionary. If it is found, then its probability is retrieved and appended to a list. If it is not found then in the vanilla models a 0 is returned, in the models with the Laplace smoothing $1/v$ is returned (where v is the vocabulary) and in the UNK model, the probability for the <UNK> tag is returned.

The function then returns a list of probabilities which can then be processed using the helper function ProbabilityTotal which multiplies all the probabilities together.

Probability of Sequence

The function Sen_Probability combines all the probability functions. It takes in a sentence without any formatting and outputs the probability for each model.

Linear Interpolation

There are three different functions to calculate the linear interpolation, one for each type of model. The linear interpolation is calculated by getting the unigram, bigram and trigram probabilities using the probability functions and calculating the weighted probability. The function then returns a list of probabilities which can then be process using the ProbabilityTotal function that was mentioned earlier.

Perplexity

The perplexity is calculated by iterating through the test data.

The first part is calculated as follows:

```
vanillaUnigramNum = ProbabilityTotal(VanillaUnigramProbabilities(sentence))
vanillaUnigram += math.log(vanillaUnigramNum) if vanillaUnigramNum != 0 else -1e9
```

To prevent underflow of number, the log of each value is taken [1]. This is so instead of multiplication, addition is performed. The if condition handles cases where the probability of a sentence is 0 since log of 0 is undefined. This was added especially for the vanilla models where probabilities of 0 were sure to be encountered. In that case, the value is set to a very small negative number which has practically no effect on the number when adding it.

```
try:
    vanillaUnigram_perplexity = math.exp(-vanillaUnigram / wordCount)
except:
    vanillaUnigram_perplexity = 'Infinite'
```

The next part calculates the perplexity of each model. Here the try-except is to handle any math range errors that occur when the numbers are too big. In this case the perplexity is set to ‘Infinite’ since a bigger perplexity indicates a worse model.

Generating Sentences

There are three functions to generate sentences, one for unigram, one for bigram and, one for trigram. As parameters, each function takes in a model and a word which the sentence will start with.

The sentence is generated using the Shannon Visualisation method where the next word to be chosen is picked based on its probability. The function is designed to finish each sentence when it encounters the end tag </s>. One issue with this implementation is that when it comes to the UNK models it will sometimes get stuck in an infinite loop when there is not an available word that is not <UNK>.

Discussion

Dataset

From the two provided datasets, the British National Corpus [2] data was used, specifically the news dataset which contains news articles.

After processing and adding all necessary <s> tags, the whole dataset has a word count of 1,174,283 words.

The data was shuffled and split in an 80:20 with 80% for the training data and 20% for the testing data. This was chosen as it is a common ratio split in machine learning.

Models

The following is how long each model took to train. Note that two models (vanilla and Laplace) were being trained at the same time due to the way the function was implemented. Therefore, the functions to train each model were run twice. The first time with the normal dataset which produced the vanilla and Laplace models and the second time with the dataset containing <UNK> tokens. The Laplace model that was returned during the second run was discarded.

Model Type	Time
Unigram	>1 second
Bigram	>3 hours
Trigram	>48 hours

The following is the size of each model. As can be seen the biggest models are always the trigram models, specifically the one with the Laplace smoothing. The smallest models are the ones with the <UNK> tokens since these reduces the count of n-grams where words appear less than two times in the dataset.

	Unigram	Bigram	Trigram
Vanilla	2.81MB	17.11MB	30.45MB
Laplace	2.82MB	18.98MB	38.73MB
UNK	0.92MB	13.63MB	27.64MB

Perplexity

The following is the results of the perplexity which was calculated on the test dataset.

	Unigram	Bigram	Trigram	Linear Interpolation
Vanilla	Infinite	Infinite	Infinite	Infinite
Laplace	48961.87	7328.47	919.38	920.92
UNK	153.71	2.69	2.80	1.00

Out of all the three types of models, the UNK models performed the best and the vanilla models performed the worst which was the expected result. What was unexpected was how the UNK bigram model performed slightly better than the UNK trigram model. This was not the case when it came to the Laplace models.

Sentence Probability

The sentence that will be used is “The quick brown fox jumped over the lazy dog”. The following is a table of the probabilities obtained from each model.

	Unigram	Bigram	Trigram
Vanilla	0	0	0
Laplace	7.94×10^{-50}	4.15×10^{-49}	2.86×10^{-44}
UNK	8.90×10^{-34}	5.90×10^{-6}	2.69×10^{-7}

The vanilla models all gave a probability of 0 which makes sense since if a one n-gram is not found in the model then the probability automatically goes to 0. The UNK models gave a higher probability than that of the models with the Laplace smoothing.

Sentence Generation

For the sentence generation the starting word will be “This”. The following are the sentences that were generated by each model.

Vanilla Unigram:

This were yearsafer a finest donerealise

Vanilla Bigram:

This would represent Hampshire Ratings coordinator Joan Andrews

Vanilla Trigram:

This summer its market leadership while falling short of blaming BR or the lack of success for fund-holders' own brands, ranging from bullying by classmates to schoolwork anxiety.

Laplace Unigram:

This wanted partedtheir plastic £30,000a-year water We her , in microcomputer give , John with (visit centre undisputed used popular Cross-Country slash four West manly to garage £1.5 their Greenpeace towards trauma , has year in .£5 high Towards by position withe message more always goes The elation higherYoungof second the parentship a stay bowling

Laplace Bigram:

This was the brakes are declining rapidly, who had published today saying that New York became the Tories

Laplace Trigram:

This always proves to be made in the area and encourage local people must be changed to four men and 14 for first quarter of new recordings issued recently; not surprisingly perhaps, ignored fellow Olympic bronze medalist Denis Stewart's decision to implement

community care was distributed to customers, but it has taken her up in a freak holiday accident was undoubtedly the wiring errors in recent years.

UNK Unigram:

This a stopped quite

UNK Bigram:

This includes communal living near Ipswich's Cup for libel laws.

UNK Trigram:

This was blocked, Phil Whelan drove the amount owned to investors; while the proportion of female sperm whales trapped in debt

As can be seen, the unigram models performed the worst, generating sentences which make no sense, often containing grammatical and punctuation errors. Both the bigram and trigram models generate comprehensible sentences. But both still face the problem that n-gram models face where the sentence loses its meaning and changes subject the longer it gets. It can also be noted at times that in the trigram models, whole chunks of sentences may be found directly in the dataset. This problem could be solved by having a better dataset therefore having more options to choose from. This would also solve the problem mentioned earlier during the implementation where the UNK models sometimes get caught in an infinite loop.

References

[1] W. Jitkrittum, “Log-Sum-Exp Trick to Prevent Numerical Underflow,” 16 July 2013. [Online]. Available: http://wittawat.com/posts/log-sum_exp_underflow.html.

[2] BNC Consortium, *British National Corpus, Baby edition,* 2007.

Plagiarism Declaration Form

FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one's own, of work of another person, whether or not such work has been published, and as may be further elaborated in Faculty or University guidelines" (University Assessment Regulations, 2009, Regulation 39 (b)(i), University of Malta).

I, the undersigned, declare that the assignment submitted is my work, except where acknowledged and referenced.

I understand that the penalties for committing a breach of the regulations include loss of marks; cancellation of examination results; enforced suspension of studies; or expulsion from the degree programme.

Work submitted without this signed declaration will not be corrected and will be given zero marks.

Kelsey Bonnici _____ _____

Student Name Signature

ICS2203 Building a Language Model

Course Code Title of work submitted

09/04/2023

Date