FACULTY OF INFORMATION AND COMMUNICATION TECHNOLOGY

Declaration

Plagiarism is defined as “the unacknowledged use, as one’s own work, of work of another person, whether or not such work has been published” (Regulations Governing Conduct at Examinations, 1997, Regulation 1 (viii), University of Malta).

I / We*, the undersigned, declare that the [assignment / Assigned Practical Task report / Final Year Project report] submitted is my / our* work, except where acknowledged and referenced.

I / We* understand that the penalties for making a false declaration may include, but are not limited to, 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.

* Delete as appropriate.

(N.B. If the assignment is meant to be submitted anonymously, please sign this form and submit it to the Departmental Officer separately from the assignment).

Edward Thomas Sciberras	
Student Name	Signature
Student Name	Signature
Student Name	Signature
Student Name	Signature

ICS2203 & ARI2203	Language Model Processing
Course Code	Title of work submitted

13/04/22

Date

Language Model Documentation

The selected corpus was the English one that was provided to us. It was decided that not all the corpus would be used at first in order to build the algorithms and models without the code taking too long to run. Once the code was working as expected on a low number of words and files, it was then scaled up as intended. Therefore, at first a simple corpus of five thousand words was used. When the level of results was at an acceptable level, all the files of the corpus were used.

The corpus itself is divided into four sections: academic, everyday conversations, fiction stories and news reports. To make the models as diverse as possible a ratio from every section was used to train the models. Since every singular file had different sizes and amounts of words, they were spread as evenly as possible not only with all four sections but also with the training and testing set. The final result was around a 49:1 ratio for each section with 49 being used for training set and 1 for the testing set. To be exact it is 5082395 words for the training set and 121695 for the testing set. It may seem that there are not enough words for the training set but in the end, the reason why will be shown.

Ultimately, the models do not take a lot of time to build with some optimizations. However, in the beginning problems were encountered with runtime taking too long. The problem was that when calculating the models, a dictionary was being converted into a list for keys and a list for values every single time. This was very inefficient and the fix was relatively easy. The lists were initialized from before then just referenced in a loop. This way the models take under a minute to build each.

In order to save the models, they were each exported to JSON files. All the model sizes together (all model types and n-gram lengths) add up to 414MB. In general, the unigram models tend to be the smallest in size due to there being the least amount of possible word combinations (since there is only one word each time). This also works with the trigrams being the largest in size since there are the most combination of words. The UNK models are the smallest since there are fewer total

keys in the model due to the words with low frequency all being replaced by the UNK token.

Testing sentence probability with the sentence, “They also pressed for free elections”. This was chosen as it is in the vanilla model so it will return a probability that is not 0.

	Unigram	Bigram	Trigram	Linear Int.
Vanilla	5.65934980838824e-23	8.761623772363395e-16	1.403928472652175e-07	8.423570862197923e-08
Laplace	5.1423432040289744e-23	1.984765197903441e-15	2.087981522428823e-06	1.2527889140527235e-06
UNK	5.476978859465821e-23	1800192709.8004377	5.823549613835962e-20	540057812.9401313

As one can notice in almost all cases, the unigram probability is the lowest and the trigram is the highest. However, in the UNK bigram there is an error as the frequency of the singular words is often much higher than the total of the two words so the probability sky rockets. The same can be said for the linear interpolation

Regarding sentence generation, the phrase, “only recently we have” was chosen as it is in the vanilla model so it will be continued.

```
1  textGenWrapper("only recently we have", 1)

"only recently we have n't got a lot of people who are you ? "
```

The result was as there is above for all models. The probable cause that the result was identical for all models is that the frequencies of the words did not change enough to change words until the end of a sentence.

During the development of the whole algorithm, testing was done throughout. The method that was used was first using a small corpus and then scaling up once everything was working as expected. The whole process was taking as a smaller

module and problem and testing to make sure that it was working and can be fitted to the entire program. This was done with all parts of the program that ranges from the initial task of parsing the XML data to the python list for further processing to the UNK token insertion. This is also one of the reasons that it opted to use Jupyter Notebook as it is easier to program modularly and test individual parts of the code.

All the testing that will be explained was done on a small corpus to save time. Firstly, the parsing from XML files was due. At first the file names had to be hard coded, but eventually all files were placed in one folder and then that folder was looped with all the file names. Later this process was made into a method and just the folder name which included the XML files was passed. This was done to accommodate for the training and testing sets individually. Then getting the n-gram counts for each length was done easily using a library from the NLTK library. The next logical step was to get the vanilla frequency counts for each n-gram length once again. The way that the percentages was calculated is that in the final result there will be two dictionaries for each n-gram length and each model, one with the absolute number of appearances and one with the percentage on the total amount of words. This was done as further down the line, when calculating probabilities of sentences, both of these dictionaries were needed. Once the laplace was working as intended, the UNK model was worked on. At first there was an error that the UNK token could not be found in the dictionary, to combat this an extra if statement was added that if it was not found, it would be made and then on just add one every time.

When it come to perplexity, at first a method for calculating the perplexity of a sentence was made just to test and see if the basics and methodology worked as expected. This all worked apart from the UNK model with 2 n-gram length due to the same reasoning as the probability before. Moving to the perplexity of whole files i.e., the testing set, a main problem that was encountered was the format that the sentences were in, as the method to calculate the perplexity needed the sentences as if they were typed normally. In order to accomplish this, the words from the whole testing set were put into a list with the start and ending tokens included. Then the list was looped through and every time there was a word it was added to a temporary list until an ending tag was found. When an ending tag was found, then that sentence would be added to another list. In this way there would a list filled with lists of sentences. However, still not in perfect formatting as the words were still tokenised.

To combat this another loop was made that just concatenated the words together and then a final list of each sentence as its own string. Finally, the calculations were made.

Another problem that was encountered was when exporting the models to JSON format. This was due to the fact that in the dictionaries for the models, the keys were a tuple, and the JSON library does not allow the exporting of tuples or lists as keys. To rival this, a method was made that takes the model as a parameter and creates two lists; one for the keys and one for the values. Then a new dictionary is made and the key list is looped through. Every cycle in the loop converts the tuple name in the list into a string and the new dictionary is updated with the values. Using that new dictionary, the JSON file was made and exported.

Perplexity Results for Testing Set

	Unigram	Bigram	Trigram	Linear Int.
Vanilla	1.000058160511728	1.0000614398375844	1.0000245659010842	0.998134694600608
Laplace	1.0000582170656382	1.0000614356522795	1.0000243883435866	0.998134694600608
UNK	1.0000581496552683	0.9981248197406482	0.9997342738282492	0.998134694600608

https://drive.google.com/drive/folders/1FPjmkjjztSE3naxCVgPPVG32vyE4RG9w?usp=sharing – link to Google Drive containing the models