2021

ANLE Coursework 1 Report

CANDIDATE NUMBER:184521

Table of Contents

Problem Outline

The Microsoft Research Sentence Completion Challenge (MRSCC) [1] requires a system to be able to predict which is the most likely word is to complete a sentence by choosing between four 'imposter' words and one genuine word. The data that makes up these sentences was constructed by Project Gutenberg's data with sentences selected from Sherlock Holmes novels, and imposter words suggested by a natural language model trained on 522 19th century novels. However, there are many different approaches when looking to complete the challenge as there are many different models that can be applied to the task. Therefore, the goal of this project and the MRSCC in general is to find what models can complete the challenge with highest accuracies whilst only using the 522 novels as training data. The reasoning behind completing this challenge is to stimulate the research of semantic models which prior to 2011 was an under-researched field. For this report Unigram and Bigram models are used as a baseline for performance as they are both simple models with average accuracies. Trigram and Word2Vec models shall than be built and run against each other and the baselines to see which completes the challenge with the best accuracy.

Methodology

The Methodology for Completing the MRSCC involves building up Trigram and Word2Vec models that once fed the appropriate sentence and words choose which word is the correct word to fill a gap in the sentence. Each of these models though has a different approach to solving this problem which is what this section shall detail.

Baseline Methodology

The Unigram and Bigram models work similarly as they are both a type of N-Gram but with the key difference being how many words are considered for the context at a time. For instance, a Unigram would only consider the probability of the token word when passed to it and not any context. However, a Bigram takes two words into account at one time with these being the queried token word and the one before the token, called context, allowing the probabilities to be more precise and a better sentence generated. The reasoning behind this is the to better fill a gap in a sentence the context must be understood therefore feeding a word prior to the gap helps give this context as N-grams value context. To create these the models must be trained on a selection of 19th century Sherlock Holmes novels with the Unigram counting the frequency of every word whilst the Bigram does so but including the previous word too as a key. These frequencies are later converted to probabilities in training. This is done with every line of every novel given to the models in training. The Bigram shall also be given an extra parameter of Kneser-Ney smoothing which shall be used to calculate the probability distribution of the Bigram based on previous parts of the files. Once built these two models can be fed sentences from the MRSCC with the Unigram taking only the queried word and the Bigram taking this as well as one before the gap as well, these are then used to guess the probability of the word that will follow. Once the probabilities assigned to these words are gathered by the get_Prob function, the highest probability is selected along with word associated with it. This chosen option is then returned and this result is scored as correct or incorrect and added to the total score which is tallied after all questions have been run and used to calculate the accuracy. These accuracies form the baseline of performance when scoring and building the other models.

Trigram Methodology

Trigram Models are closer to the base line models of Unigrams and Bigrams than Word2Vec as it is also an N-Gram with a Trigram storing the probabilities of a word following two words of context instead of one or zero like Bigrams and Unigrams. This is done by creating a dictionary of phrases, each key consisting of two words, followed by a dictionary of words each with a frequency value associated with them. This frequency value is what is used to determine the probability that a chosen word follows the previous two-word context. This is done by training in the same way as with Unigrams and Bigrams but by counting the frequency of every word that follows the previous two instead of one or none. The Trigram shall also have smoothing parameters enabled with this model using Kneser-Nay smoothing in the same way as the Bigram but using the Trigram to calculate the probability distributions instead. For the MRSCC, the two words before the gap in the sentence form the context and are given to the Model, the probabilities given back by the dictionary are then observed so only ones that are possible answers are returned. The returned word with the highest probability is then chosen and returned to the algorithm so that it can be scored and used to calculate the accuracy of the model. This accuracy can then be compared to the baseline scores as well as the Word2Vec model's in order to gauge their and the Trigrams comparative performance.

Word2Vec Methodology

Word2Vec is a model obtained from the genism library and works by representing words features as vectors and then groups similar words together in a vector space. Therefore, when given a tokenised string of words it should be able to give the most similar words to follow the given context. This is as a Word2Vec model has the ability to recognise the context of individual words, when supplied with enough training data, by looking at its use in previous appearances. This ability to recognise the context of words is what is utilised in this project when comparing it to Trigrams and the base line models. This by utilising this to gather the context of words in the sentence and seeing what words in the vector space best relate to context of the sentence. These words are then iterated through to see if any of them are any of the possible answers and if an answer does relate to the sentence its probability is taken and stored in a list. This list is then used in the same way as the N-Gram models with the answer with the highest probability being taken and used as the answer for that question. These probabilities are gotten through the Word2Vec models built in predict_output_word function which outputs the possible words and their probabilities of being related to the sentence. In building this model however it was found that by default this function only gave back the top 10 probabilities and therefore an extra parameter had to be given to extend this. This parameter was called Topn and was tested at various values starting with 100 but moving up to values of 10000 before settling at 200,000. Although higher values were tested the time to train grew exponentially whilst giving diminishing returns in performance. In theory, for each question only one answer's probability needs to be gained cause the first one to be discovered will be the highest and will be selected as the appropriate answer. This is as the predict_output_word function returns the words with their probabilities from highest to lowest. Therefore, any subsequent probabilities will be forgotten and therefore the ideal Topn value will be one that gets at least one probability for each question with approximately 250,000 being found to be this ideal value.

Analysis and Evaluation

This section gives my analysis for the accuracies of all the models trained in this paper with each being run ten times in order to record their average accuracy when completing the MRSCC. The results of each of the ten runs are noted in Figure 1 as well as the calculated overall average of each of the trained models. It should be noted before continuing that all models, but the Unigram, possess a function that if no answers probabilities are found a random answer shall be chosen instead. This gives all the models shown here at least an accuracy of approximately 20% as this is the same as the 1 in 5 chance, they have of picking the correct answer randomly in these circumstances. Therefore, moving forward anything above 20% would be considered a success as it performs better than what a random pick could achieve. This also means that difference between each run of a model is likely caused by the difference in options picked by the random function which is why the Unigram's results are all the same.

Run No.	Unigram	Bigram	Trigram	Word2vec
1	24.904%	27.115%	27.885%	48.269%
2	24.904%	27.212%	27.788%	47.981%
3	24.904%	26.827%	27.404%	46.731%
4	24.904%	26.731%	27.500%	44.135%
5	24.904%	27.019%	27.788%	44.327%
6	24.904%	27.212%	28.750%	44.327%
7	24.904%	27.019%	28.654%	45.000%
8	24.904%	26.923%	28.269%	46.827%
9	24.904%	27.019%	27.404%	45.769%
10	24.904%	26.923%	28.558%	46.154%
Average	25%	27%	29%	46%

Figure 1: Accuracies & Averages Recorded from each Model

On observing the data of Figure 1 there is an obvious improvement in the n-grams as they view more of the question as context to the given token, with these context's being zero extra words to one extra word to two extra words. This is with Bigrams improving on Unigrams with an average of 27% compared to 25% and then Trigrams improving on Bigrams with an average of 29%. This shows taking more of a sentence as context leads to the model being able to predict the following word a lot more accurately with approximately a 2% increase in accuracy with every word gained in context. This would be an interesting hypothesis to test in the future to see if this increase continues with quadgrams or even pentagrams. However, I would theorise that there would be an upper limit to this and that the low amount of training data would cause issues when going much further than a Trigram. This is as with Trigrams there were already a large amount of cases where the model was unable to find a single word that related to the context given to the model. For instance, in the final run more than 60% of the results had none or very few of the possible answers in the Trigrams dictionary and this is likely to increase with the size of the N-gram. Therefore, with a three- or four-word context plus the token the models dictionary would need to have very precise keys that were unlikely to be obtained from the 522 documents used in training. This would therefore likely lead to a lower accuracy when completing the MRSCC.

The accuracies gained for the N-grams are supported by various papers that were researched prior to this project such as Sentence Completion Using Text Prediction by Gangwar et al[4] which showed that trigrams have an average accuracy of 31%. This is slightly above the 29% that my Trigram was able to perform at when tested though they were able to exceed this in their testing with their Trigram reaching 38% in some cases. This means that the Trigram model that was built here ended

up performing worse than expected. One paper, MRSCC by Zweig and Burges was able to get the Trigram accuracy to 36% when using the MRSCC training data but used a different smoothing method to this paper. They also only kept words in their trigram that appeared more than five times as well as some other different parameter changes which may account to this 7% increase in accuracy. Other papers such as Computational Approaches to Sentence Completion [2] were able to get a Trigram to perform with an accuracy of 39% but this was not using the MRSCC training or testing data as instead they used a larger dataset and SAT questions. This is a trend that is also shared by Exploiting Linguistic Features for Sentence Completion [3] who trained their trigram on a larger dataset to complete the MRSCC and achieved an accuracy of 61.44% by combining Unigrams, Bigrams and Trigrams together. This is another method that could be used to complete the MRSCC but would have to be tested with just the 522 files of the MRSCC training data.

Looking closer at the base line accuracies given by the Unigram and Bigrams it is possible to see that Unigrams give the exact same accuracy across all ten runs whereas Bigram's accuracies change each time. The reason for this is that a Unigram pulls the highest probability for a token every time which can't change between runs. However, when a Bigram comes across a token that's not in its dictionary it shall randomly choose from the available answers or return the values it does have which may be for an incorrect option. This causes the accuracy of the Bigram to vary by 0.5% with accuracy values being between 26.7% and 27.2%. This same trend is also represented in Trigrams where the values vary by a larger 1.3% with values ranging from 27.4% to 28.7% for the same reason as Bigrams. The reason for these varying for a larger number may be due to the number of tokens stored for each dictionary key being more irregular in a Trigram. This is something that is likely to increase as the size of the N-gram increases. If this continued to increase at the same rate quadgrams results would vary by approximately 3% which could lower its average performance substantially and make it less reliable to use.

Comparing the n-grams performances to those of the Word2Vec model it is clear to see that the Word2Vec has superior accuracy to all three of the other models. This is as it boasts an average an accuracy of 46% which is 17% higher than Trigrams, with Trigrams being the next best model, whilst its lowest recorded result was only 44.1%. This lowest value was still found to be 15% above the next best. However, these results are still lower than expected, as shown in "Evaluating distributed word representing distributed word representations for predicting missing words in sentences" by Saifee [5]. This paper tested two different Word2Vec models and was able to record accuracies of 52% and 57% which is more than the 46% average found here. Though, as with other Trigram and Bigram models shown previously, these were trained on larger datasets and setup differently allowing them to perform more optimally. It should be noted than when testing different parameters, it was possible to get the Word2vec model performing on par with these models at approximately 50% but the long execution times meant this had to be lowered. To get this optimality the Topn value can be set to 1 million though this this was only gave about a 4% gain for a much longer run time.

Reasoning why the Word2Vec model performs so much better than the N-Gram models is that due the Topn value they are much less likely to resort to a random pick as they are more likely to find a value. Therefore, the answers that the Word2Vec does not get are likely because the word with the highest probability was wrong likely due to training. Another reason the Word2Vec model got more correct is also due to it having the whole sentence as context and not just a few words allowing it to predict better.

Conclusion

In Conclusion, trigrams seem to be the best model to complete the MRSCC, from the selection tested in this paper, though it is likely there are better solutions that have not been tested. The Word2Vec performed best when compared to its peers which was unexpected when you look at the more precise nature of its context when compared to the simple n-grams. This was also backed up by multiple pieces of research gathered prior to researching this project.

In the future it would be beneficial to test other models such as combing multiple n-grams so when one does not have the key in its dictionary it checks the next n-gram down. For example, if a trigram doesn't have that key in its dictionary it checks bigram then unigram before returning zero if its not in any. This would likely improve the overall accuracy of these models as it would eliminate the short comings of both long and short n-grams. This has already been completed by Exploiting Linguistic Features for Sentence Completion [3] but they used a dataset larger than that of the MRSCC which bolstered their accuracy.

Another piece of future work that would be beneficial would be testing the hypothesis of as the size of an n-gram increases the accuracy increase by 2% every time. This would be done by evaluating n-grams such as Quadgrams and Pentagrams to see if they perform better as previously described in the Analysis. Although, as also previously mentioned, this would likely not be the case with such a small dataset of 522 files being used.

A final option would be to increase the size of the dataset available to the models as this would likely increase the accuracy of the models substantially as shown by [2]. However, this would result in the models being unable to officially perform in the MRSCC which requires only the 522 files to be used as training. This a method has already proved to be successful with trigrams in [2].

Therefore, the best-case scenario for the MRSCC would be to use a Word2Vec, perhaps in combination with a Trigram or Bigram for when it can't find the appropriate word, as Word2Vec has been shown to be the most accurate of the models in both this study and others.

Bibliography

Bibliography: 44 Words

Total Word Count=3018 Words

[1] Microsoft Research Sentence Completion Challenge; https://www.microsoft.com/en-us/research/publication/the-microsoft-research-sentence-completion-challenge/

[2] Computational Approaches to Sentence Completion https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.4850&rep=rep1&type=pdf

[3] Exploiting Linguistic Features for Sentence Completion https://www.aclweb.org/anthology/P16-2071.pdf

[4] Sentence Completion Using Text Prediction

https://www.cse.scu.edu/~mwang2/projects/NLP_sentenceCompletionByTextPredict_18w.pdf

[5] Evaluating distributed word representing distributed word representations for predicting missing words in sentences https://academicworks.cuny.edu/cgi/viewcontent.cgi?article=1631&context=cc_etds_theses