Machine Learning Group Report

Paraskevas Kryonas, Mai Tran, Anne Eschenbruecher
Machine Learning
Instructor: Dr Emad Mohamed

July 19, 2020

Contents

1 Introduction

As part of the Machine Learning for NLP course of our MA Computational Linguistics, we had to complete a group project. For this, we had been given a text file which had been stripped down to a long string of characters. As an example, a text such as *"I study Computational Linguistics. I like it."* would have looked like *"istudycomputationallinguisticsilikeit"*. The group project consisted of a total of four individual problems to recover the text and make it readable. Firstly, word boundaries were to be detected and established. Secondly, after having found the words the text is made up of, the next step was to detect the sentence boundaries and insert full stops wherever our system would deem appropriate. Thirdly, commas were to be inserted into the already established sentences. And lastly, capitalisation was to be restored.

2 Problems

2.1 Word Boundary Detection

The first problem to be dealt with was that of word boundary detection. Word boundary detection, in simple terms, is the task of finding where one word starts and the next word begins. In our example above, the desired output for this task would be *"i study computational linguistics i like it"*.

The training data used for this task consisted of the entire Gutenberg corpus within NLTK. Furthermore, the entire Brown corpus was used, as well as the testing files of the Reuters news corpus. All of the data was compiled into one big file, stripped of spaces and punctuation marks, and made entirely into lowercase characters so as to match the file we had been given. In total, the data consisted of 15,340,314 individual characters. For training and testing purposes, the data was shuffled and then split up with a 80/20 ratio. The training data hence consisted of 12,272,251 and the testing data of 3,068,063 individual characters.

To detect word boundaries, we used the IB1-IG algorithm within TiMBL, an open-source software package. The algorithm is a version of a k-nearest-neighbour algorithm that uses feature weighting. For TiMBL to be able to read the training and testing files, they were transformed into sliding windows with a size of fifteen characters per window. Seven characters, then the character the algorithm focuses on followed by an additional seven characters. In order to not skip a few characters in the beginning of the file, '=' characters were inserted as placeholders. Finally we inserted '0's and '1's into the file to indicate whether the character in focus was followed by a space (1) or not (0). The files could then be fed into the system for training and testing.

A total accuracy of 0.987 (99%) could be reported on the test set. This trained algorithm was then used to make predictions on the file we had been given for the group project and the output was captured in the file *word_boundary*

_detection_output.txt.

2.2 Sentence Boundary Detection

The next task was sentence boundary detection. This means that we would take the output from the word-boundary detection task and see if our algorithm could detect individual sentences. In our example above, the outcome for sentence boundary detection would be *"i study computational linguistics. i like it."*, where individual sentences are marked off with a full stop.

The training and testing data for the sentence boundary detection task consisted of the book "Emma" by Jane Austen taken from the Gutenberg corpus. In total, Emma consists of 166,725 which were preprocessed similarly to the data for the word boundary detection task. Firstly, the data was entirely stripped of punctuation marks and made entirely lowercase. Since spaces had already been predicted, these were left in so that we were working on a word level for this and not a character level like we did in the last task. This time, a window of 11 tokens was taken, each token being a word instead of a a character. That means there were five words, followed by the word in focus, followed by another five words. Each row was then followed by either a '0' or a '1' depending on whether a sentence ends after this row (1) or not (0). Furthermore, we also inserted placeholders here into the first few rows. The data was then shuffled into a training and a testing part with a 80/20 ratio. The training data hence consisted of 133,380 words (and hence rows) while the test data contained 33,345 words (and hence rows).

As algorithm, we chose once again TiMBL's IB1-IG algorithm which implements a K-nearest-neighbour approach with feature weighting that stores representations of the training set in memory.

We could report a total accuracy of 29.34%. It was attempted to improve

this percentage by additionally adding part-of-speech tags into the "Emma" text file, however, accuracy went down to 28.97% so that the first algorithm was used. A more detailed analysis of all errors can be found in the "Error Analysis" section below. The trained IB1-IG algorithm was then used to make predictions on the output from the first task in order to insert full stops at each sentence boundary.

2.3 Comma Detection

Thirdly, after inserting full stops at the sentence boundaries, the next task at hand was the prediction and insertion of commas. As an example, a sentence such as *"i love computational linguistics it is a great field."* would ideally have an output such as *"i love computational linguistics, it is a great field.*

In order to accomplish this, we once more used "Emma" by Jane Austen, taken from the Gutenberg corpus. This time, the text was merely stripped of all punctuation marks that are not full stops and then made entirely lowercase. Once again, a sliding window approach was used with a window of 16, however, this time the word in focus was the 16th word. In a 17th column we stored the response variable which again consisted of '0's and '1's so as to show the algorithm whether there would be a comma after the word in focus (1) or not (0).The data from "Emma" consisted of a total of 174,779 words (and hence rows). The data was split into train and test with a 0.75/0.25 ratio which is the standard ratio in H20.

To detect commas, three algorithms were implemented: Extreme Gradient Boosting, Gradient Boosting Machine, and Distributed Random Forest. The algorithm that achieved the highest results was the Gradient Boosting Machine algorithm using 50 trees per instance with 10 rows and a maximum depth of 2 per tree. The algorithm had a learning rate of 0.2. Before using "Emma" annotated with POS-tags, we first tried to implement an approach using data

with just one word per column. However results proved to be better when the data was annotated with part-of-speech tags.

Results show an accuracy of 98.1%, precision of 89.9%, and recall of 80.7%. The predictions were saved as a text file which was then used as input for the capitalisation detection task.

2.4 Capitalisation Detection

Finally, the last task at hand was that of restoring capitalisation within the text. This included capitalisation at the beginning of each sentence, as well as the capitalisation of proper nouns and "I". As an example, the sentence *"i study computational linguistics. i like it."* would ideally have an output such as *"I study Computational Linguistics. I like it."*.

The data used for this task also consisted of "Emma" by Jane Austen taken from the Gutenberg corpus within NLTK. This time, the data was stripped of all punctuation marks that are not in the prediction file (such as question marks, quotation marks, etc...) and made entirely lowercase again.

The window used for capitalisation detection consisted of eleven characters. Five words followed by the word in focus followed by another five words. Each row was followed by a '0' or '1', depending on whether the word in focus was to be capitalised (1) or not (0). Again, placeholders were used for the first five rows in order to not skip the first five words.

The algorithm used was again TiMBL's IB1-IG algorithm as we achieved quite good results with it during the previous tasks. Results show an accuracy score of 78.1% on the testing data. We tried once again to add part-of-speech tags to the data in order to improve the output, however, no better results could be reported so that the output generated for this task stems from training and testing data with no POS-tags.

3 Error Analysis

3.1 Word Boundary Detection

When looking at the predictions from the word boundary detection task, it is quite obvious where most of the errors are: There are not enough spaces. The algorithm seems to have been troubled with small words that only consist of one to three characters, such as e.g. *"i", "of"*, or *"our"*. Many of these small words are still merged onto other words. Curiously though, whenever the algorithm had the choice between a word and a slightly shorter word, it went for the shorter version. This can f.e. be seen in *"chalice"* which was predicted as *"ch alice"*.Often, the algorithm also inserted spaces in a way that left single characters appear as words. Even though this is incorrect in many cases, occasionally it does so because the single character is indirectly part of another word. As an example, the algorithm predicted *"child s"* which looks incorrect at first glance. However, when looking closely at the text, the whole phrase includes *"child s play"* which would be *"child's play"* when punctuation marks are included, so the apparent mistake is actually not a mistake at all.

Furthermore, the algorithm had problems predicting spaces around proper names. This was to be expected as most of these names never occurred in the training data. The same can be reported for old or very infrequently used words such as *"bazaar"* which were also frequently not separated from their neighbouring words.

3.2 Sentence Boundary Detection

As was already briefly mentioned above, some spaces where detected where apostrophes would normally be placed. A similar phenomenon can be observed within sentence boundary detection where the algorithm predicted full stops

where we would usually place other punctuation marks such as commas, exclamation marks or question marks. This error could have been avoided in part, if during preprocessing, sentence boundary markers would have been inserted also where exclamation marks or question marks are in the source text. However, as our task was to merely predict full stops, they have been disregarded this time.

It is also worth mentioning that the accuracy that has been reported is inflated. This is because there is a mass of true negatives every time the algorithm didn't predict a full stop which distort the accuracy score. A better measure for the performance of the algorithm would have been precision as this is a way of reporting a sort of accuracy for the minority class.

Furthermore, our algorithm predicted a lot of false negatives resulting in huge chunks of the output text being without any full stops. This also works in favour of our precision score as a full stop that hasn't been predicted would influence that score, but it brings down the recall score.

One possible explanation for the under-performance of our algorithm is the window size that was used during testing and training. Each window consisted of a total of eleven words. However, many sentences are much longer than that so that the algorithm is effectively never trained on an entire sentence but only on sentence chunks. For the future, we suggest implementing an approach with a bigger window size to see whether this positively impacts results.

3.3 Comma Detection

Just as with sentence boundary detection, the major problem with comma detection also were other punctuation marks. When looking at where exactly the algorithm put commas, we can see that it put a lot of commas in places where fulls stops should have been. This is especially noteworthy in the prediction file where we did not predict all full stops during sentence boundary detection and

the algorithm hence inserted many commas in places where full stops were not predicted in the previous task, but should have been.

Additionally, the same problem as above occured where the system reports an inflated accuracy of 98.1%. This is also because there are a lot more true negatives in the text since there are only very few commas to be predicted overall. Here as well, precision and recall are better measures of the performance of the algorithm as both of these metrics leave true negatives out. Precision for this task was reported at 89.8% while recall was reported at 80.7%.

Since so many commas were inserted where full stops should have been and vice versa, it would be interesting to try and implement a classifier that does not make binary predictions but that tries to predict commas and full stops at the same time. This could be done with having a '0' as response for nothing, a '1' as response variable for a full stop and a '2' as response variable for commas. This classifier could also be implemented with part-of-speech tags in order to improve results.

3.4 Capitalisation Detection

Errors for capitalisation were difficult to analyse. The algorithm only ever capitalises characters at the beginning of each word, which is a good start. However, it misses quite a few proper nouns. This can be explained by the fact that many proper nouns that appear in the prediction data set never appeared in the training data set and the algorithm hence did not know them. Some words, though, may have appeared as capitalised proper nouns in the training data and were hence also capitalised in the prediction process even though these words did not appear here as proper nouns. One example for this is e.g the word "brown" where the colour is meant which was capitalised to "Brown", a last name that was present in the trainig data.

Our system almost always capitalises after a full stop. It is also nice to see that it caught a lot of "I"s and capitalised these.

Just like it was the case with the other tasks, there are a lot more true negatives in the data which is due to the fact that not every word is capitalised by nature. This, again, inflated the accuracy score so that it would be better to rely on precision and recall to tell us how well the algorithm performed that capitalisation detection task. Furthermore, our system predicted a lot more false positives than false negatives meaning that it was more prone to capitalise where no capitalisation was needed than it was to use lowercase where capitalisation was needed.

4 Contributions

In this project, there were many tasks to be dealt with and any problems to be solved. Even though programming tasks were mostly done by individuals rather than the group, there were some tasks that the group handled together. Among those were the discussion of algorithms and results at regular meetings. Especially in the beginning of the project, we often met to discuss different approaches to solve this problem and to distribute the work and see who would try what task. In addition to that, whenever one of us was stuck with a particular task or needed someone to look over their code, the others were there to help. We encouraged each other and were honest when we thought that a particular approach would not yield good enough results.

It is worth mentioning, that even though we only handed in the output and algorithm that gave us the best results, all of us tried ourselves on more than just one of the four problems, which ones exactly will be mentioned in the individual sections.

4.1 Paraskevas

Paraskevas created training and testing sets for the word boundary detection task, the sentence boundary detection task, and the capitalisation task. For the word boundary task, he used various corpora resources within NLTK. It is necessary to mention that Paraskevas was limited in the data he could compile by the fact that he was running the algorithm on a virtual machine on his computer which limited him in terms of the data size which he could feed into the algorithm and also the speed at which the algorithm ran.

For the sentence boundary detection, he also created several training/testing sets, among others from the Bible within the Gutenberg corpus, from "Emma" (which ended up being the one used for the output), and from "Emma" annotated with POS-tags. Adding POS-tags in order to improve results was Paraskevas' idea and even though annotation with POS-tags turned out to produce worse results within sentence boundary detection, it produced significantly better results within comma detection. Furthermore, Paraskevas also managed to create the best output for the capitalisation detection task, also from the "Emma" data.

Finally, he also created the preprocessing file which creates the sliding window for the word boundary detection task. This file was then taken by Mai and Anne and amended for the algorithms they used, but the original file was coded by him.

4.2 Mai

For the group project, Mai created training and testing sets for the word boundary detection task. For the word boundary detection task, she created data files with sliding windows from the fiction and news sections within the Brown corpus within NLTK. She then slowly added on the entire Brown corpus, all Jane

Austen files from the Gutenberg corpus and then also the Bible from the Gutenberg corpus. She then improved results even further by adding on the entire Gutenberg corpus as well as the testing part of the Reuters news corpus. The output file for the word boundary detection task , which would later on act as input file for the sentence boundary detection task, was created using an algorithm that was trained on that last set. Mai also created an even bigger training and testing set by adding on the training part of the Reuters news corpus, however, results could not be improved compared to only the testing part of the Reuters corpus.

Furthermore, Mai also created various training and testing sets for the sentence boundary detection task using the data mentioned above. Nevertheless, her results were not as good as Paraskevas' which is why they were not submitted.

4.3 Anne

Anne created training and testing sets for the word boundary detection task, the sentence boundary detection task, as well as the comma detection task. She used a variety of data for the word boundary detection task combining different corpora resources within NLTK. Her biggest dataset included all of Gutenberg and all of the Brown corpus plus additional resources downloaded from the Gutenberg Project online. In contrast to Paraskevas and Mai, who used TiMBL, Anne used the H20 Machine Learning toolkit. Within this, she played with different algorithms for all three tasks mentioned above. The algorithms include Deep Learning algorithms, Extreme Gradient Boosting, Gradient Boosting Machine, Naïve Bayes, as well as Distributed Random Forest. None of her implementations achieved higher results than Paraskevas' or Mai's did for the word and sentence boundary detection.

Anne got the highest results for the comma detection task and created the

output that was then used for the capitalisation detection.In order to do so, she implemented Extreme Gradient Boosting, Distributed Random Forest and Gradient Boosting Machine algorithms, the latter being the one being used to generate the output.

Anne also looked into the possibility of using sequence to sequence models implemented with Ludwig or OpenNMT, however, this idea was later discarded. Furthermore, she wrote this report from the notes created in the group discussions.

5 Conclusion

Overall it can be said that even though our results are not bad, much more is to be done in the future in order to still improve upon those. It could also be beneficial to implement an approach where machine readable dictionaries such as e.g. WordNet are used in the word boundary detection task in order to only allow those words to be predicted that appear as words in the dictionary and hence reduce false output.

Appendix

Files for Word Boundary Detection		
	File Name	Purpose
1	word_boundary_preprocessing.py	Code that compiles the data used for training and testing and creates the sliding window used as input
2	word_boundary_train_test_input.txt	File that was created with code above and serves as input for training and testing
3	prediction_text_preprocessing.py	Code that transforms the prediction text to a sliding window text file
4	word_boundary_prediction_input.txt	File that was created using the code above and serves as input for the prediction
5	collect_word.py	Collecting the output file from TiMBL and starting to read output
6	word_boundary_timbl_decoder.py	Transforming output into readable text file below
7	word_boundary_prediction_output.txt	Final output from word boundary detection task

Files for Sentence Boundary Detection		
	File Name	Purpose
1	sentence_boundary_train_test.py	Preprocessing of input text from word boundary detection to sliding window, training and testing of file
2	austen_sent.txt	File that was created with code above and is used for training and testing
3	story_sent_test.txt	File that was created with code above and serves as input for the prediction
4	as_test.txt.IB1.O.gr.k1.out	Out-file generated by TiMBL that contains predictions
5	sentence_out_to_text.py	Code that turns the above out-file into the readable text file below
6	sentence_boundary_prediction_output.txt	Final output from sentence boundary detection task

Files for Comma Detection		
	File Name	Purpose
1	commadetection_preprocessing.ipynb	Code that transforms output from sentence boundary detection task into sliding window to be used as input for GBM algorithm for comma detection
2	commadetection.ipynb	Code that trains the GBM algorithm on the file created with code above and extracts predictions. Also transforms predictions with POS-tags into full text without tags
3	comma_prediction_output.txt	Output file from comma detection task

Files for Capitalisation Detection		
	File Name	Purpose
1	capitalisation_train_test.py	Preprocessing of input text from comma detection to sliding window, training and testing of file
2	austen_cap.txt	File that was created with code above and is used for training and testing
3	story_cap_test.txt	File that was created with code above and serves as input for the prediction
4	ac_test.txt.IB1.O.gr.k1.out	Out-file generated by TiMBL that contains predictions
5	capitalisation_out_to_txt.py	Code that turns the above out-file into the readable text file below
6	story_capitalised.txt	Final output file with full capitalised text