In this group project, we aimed to compare different text classification models on a large dataset. We chose to work in the medical context, and found a dataset containing a large corpus of medical abstracts, each one labelled as belonging to one of five categories. We chose to assess the performance of the models on accuracy and by examining the confusion matrices produced, and focused more on model implementation and tuning than rigorous comparison of results. We decided to investigate embedders such as count vectorizer, tf-idf, Word2Vec and Bert as well as classifiers such as Naïve Bayes, Bert and Nueral networks. The rest of my group were keen to investigate their respective areas as they had prior experience, so I decided to work on preprocessing and the Bag of Words model (count vectorizer and tf-idf).

Text classification involves first embedding our text into a numerical vector space, this leaves us with a dataset that our established classification models can 'understand' and work with more efficiently than raw text. It is important that the embedding captures the relevant information from the corpus, and to classify the text efficiently, it is also useful to remove information that is not useful, for this reason, text preprocessing is required before we apply our embeddings.

Since the dataset is taken from the real world and is quite large, it was messy and difficult to handle, I therefore decided to use the broad stroke of removing all tokens that were not n-grams, this is not the optimal method as we are discarding information that could be useful to the classifier, however to tackle this problem in a way that can reliably be applied to the whole dataset would be much more involved and since the dataset is so large, we can afford to lose some of it to make the task more manageable. The rest of the preprocessing is fairly standard practice. Preprocessing was the most computationally intensive and least time efficient step in the method, however it is also easily parralelised by preprocessing multiple documents at the same time. Therefore if the dataset were increased in size dramatically, my method would still be applicable with parrelelisation incorporated.

Next was text embedding, count vectorizer and tfidf were tools that I was able to use of the shelf, and both produced embeddings that the classifier (once tuned) could perform well with. If the data were to be scaled up, count vectorizer could be easily parallelised to keep up the efficiency, since we can look at multiple documents at the same time to produce the dictionary and count the word frequency. Since count vectorizer also produced better results in the end, we would use it if working on a much larger dataset (with parrelelisation) so as not to bottleneck the method.

The multinomial naïve bayes classifier produced acceptable results when fed embeddings from count vectorizer, but struggled with tf-idf. This was partly due to the inbalanced data,

which could be fixed by setting uniform prior probabilities on the classes. It was also very time efficient, allowing us to conduct a gridsearch on the smoothing parameter and tune it to produce good predictions. This efficiency also means that the classifier would not be a bottleneck if the data was scaled up.

Overall the methods I used benefited from there simplicity in that they could be applied and tuned quickly to produce good results and could be applied to a much larger dataset (with parrelelisation). However, to produce better results, more computationally intensive methods such as neural networks would be required.