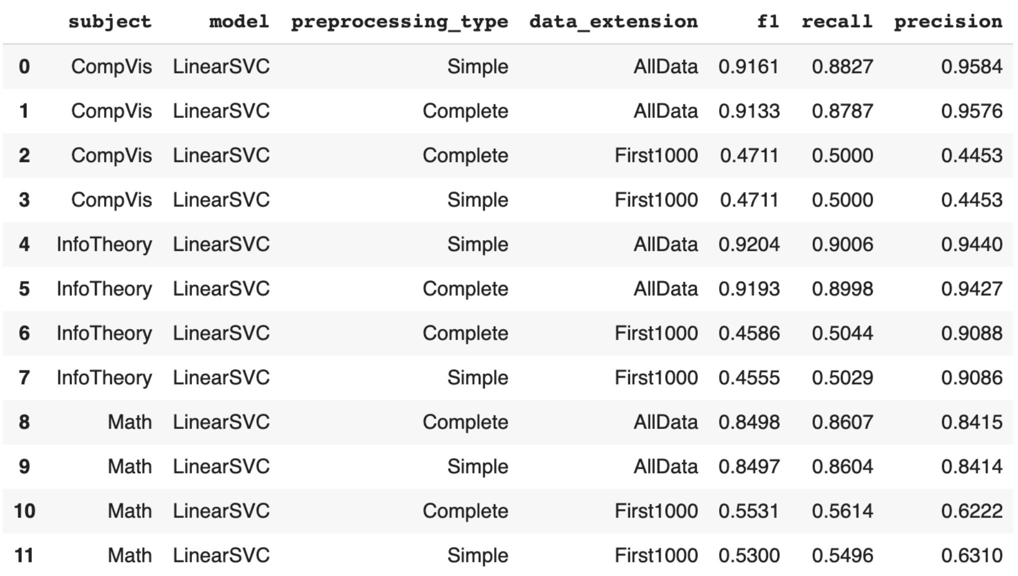
Text Classification

1.1 Methods

1.1.1 Statistical Methods

We first tried 4 different statistical algorithms for each of the 3 binary classification tasks, without any hyperparameter tuning, to be able to compare their performances. As the data was severely imbalanced, we used k-fold cross validation on the entire training set which I believe was a good choice because it can help avoid the potential issue of having too few instances of the minority class when splitting randomly with the validation set approach. If this happens, it can damage our predictions by leading them even more towards the majority class.
This was the preliminary model selection in which we compared the performances of Random Forest Classifier, LinearSVC, Naives Bayes and Logistic Regression and were able to discard 2 of those models, keeping only LinearSVC and Logistic Regression to continue with the hyperparameter tuning. We could see, after playing around with the hyperparameters for both of the pre-selected algorithms, that they both yielded better results in all the subjects once the best combination of hyperparameters was found. Particularly, LinearSVC outperformed Logistic Regression for all 3 subjects. This is why we ended up selecting LinearSVC as our definitive model. At this stage and the previous ones, we were working with only 1 configuration of data pre-processing.
Once we selected the definitive model, we trained it using 2 different configurations of data pre-processing: complete (involving multiple steps of pre-processing such as remove stop-words, short tokens, lemmatizing, including bigrams, etc) and simple (only word tokenization) to see if this had any effect in the final outcomes. We can see that the results we obtained by training on the entire training set and testing on the test set were not very different, comparing the different pre-processing configurations:



	subject	model	preprocessing_type	data_extension	f1	recall	precision
0	CompVis	LinearSVC	Simple	AllData	0.9161	0.8827	0.9584
1	CompVis	LinearSVC	Complete	AllData	0.9133	0.8787	0.9576
2	CompVis	LinearSVC	Complete	First1000	0.4711	0.5000	0.4453
3	CompVis	LinearSVC	Simple	First1000	0.4711	0.5000	0.4453
4	InfoTheory	LinearSVC	Simple	AllData	0.9204	0.9006	0.9440
5	InfoTheory	LinearSVC	Complete	AllData	0.9193	0.8998	0.9427
6	InfoTheory	LinearSVC	Complete	First1000	0.4586	0.5044	0.9088
7	InfoTheory	LinearSVC	Simple	First1000	0.4555	0.5029	0.9086
8	Math	LinearSVC	Complete	AllData	0.8498	0.8607	0.8415
9	Math	LinearSVC	Simple	AllData	0.8497	0.8604	0.8414
10	Math	LinearSVC	Complete	First1000	0.5531	0.5614	0.6222
11	Math	LinearSVC	Simple	First1000	0.5300	0.5496	0.6310

We are able to observe in the above table that the performances for the CompVis and InfoTheory models are better than the Math model, which happens to be the least imbalanced class (30% of positive class vs 19.25 and only 4.25 in CompVis).
In addition, we also tested our optimal model's performance, using both pre-processing configurations, having trained it using only the first 1000 rows of data (data_extension: First1000 in the table). In this case, the results were definitely affected. All the possible combinations of models involving only the first 1000 instances yielded substantially lower performance. This could have been due to the fact that, apart from the classes being imbalanced, the data was ordered in a certain manner and we could have ended up with very very few instances of the minority class if taking the

first 1000 rows instead of shuffling for example. Particularly for the most imbalanced class, CompVis, our 1K instances model predicted all majority class:

```
LinearSVC   CompVis
 Pre-processing: Complete
 Data Extension: First1000
 f1score:  0.4710783786689603
 recall:  0.5
 precision:  0.44531964630551885
 [[17526     0]
 [ 2152     0]]
```

It is a good moment to clarify that we used F1 Macro Score (Averaged) which returns an average of the F1 Scores for both classes. I believe this to be a good indicator of performance when trying different models to deal with imbalance because sometimes by trying to overcome the issue of the minority class you could end up compromising the performance of the majority class, which would not be reflected entirely in the binary F1 Score. In addition, if we would have used Accuracy as our metric, we would have ended up with almost 90%, which seems very good, but is really misleading, as the only thing our model is doing is predicting negative.
With only 1000 instances, the data is also clearly not rich enough for the model to learn properly. It is important to remark that this kind of behaviour would repeat in the use of other algorithms as well, so that is why we only explored the training with the entire training set cases for RNN LSTM and Bert.

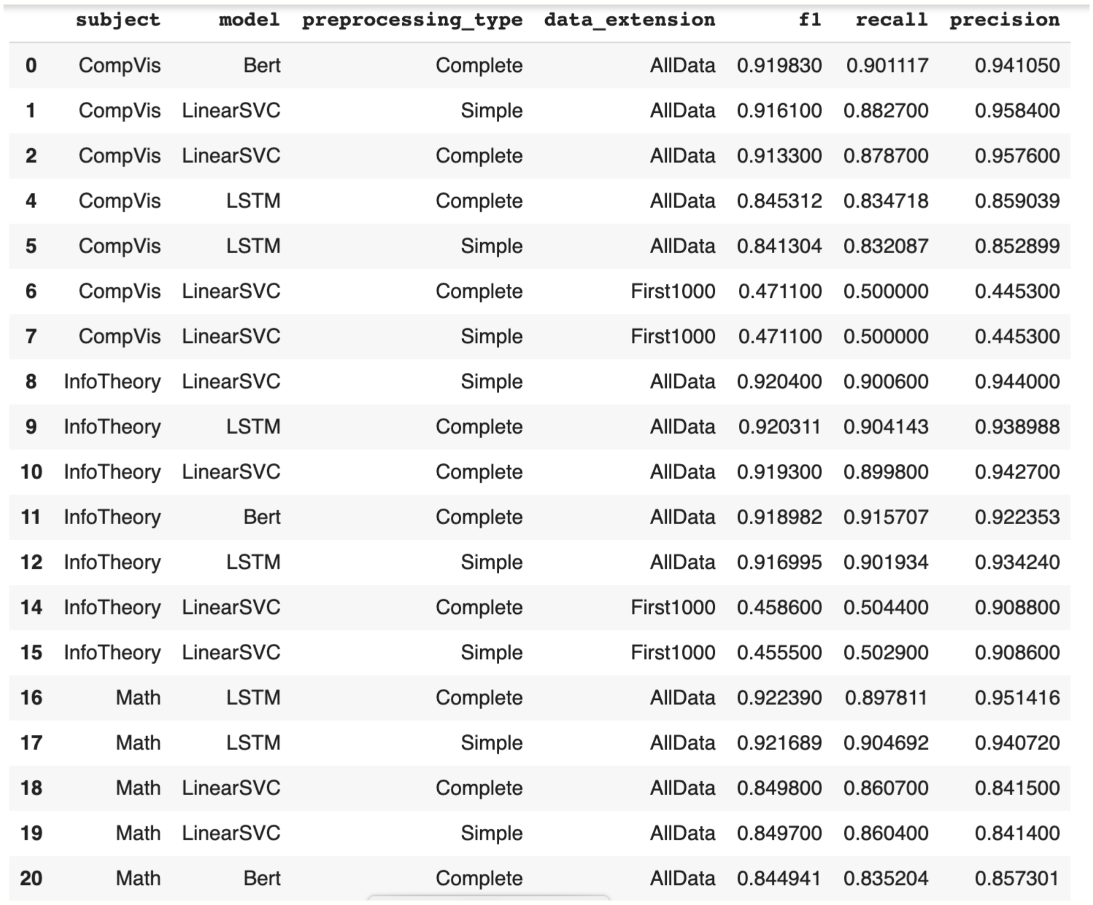
1.1.2 RNN LSTM Method
We first tried with a classic RNN architecture but the performance was way below the one for statistical methods. This method is an improvement that is able to overcome the problem of the vanishing gradients.
We also made use in this case of the same 2 pre-processing configurations. Different from the statistical methods in which we used tf-idf to vectorize our corpus, we utilised Glove's pre-trained word embeddings.
The statistical methods yield significantly better results in the most imbalanced class (CompVis) while LSTM has increased a lot the performance in the Math class. The results in the remaining class were pretty similar across both methods but the interesting thing here is that we didn't have to go through all the process of comparing different algorithms, tuning their hyperparameters, etc, making the implementation more straightforward.
It is also important to mention that the different preprocessing types didn' t seem to have any significant impact in the final performances.

1.1.3 Bert Method
We finally built a model based on the 12-layer BERT model, with one extra classification layer on top. This model did not require either any hyperparameter tuning such as the statistical methods.
We did face computational problems in this case: the parameter max_length (belonging to the Autotokenizer, that handles the tokenization, indexation, padding, masking, truncation),
which refers to the maximum number of allowed tokens as inputs of the transformer model, seemed to have a lot of influence over the running time and the final performance of the algorithm. We finally set it to 256 (half of the allowed maximum of 512), meaning that we were able to process up to 256 tokens per abstract while several abstracts, even after being pre-processed, had up to 700 tokens.
In this way, it took me about 1 hour to train each different subject's model (CompVis, Math and InfoTheory). With this in mind, we only trained 1 pre-processing configuration (complete).

Regarding Bert's results, it was the best model for the most imbalanced class CompVis and yielded similar performances in the overall.
I believe it is interesting to point out that if we were able to deal with the corresponding computational capacity and set maximum_length to 512, this model could probably outperform the other 2 in all the subjects.

1.2 Final Performance Results

The final performance results were retrieved in the following table:

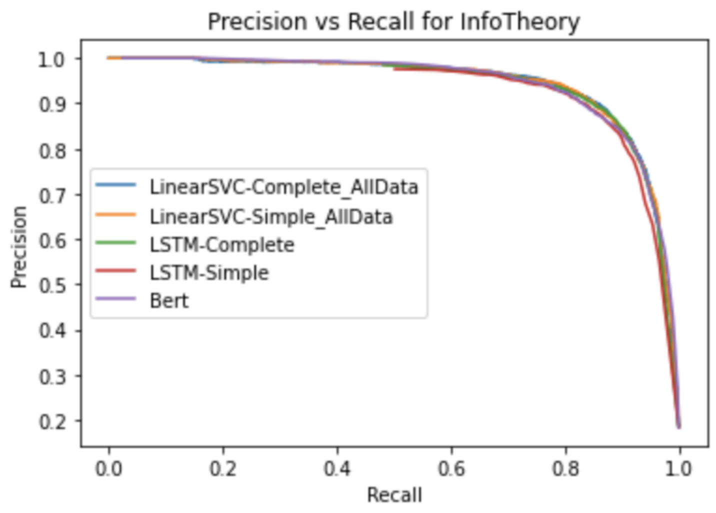


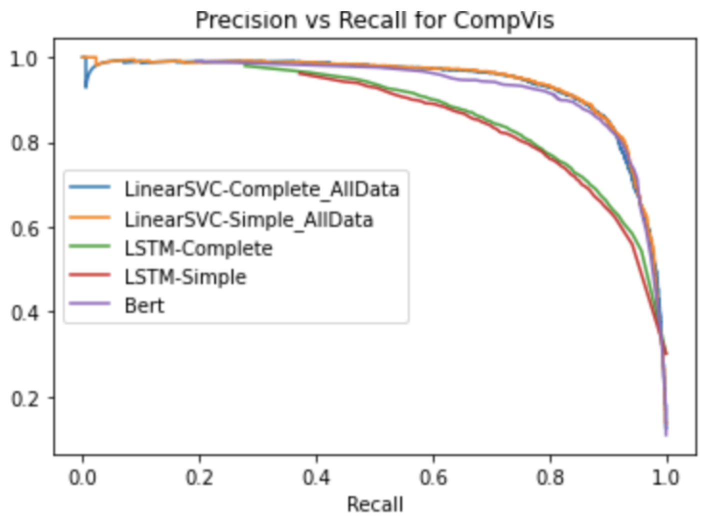
	subject	model	preprocessing_type	data_extension	f1	recall	precision
0	CompVis	Bert	Complete	AllData	0.919830	0.901117	0.941050
1	CompVis	LinearSVC	Simple	AllData	0.916100	0.882700	0.958400
2	CompVis	LinearSVC	Complete	AllData	0.913300	0.878700	0.957600
4	CompVis	LSTM	Complete	AllData	0.845312	0.834718	0.859039
5	CompVis	LSTM	Simple	AllData	0.841304	0.832087	0.852899
6	CompVis	LinearSVC	Complete	First1000	0.471100	0.500000	0.445300
7	CompVis	LinearSVC	Simple	First1000	0.471100	0.500000	0.445300
8	InfoTheory	LinearSVC	Simple	AllData	0.920400	0.900600	0.944000
9	InfoTheory	LSTM	Complete	AllData	0.920311	0.904143	0.938988
10	InfoTheory	LinearSVC	Complete	AllData	0.919300	0.899800	0.942700
11	InfoTheory	Bert	Complete	AllData	0.918982	0.915707	0.922353
12	InfoTheory	LSTM	Simple	AllData	0.916995	0.901934	0.934240
14	InfoTheory	LinearSVC	Complete	First1000	0.458600	0.504400	0.908800
15	InfoTheory	LinearSVC	Simple	First1000	0.455500	0.502900	0.908600
16	Math	LSTM	Complete	AllData	0.922390	0.897811	0.951416
17	Math	LSTM	Simple	AllData	0.921689	0.904692	0.940720
18	Math	LinearSVC	Complete	AllData	0.849800	0.860700	0.841500
19	Math	LinearSVC	Simple	AllData	0.849700	0.860400	0.841400
20	Math	Bert	Complete	AllData	0.844941	0.835204	0.857301

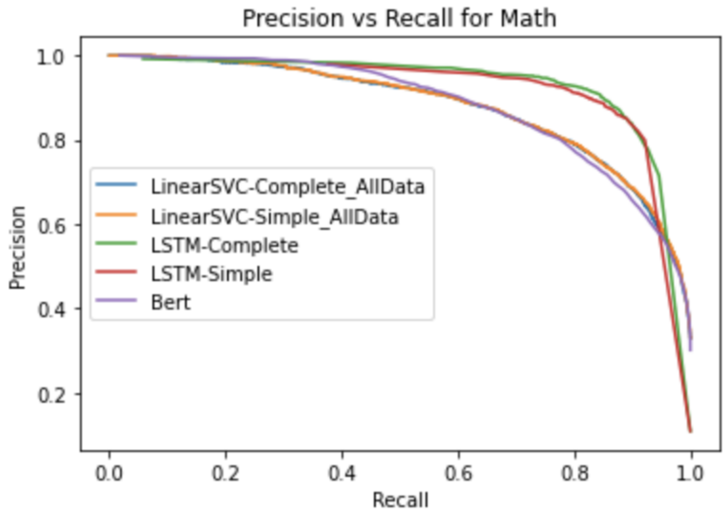
We are able to observe that, as said before, there is not much difference between the 2 different pre-processing techniques in any of the models.
All 3 models have 1 subject in which they do not perform as well (Math for LinearSVC and Bert; CompVis for LSTM). The 3 models perform quite consistently and similarly in InfoTheory.
In a more general sense, the 3 models performance is pretty consistent and similar, yielding each an average of F1 Score across all subjects (with the max performance for each subject) of almost 0.9.

1.3 Precision-Recall curves

We finally plotted the precision-recall curves of the different models for each of the 3 subjects, which in some sense, is a visual representation of what we just discussed above:







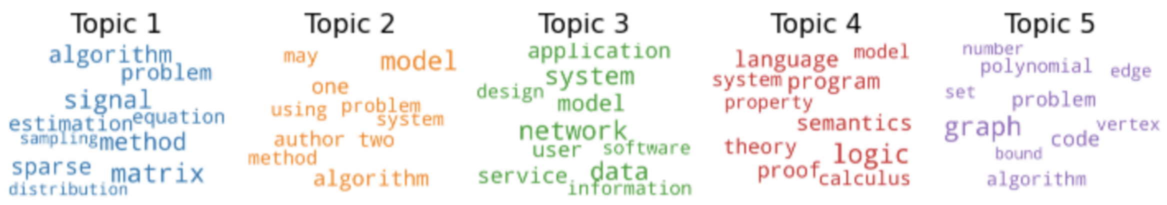
We can reach the same conclusions as before:

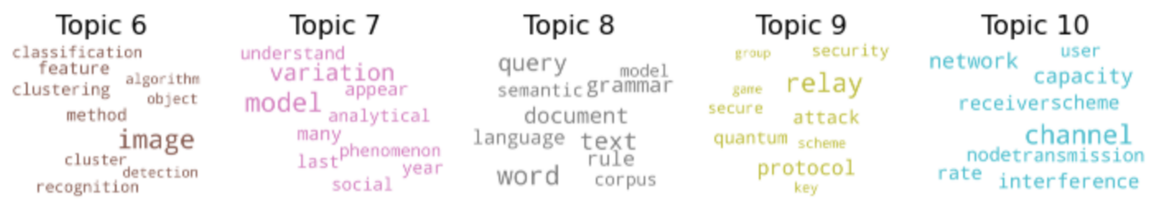
- There is not much difference between the 2 different pre-processing techniques, as each pair of Complete-Simple curves are almost always overlapped
- All 3 models have 1 subject in which they do not perform as well : Math for LinearSVC and Bert as we can see how their curves are a little below LSTM and CompVis for LSTM as we can see its curves falling behind the other model's.
- The 3 models perform quite consistently and similarly in InfoTheory, as we can see that all the curves are overlapped.
- I did not include the curves here but we can see in one of the plots presented in the notebook that for the case of the models trained only with 1K instances, the curves are way below the other ones, meaning that these models do not perform well.

Topic Modelling

2.1 First LDA Model

The topics found after training the first model were quite consistent. By taking a quick look at the following Word-Cloud we can start to have an idea of what each topic is referring to:





If we contrast this with the titles of the most meaningful articles for each topic (shown in the below image), we can quickly label a few of them:

TOPIC 8: NLP (or something related to it):

```
8: ['Statistical analysis of the Indus script using $n$-grams',
  'Best-first Model Merging for Hidden Markov Model Induction',
  'Does Baum-Welch Re-estimation Help Taggers?',
  'Analyzing and Improving Statistical Language Models for Speech Recognition',
  'Uniform Representations for Syntax-Semantics Arbitration'],
```

TOPIC 5: GRAPH THEORY

```
5: ['The geometric stability of Voronoi diagrams with respect to small changes of the sites'
  'Correlation Decay in Random Decision Networks',
  'Polynomial Bounds on the Slicing Number',
  'On Making Directed Graphs Eulerian',
  'On Dynamic Range Reporting in One Dimension'],
```

TOPIC 9: CYBER-SECURITY (or something related to it)

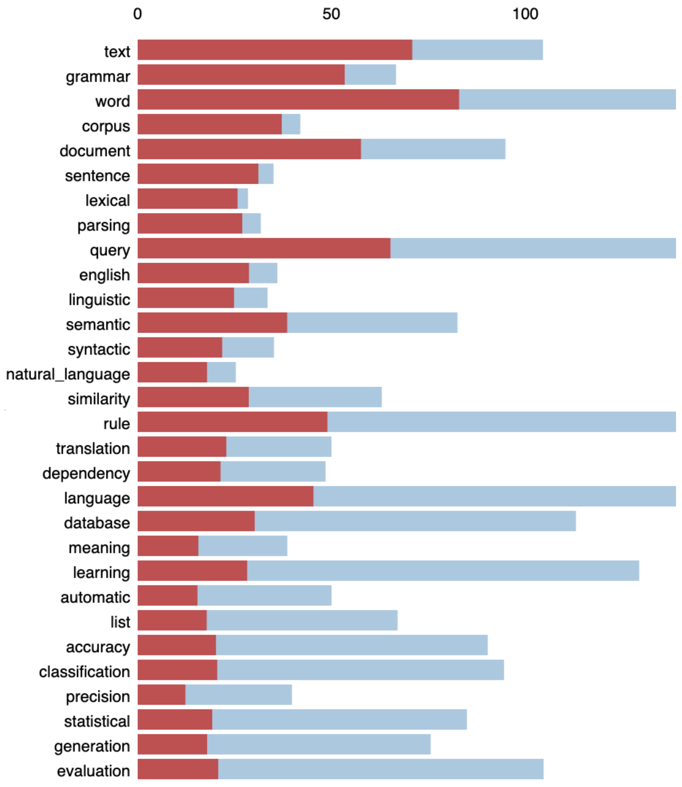
```
9: ['LDPC codes in the McEliece cryptosystem: attacks and countermeasures',
  'Notes on Recent Approaches Concerning the Kirchhoff-Law-Johnson-Noise-based Secure Key Exchange',
  'Distributed Relay Protocol for Probabilistic Information-Theoretic Security in a Randomly-Compromised Network',
  'Small Pseudo-Random Families of Matrices: Derandomizing Approximate Quantum Encryption',
  'New Extensions of Pairing-based Signatures into Universal (Multi) Designated Verifier Signatures'],
```

TOPIC 10: COMMUNICATION SYSTEMS

```
10: ['Traffic Dynamics of Computer Networks',
  'An overview of the transmission capacity of wireless networks',
  'The Enigma of CDMA Revisited',
  'A Turbo Coding System for High Speed Communications',
  'Performance Analysis of 3-Dimensional Turbo Codes']}
```

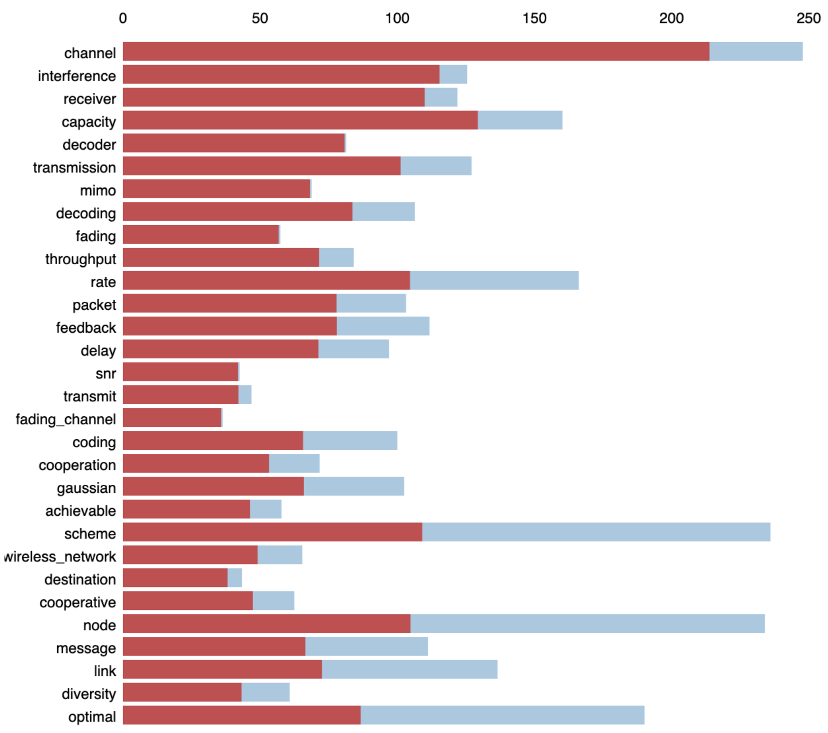
Not all topics are as easily detected, not because a relationship does not exist between the different articles under the topic but because it also requires domain knowledge to be able to detect this kind of relationships. This corpus is particularly technical so, of course, I will miss some of those relationships

when looking at the topics. But the important thing is that, without any deep particular knowledge of the domains, I already see topics that make sense.
To further understand if we are dealing here with a logical set of topics, we can explore the pyLDavis plot, which will give us detailed information about the keywords in the different topics, the similarity between the topics, etc.
If we take a look to TOPIC 8 for instance in the LDavis plot, we can confirm that the topic is definitely related to NLP:

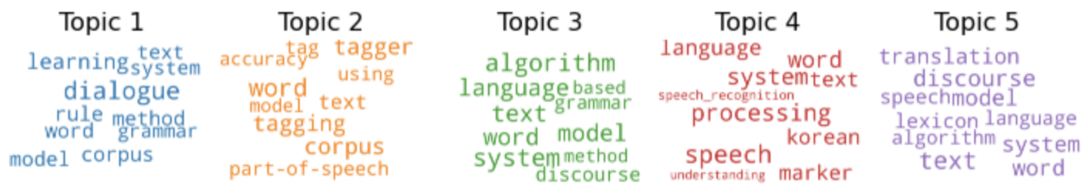


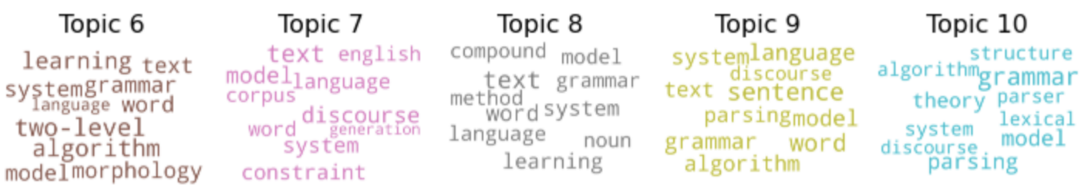
This distribution of words was achieved using a lambda of 0.4. By adjusting the relevance metric lambda we can identify the words that are more specific to the particular topic (the lower the lambda, the more specific the word).

Same thing with TOPIC 10, we can confirm its relationship with the Communication Systems domain:



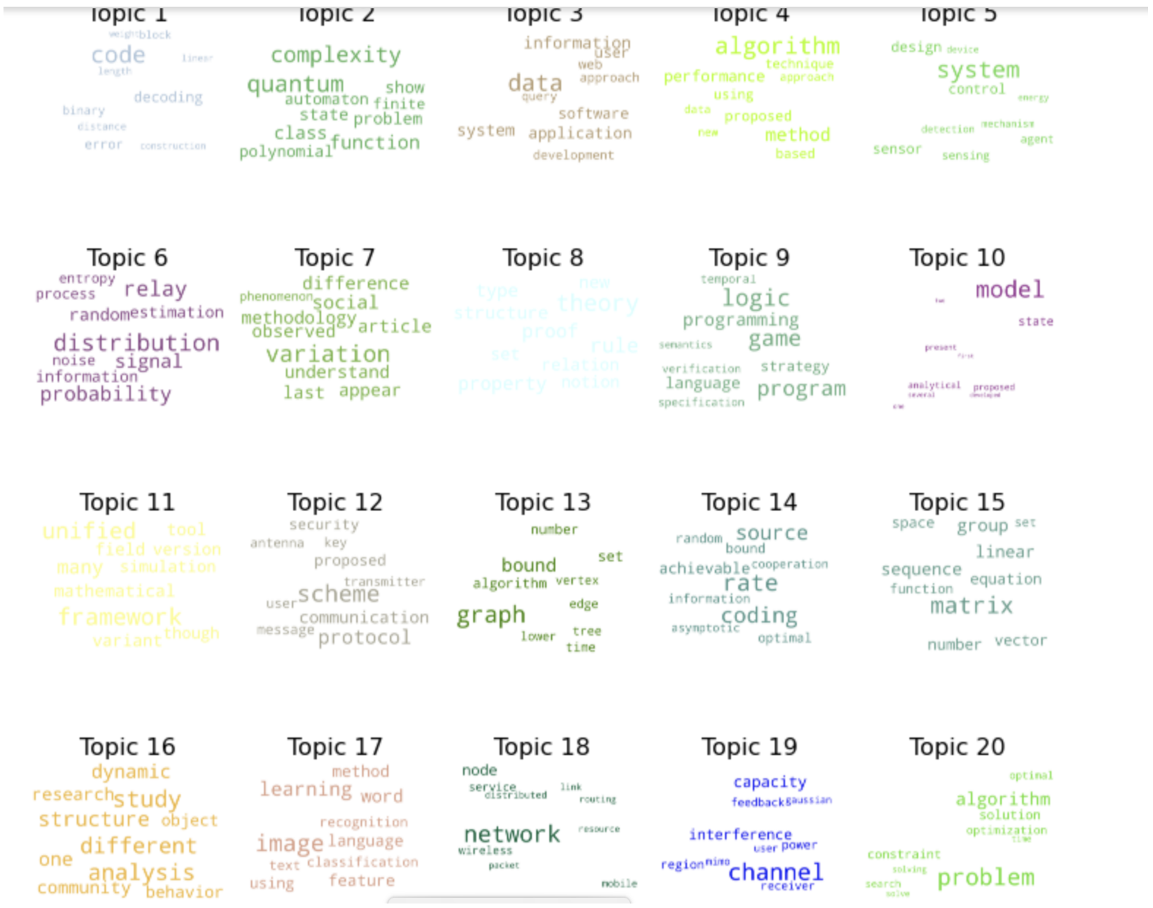
Using the same pre processing conditions but training only with the first 1000 rows of data, the results are not as consistent:





We can see that almost all the topics have repeated words and the theme NLP (or related) seems to be the central one in most of them. This could be due to the fact that the data is somehow ordered and all the first rows of data belong to that same subject. We could try shuffling the data and then subsetting with the first 1K rows. Whatever is the case, I believe that with 20K we will be able to better capture the true nature of the data which will produce better results (as it happened).

2.2 Second LDA Model



By conducting an analogous procedure with the second model, the resulting Word-Cloud is the above. We also seem to end up with relatively logical sets of words for each topic.
It is harder now though to manually assign each topic to a known label because we have doubled the number but still we can see some topics that appeared in the previous model:

TOPIC 12: CYBER-SECURITY

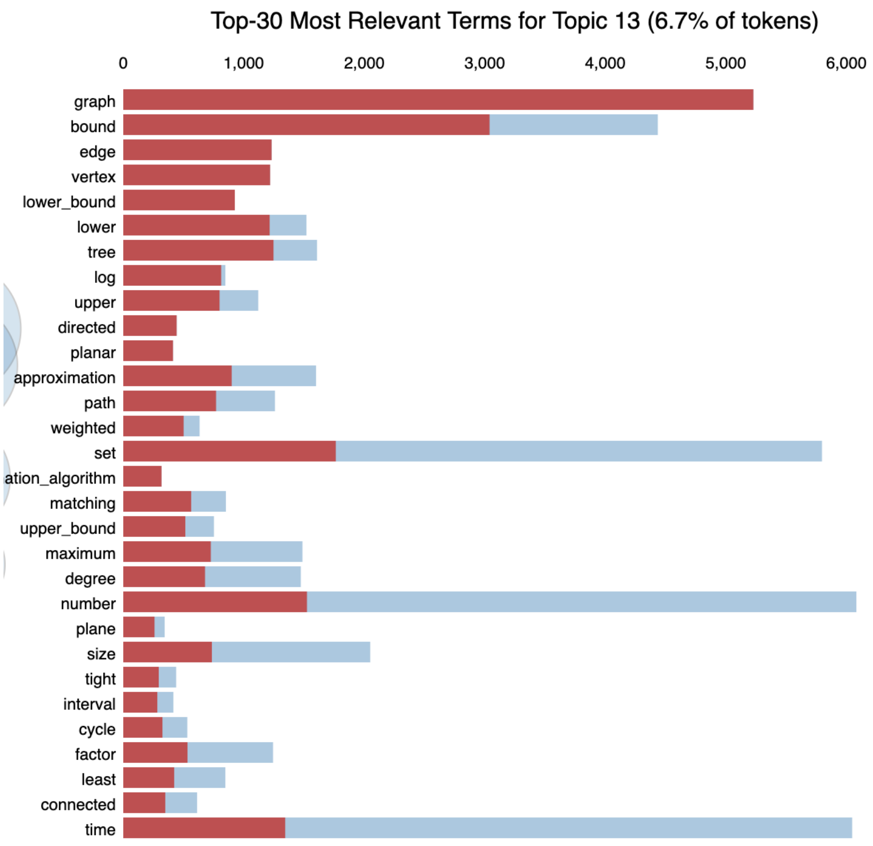
```
12: ['Efficient Authenticated Encryption Schemes with Public Verifiability',
 'Comment on A dynamic ID-based Remote User Authentication Scheme',
 'Weakness Analysis and Improvement of a Gateway-Oriented Password-Based Authenticated Key Exchange Protocol',
 'An Improved Remote User Authentication Scheme Using Smart Cards',
 'Cryptanalysis of three matrix-based key establishment protocols'],
```

TOPIC 13: GRAPH THEORY

```
13: ['Approximating subset $k$-connectivity problems',
 'Constructing Optimal Highways',
 'Spanners of Additively Weighted Point Sets',
 'Local Maximum Stable Sets Greedoids Stemmed from Very Well-Covered Graphs',
 'The rainbow $k$-connectivity of two classes of graphs'],
```

ETC…

We can confirm by taking a look at the LDavis plot for TOPIC 13 that it is something related to Graph Theory:

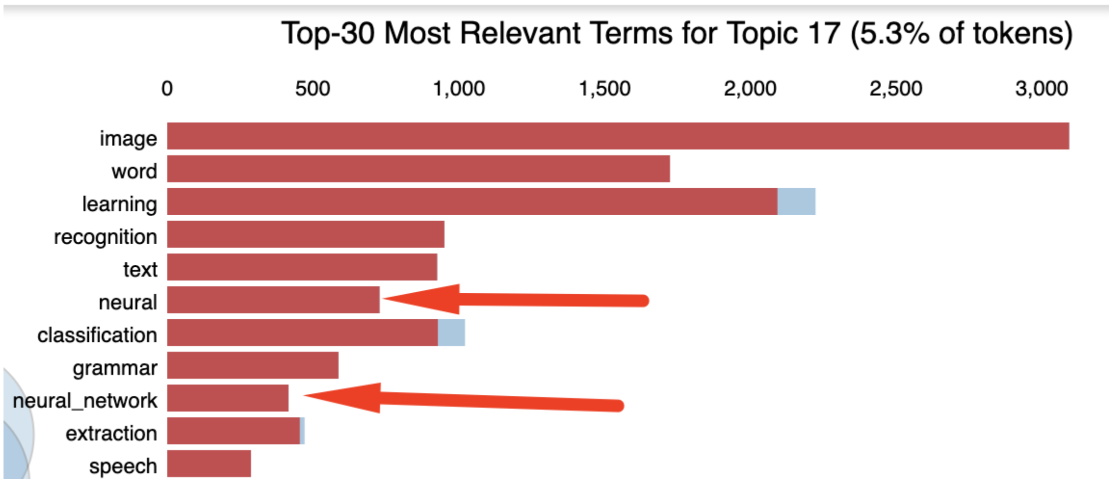


If we conduct the same analysis but training with the first 1K rows of data, the same thing as last time happens: almost all the topics revolve around Language.
This can be confirmed by showing the 2 most meaningful article titles for the last 5 topics:

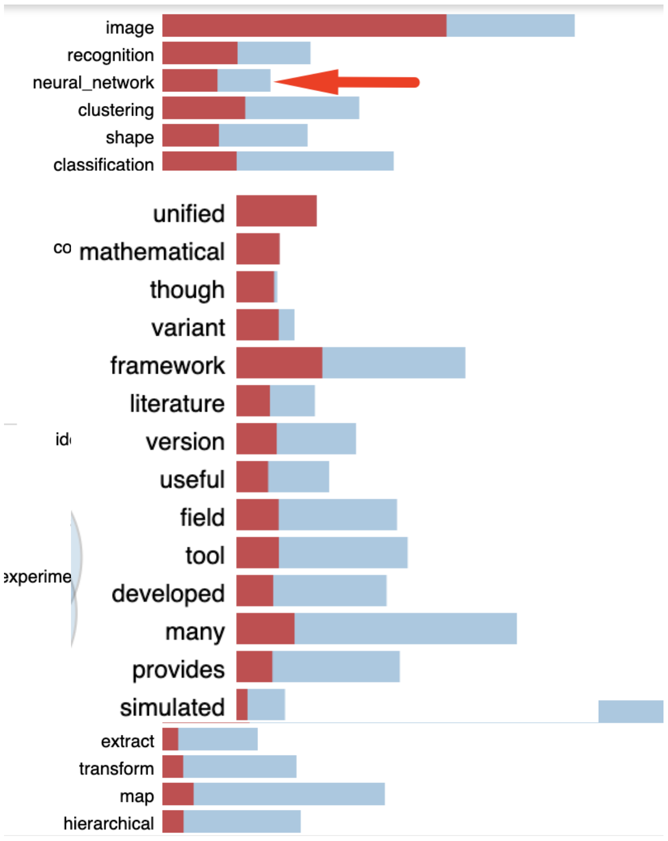
```
15: ['Using Qualitative Hypotheses to Identify Inaccurate Data',
 'Nonuniform Markov models'],
16: ['A lexical database tool for quantitative phonological research',
 'Tagging and Morphological Disambiguation of Turkish Text'],
17: ['A Compositional Treatment of Polysemous Arguments in Categorial Grammar',
 'ProFIT: Prolog with Features  Inheritance and Templates'],
18: ['Type-driven semantic interpretation and feature dependencies in R-LFG',
 'A Deductive Account of Quantification in LFG'],
19: ['Centering  Anaphora Resolution  and Discourse Structure',
 'Discourse Coherence and Shifting Centers in Japanese Texts'],
20: ['Phoneme-level speech and natural language intergration for agglutinative languages',
 'Automatic Discovery of Non-Compositional Compounds in Parallel Data']}
```

2.3 Comparison between models

Both models include lemmatizing and removing stop-words as part of their preprocessing steps. We have seen that these steps are very useful in order to avoid non-meaningful tokens to form part of the final keywords of the topics
Model 1 includes trigrams which at the end does not seem to make much of a difference. Not a single trigram has appeared among the keywords of any of the topics, not even when decreasing the lambda relevance value of the pyLDavis plot to the minimum, where words very specific of that topic should emerge.
Model 1 includes bigrams by replacing the original unigrams while Model 2 includes bigrams and the original unigrams that conform them. I believe that the first approach is better because sometimes with the second approach you can find the bigram and one (or both) of the unigrams that conform it as keywords of the same topic, like the following example:



Here we can see that neural and neural_network are being considered as key terms which I find to be a little redundant and irrelevant. I don't think neural is being used much on its own. Now, we can see the same case but for model 1, where we see that only neural_network is considered as key term:



Regarding the quality of topics I believe both models managed to output some very reasonable topics that can be easily identified, even without the need of domain expertise. Nevertheless, I believe it is slightly easier to relate with topics for K=10 (model 1). When navigating some of the topics keywords in the LDavis plot for model 2, where K=20, I can observe a few of them that seem like a unspecific and unconnected group of words:
Maybe this is an indicator that K=20 is too big.
Finally, we compare the coherence score for both models, which is a metric of how consistent are the topics (the higher the coherence, the better).

MODEL 1:

```
model_coherence = CoherenceModel(model=model_1, texts= docs_1, dictionary = dictionary_1, coherence = "c_v")
model_coherence_score_1 = model_coherence.get_coherence()
print("Coherence Score:", model_coherence_score_1)
```

```
Coherence Score: 0.4582452558920426
```

MODEL 2:

```
model_coherence = CoherenceModel(model=model_2, texts= docs_2, dictionary = dictionary_2, coherence = "c_v")
model_coherence_score_2 = model_coherence.get_coherence()
print("Coherence Score:", model_coherence_score_2)
```

```
Coherence Score: 0.45517656517114774
```

We can observe that both models have very similar coherence scores.