Comparison of Transformer and non-Transformer Models for a Text Classification Task

A Machine Learning Approach to find when not to use a Transformer

Oriol Garrobé Guilera, origa255
MSc Statistics and Machine Learning

Abstract

***Abstract*—The Natural Language Processing (NLP) field is changing with the introduction of the Transformer in 2017. The Transformer has replaced many Machine Learning (ML) methods achieving state-of-the-art results. However, in some particular cases such complex and computationally expensive models achieve results very similar to other, simpler, models.**

From this regards, 3 different ML approaches are trained on a dataset that provides information of Headlines and short descriptions of news articles from the Huffington Post and classifies them between different categories, such as Politics or Entertainment, among others. In this scenario non-Transformer models achieve similar results to the Transformer models for large amounts of data while needing much less training time. On the other hand, Transformer models achieve substantially better results for more difficult tasks and small amounts of data. All this results are thoroughly analysed, providing a clear image of when to use a Transformer-based model.

I. Introduction

SINCE the introduction of the Transformer in 2017 (Vaswani et al.) [1] many NLP practitioners have changed their previous go-to methods to Transformer-based models. The Transformer, in fact, achieves state-of-the-art results in most of the tasks, but everything comes at a cost. This models are very computationally expensive and memory demanding (Strubell et al.) [2]. From this point, in many cases, using other – simpler - models could be a better decision.

The aim of this project is to draw a line between those tasks where a Transformer is advisable and those where it is not. Although this border can be blurry and dynamic, this study tries to provide a picture on when it is clearly necessary a Transformer model and when it is not.

Many studies have shown this concern about the overuse of too complex models for relatively simple tasks. More in particular, (Adhikari et al.) [3] showed that some ML models can surpass Neural Networks (NNs) on many tasks. Also, transfer learning techniques, such as Knowledge Distillation proved that simple ML models can learn from more complex models achieving state-of-the-art results with less resources (Adhikari et al.)[4]. This project will follow this trend but only focusing the study in text classification tasks.

The work will be structured as follows. A brief theoretical explanation of the models and metrics used, followed by a description of the method. All the data, pre-processing steps, feature extraction techniques and classifiers that have been used along the process will be explained. This detailed description of the work should allow the reader to replicate the experiments obtaining similar results. Finally, the results will be provided and analysed while future work ideas will be proposed.

II. Theoretical Background

A. Text Representations

Machines cannot understand words as humans do, therefore it is necessary to convert our natural language to a machine-friendly representation. To do so, the unstructured text documents are numerically represented by means of the following text representations.

1) **Count Vectors:** A count vectorizer is based on term frequencies. It generates a vector of word counts for each document and stores them in a matrix where each column is a word and each row is a document. Usually the word counts are normalized. It does not provide any ordering information of the original sentence – only term frequencies are reflected in count vectors.

2) **TF-IDF Vectors:** Term Frequency – Inverse Document Frequency (TF-IDF), generates a vector of words assuming an inverse relationship between a term's relevance and its frequency. TF assigns high weights to words that are supposed to be good representatives of the document. In a similar way, IDF assigns low weights to words that are present in many documents. The idea is that the words that appear a lot in a document but not much in other documents are good representatives of the document.

3) **Word Embeddings:** Word Embeddings are a featured low-dimensional distributed representations that capture the semantics of a text. Words are mapped into a vector space, where words with similar meaning are close in the spatial representation. It is up to the model to decide which features are the best ones to represent the word vectors.

B. Models for text classification

1) **Logistic Regression/Multinomial Logistic Regression:** The Logistic Regression model (LR) and Multinomial Logistic Regrerssion (MLR) are based on the logistic function (sigmoid) which is an S-shaped curve that takes any real number and maps it to a value between 0 and 1. From this point, it is possible to predict the probabilities of the different possible outcomes and segregate data points in a number of classes. The regression coefficients (betas) of the Logistic Regression are estimated by training the model.

2) **Support Vector Machines:** Support Vector Machines (SVM) are a supervised ML algorithm mostly used for classification tasks. The idea behind SVM is to find an optimal hyperplane that separates all classes. This hyperplane maximizes the distance to the support vectors – closest observations to the frontier - which is called margin. To separate classes in a high-dimensional space, SVMs use the Kernel Trick, where the input is converted to a higher dimension where it is possible to segregate the points. Depending on which side of the hyperplane observations belong they are labelled as on class or another.

3) **BERT:** The Bidirectional Encoder Representations from Transformers (BERT) is a Google open-source deep pre-trained language model. BERT takes full advantage of the bidirectional information of text sequences. Also, BERT is pre-trained using two techniques. First, Masked Language Modelling where the network is trained to predict half of the words in the training corpora that are randomly masked. Second, Next Sentence Prediction, where the network is trained to predict whether two given sentences are coherent together. BERT can be fine-tuned with a relatively small amount of task-specific data.

BERT's architecture is composed of stacked encoders and decoders that use Attention mechanisms to assign different weights to parts of the input based on their significance. Attention makes the network focus on specific data points, which is especially useful in order to learn about the context of words. In particular, the 'bert-base-uncased' model (Devlin et al.) [5] used has 12 self-attention layers, and the length of the embeddings is 768. Overall, it has 110M parameters.

C. Performance Measures

1) **Precision:** Precision stands for the number of properly predicted observations in the positive class over all the observations. In other words, it is a ratio of what the model predicted correctly.

2) **Recall:** Otherwise called sensitivity, stands for the correctly predicted positive observations over all observations in the actual positive class. This metric checks how many properly predicted points are predicted within a class.

3) **F1-score:** This metric is a weighted average of precision and recall. From this point, it takes both false positives and false negatives into account, it is very useful when the goal is to predict unbalanced classes. In this case, as classes are unbalanced, this metric is the one that mirrors the quality of the model best.

III. Data Exploration

This project is based on a dataset from Kaggle [6]. The dataset is called News Category Dataset [7] and contains 202,372 records from the year 2012 to 2018 obtained from the HuffPost [8].

Each record contains following attributes:

1) category: Golden label of the category the article belongs to.
2) headline: Headline of the article.
3) authors: Person that authored the article.
4) link: Link to the post.
5) short description: Brief description of the article.
6) date: Date when the article was published.

In particular, there are 41 different categories and only the 6 largest are used. The classes are imbalanced as can be seen in Figure 1. The largest category is Politics with 32739 observations, and the smallest is Parenting, being the former approximately 4 times larger than the latter. Samples from the dataset, each with different number of classes to predict, are used for this study. Also, different sample sizes are used for training and 2000 observations are used for testing.

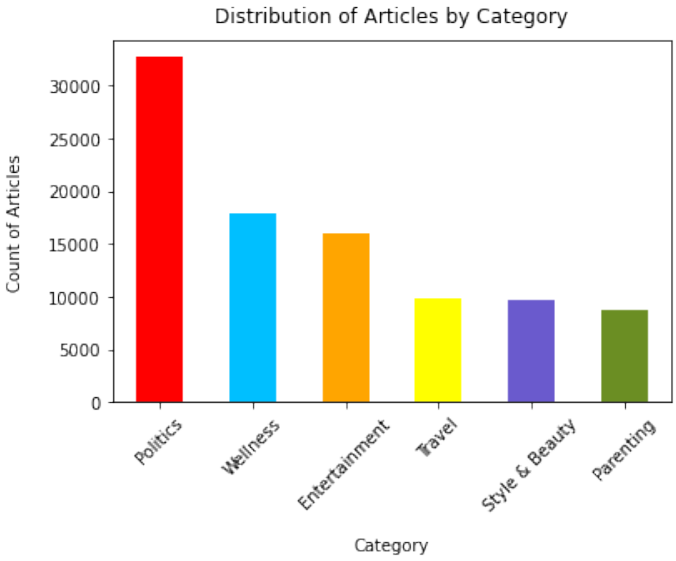

Fig. 1: Barplot of the number of articles of the 6 largest categories in the HuffPost dataset.

IV. Method

A. Data Pre-processing

The first step of the project is to pre-process the data. The dataset contains 41 different categories and only the 6th

largest are selected for the study. There are 6 attributes for each observation, only 3 are relevant: 'headline' and 'short description' as the text fields, and 'category' as the gold label. The two text fields are combined into one. The 'category' field is factorized, which means that every class is assigned to a numeric value.

B. Tuning Hyperparameters

1) **Non-transformer models.:** LR and SVM are good baselines for text classification when using the appropriate text representation. Count Vectors and TF-IDF vectors are commonly used for this models. Stop Words are removed and training the models with default parameters the TF-IDF yields the best results. The text representation – a sparse matrix - contains both unigrams and bigrams - single words and pairs of words -, and all the values that are 0 are set to 1 for computational purposes. From this point and using the TF-IDF text representation we explore both LR and SVM to find the optimal hyperparameters by means of a simple grid search. LR performs best when balancing the data and using the 'newton-cg' solver – the algorithm to use in the optimization problem - with an 'l2' regularization. SVM performs best using an 'adaptive' learning rate balancing the data and the 'l2' regularization.

2) **Transformer models.:** The text representation used for this model is the Word Embedding. The libraries used take care of this step of the process. At this point, the pre-trained 'bert-base-cased' model is used - a pretrained model on English language that does not make a difference between Uppercase and lowercase letters-. As the model is pretrained, training it for 2 epochs with the default parameters it manages to perform the task.

C. Comparison

Once the optimal models are found, the main part of the method is performed. The idea is to compare Transformer models to non-Transformer models. In this part then, the 3 models are trained and tested in different samples of the dataset. In particular they are trained in 40 different samples. The samples contain observations with different number of classes, from 2 classes to 6. Different sizes of samples for each classification task are used, in particular the following sizes : 1000, 2000, 4000, 6000, 10000, 20000 and 40000. This will provide an image of how different models perform in different scenarios.

The source code of the project can be found on Github[1].

V. Results

In Table I it can be seen the F1-score of each model for the test data. The columns are the sample size and the rows are the number of categories that the model has to classify for each sample of the dataset. Although precision and recall were computed, only the F1-score is displayed. For every case the F1-score yields a result between precision and recall, therefore it is the measure that mirrors best the results obtained.

LR and SVM report results for this models trained on a TF-IDF text representation. All the models share the same hyperparameters. Bert shows the results obtained training the pre-trained 'bert-base-uncased' model for two epochs on default parameters. Tuning hyperparameters is not done due to computational resources and assumed not necessary due the fact that the results are already better than the non-transformer model baselines.

Figure 3 displays the time necessary to train the different models for each sample size. It can be seen that non-transformer models do not need long training times, and they barely increase when increasing the amount of data. On the other hand, pre-trained Bert model needs longer training times and they increase almost linearly with the data.

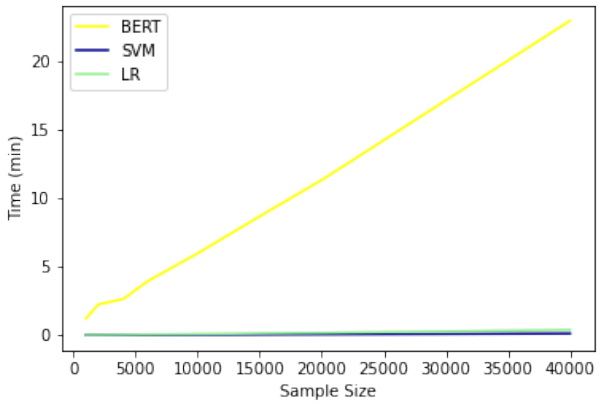

Fig. 2: Comparison between training times for the different models.

It can be seen that the F1-score of each model increases with the sample size, more data means better results. Bert does not change dramatically its performance with more data. BERT improves 4.8 points on average from the smallest sample to the largest. LR and SVM display noticeable better results with the increase of data, they improve an average of 10.6 and 10.4 points from the smallest sample to the largest, achieving similar results to BERT for large amounts of data.

It is also important to point out that the F1-score for the models gets worse as the number of categories increases, the task becomes more difficult. LR and SVM can worsen their performance 21 and 24 points respectively for small data sizes but only 10 points for larger amounts. Bert has a more robust performance in this sense, its performance gets worse 13 points in the smallest sample and 6 points in the largest.

[1]https://github.com/oriolgarrobe/Text-classification

Size / Class	1000			2000			4000			6000			10000			20000			40000		
	LR	**SVM**	**Bert**	**LR**	**SVM**	**Bert**	**LR**	**SVM**	**Bert**	**LR**	**SVM**	**Bert**	**LR**	**SVM**	**Bert**	**LR**	**SVM**	**Bert**	**LR**	**SVM**	**Bert**
2	0.91	0.93	0.97	0.94	0.95	0.97	0.94	0.95	0.98	0.95	0.96	0.98	0.95	0.96	0.98	0.96	0.96	0.98	0.96	0.96	0.98
3	0.85	0.85	0.92	0.88	0.87	0.93	0.89	0.90	0.94	0.90	0.89	0.95	0.91	0.93	0.95	0.92	0.93	0.96	0.92	0.92	0.97
4	0.79	0.81	0.90	0.83	0.84	0.90	0.85	0.86	0.92	0.86	0.88	0.93	0.86	0.89	0.94	0.90	0.90	0.94	0.91	0.90	0.95
5	0.74	0.73	0.87	0.81	0.81	0.89	0.81	0.85	0.91	0.84	0.84	0.92	0.84	0.88	0.93	0.86	0.89	0.94	0.89	0.89	0.94
6	0.70	0.69	0.84	0.74	0.75	0.87	0.79	0.79	0.88	0.80	0.81	0.88	0.82	0.84	0.89	0.81	0.84	0.91	0.85	0.86	0.92

TABLE I: Results of the F1-score for each model on the test set. Columns refer to sample size. Rows refer to number of classes to classify.

Table II emphasizes the comparison of performance and training time between non-transformer models and Bert. It shows the percentage of F1-score achieved and the training time necessary relative to Bert. LR and SVM achieve 88 and 89% of BERT's performance using less than 1% of the training time in the worst case. Also, they perform almost as good as BERT with the increase of data, 95% as good, and they only need 1.7 and 0.5% of the training time needed for BERT.

Model /	LR		SVM	
Sample	**F1**	**Time**	**F1**	**Time**
1000	88%	0.7%	89%	0.4%
2000	92%	0.7%	93%	0.4%
4000	92%	1.1%	94%	0.5%
6000	94%	1.3%	95%	0.5%
10000	93%	1.3%	96%	0.5%
20000	94%	1.5%	96%	0.5%
40000	95%	1.7%	95%	0.5%

TABLE II: Comparison between non-transformer based models and Bert.

A more graphical representation of this results can be seen in the Appendix A.

VI. Discussion

As said previously the aim of this project was to provide a clear image of performance against resources required for non-transformer and transformer based models. Many results have been provided and from them it is possible to learn some insights.

It is clear from the results provided that the Transformer outperforms LR and SVMs in all the tasks. However, this study was focused in the fact that some models can be almost as good as the Transformer in some particular cases while needing less training time and computational resources. From this regards, the results showed that in the limit case - the Transformer is furthest in performance - ML models are only 10% worse. In some cases they are only 5% worse. This is due the amount of data to train. The difference is larger with less data because the Transformer is pre-trained, which means that is a model that has been trained in huge amounts of data for long time. If the Transformer model was not pretrained, it would achieve worse results than the ML models due the fact that the Transformer is formed by a number of internal NNs, and these require large amounts of data to train its weights and biases. Training a model from scratch - where the weights and biases would be initialized at random - was not used due to limited computational resources available for the study.

When it comes to the resources needed, many studies has shown that that the Transformer is much more computationally expensive and memory demanding than other ML models. This study has proven the former but not the latter. In particular the 3 models were trained on the Google Colab Environment with enabled GPUs. LR and SVM did not require the GPUs and were trained using the CPUs that Colab provides. Even with this, slower, processing units, LR and SVM require on average 1% of the time to train. Also, they do not increase such time with the amount of data, achieving for the largest datasets, as seen above, similar results to BERT. The Transformer on the other side, increases the training time proportionally to the amount of data. All this facts suggest that with the increase of data LR and SVMs are reliable models and much cheaper than the Transformer. It is seen then that depending on the case, LR and SVMs could be more beneficial than the Transformer.

VII. Conclusion and Future work

The main purpose of the project, finding a border between models, was achieved. However, there are some thoughts that are worth mentioning.

The initial idea when the project started was that for small amounts of data LR and SVMs would perform better than the Transformer. This thought was founded on previous works where this models outperformed NNs. The reason of this is that NNs require large amount of data to train, and Transformers are founded on stacked NNs. From this point, this hypothesis was contradicted due the fact that a pre-trained model was used. A future step on this research would be to train a Transformer model from scratch, with random initialization of weights and biases and compare its performance and training time to LR and SVMs. This should provide positive results for the simpler ML models.

Due to limited computational resources, optimal parameters were found for each model on a single task and with all the data available. From this point, this parameters were applied for all the different data samples. It could be a better approach to find optimal parameters for each task proposed to study. This would provide finer results of the limits of this models.

Finally, it is not clear why LR and SVM decrease their performance with the increase of categories to segregate that much, especially when in some cases classifying a category is close to a word spotting task. This could be due to the regularization technique used. It could also be due the fact that some categories overlap, which means that some categories are very close to each other and use very similar texts, one example could be the classes 'wellness and beauty'. This is object of further study.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, cite arxiv:1706.03762Comment: 15 pages, 5 figures, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762.

[2] E. Strubell, A. Ganesh, and A. McCallum, "Energy and policy considerations for deep learning in NLP," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 3645–3650. DOI: 10.18653/v1/P19-1355. [Online]. Available: https://www.aclweb.org/anthology/P19-1355.

[3] A. Adhikari, A. Ram, R. Tang, and J. Lin, "Rethinking complex neural network architectures for document classification," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4046–4051. DOI: 10.18653/v1/N19-1408. [Online]. Available: https://www.aclweb.org/anthology/N19-1408.

[4] A. Adhikari, A. Ram, R. Tang, W. L. Hamilton, and J. Lin, "Exploring the limits of simple learners in knowledge distillation for document classification with DocBERT," in *Proceedings of the 5th Workshop on Representation Learning for NLP*, Online: Association for Computational Linguistics, Jul. 2020, pp. 72–77. DOI: 10.18653/v1/2020.repl4nlp-1.10. [Online]. Available: https://www.aclweb.org/anthology/2020.repl4nlp-1.10.

[5] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, Minneapolis, Minnesota: Association for Computational Linguistics, Jun. 2019, pp. 4171–4186. DOI: 10.18653/v1/N19-1423. [Online]. Available: https://www.aclweb.org/anthology/N19-1423.

[6] *Kaggle*, https://www.kaggle.com.

[7] R. Misra, *News category dataset*, Jun. 2018. DOI: 10.13140/RG.2.2.20331.18729.

[8] *Huffpost*, https://www.huffpost.com/.

APPENDIX A

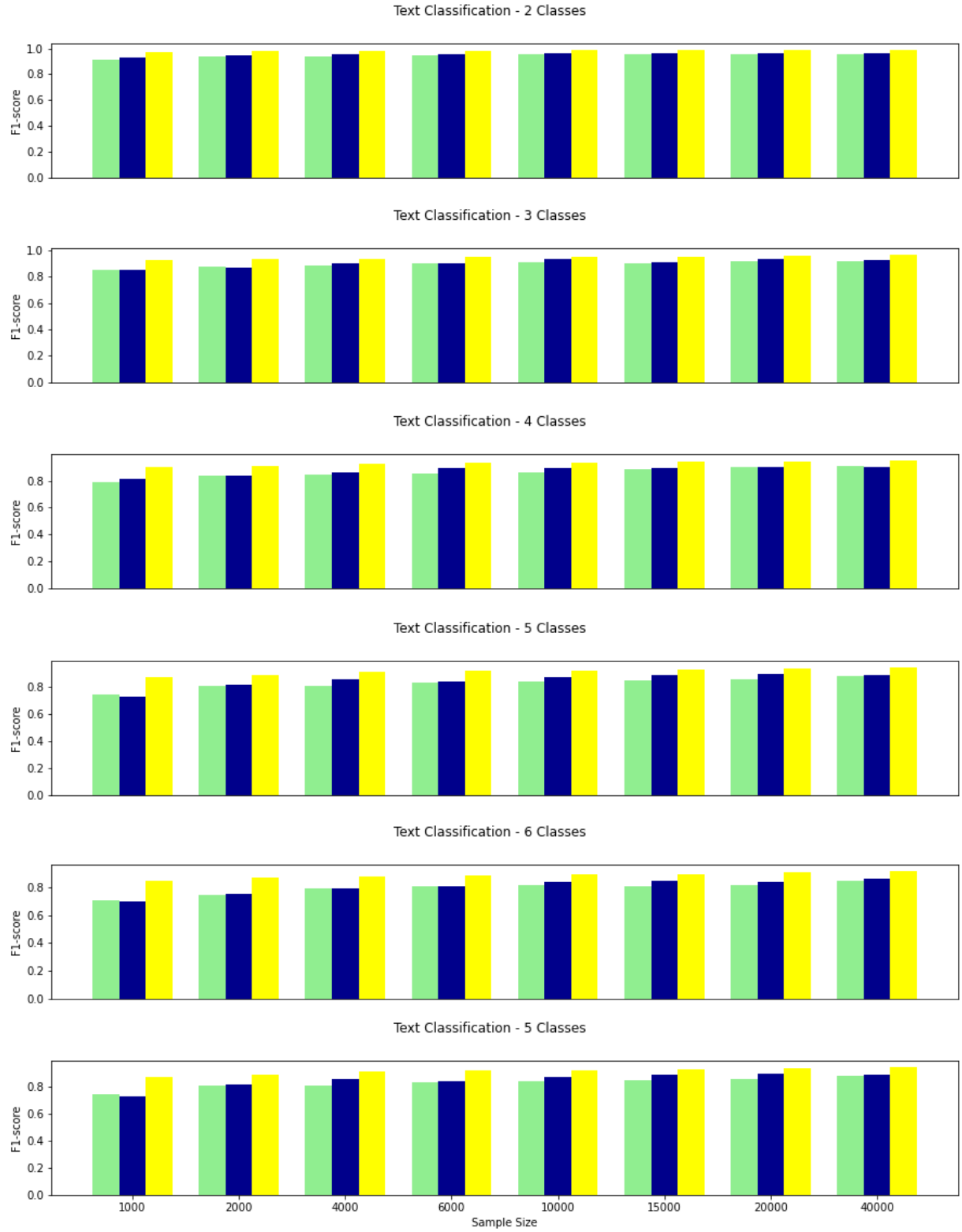

Fig. 3: Comparison of F1-score between models for different amount of categories to classify and sample sizes.