Tweet Sentiment Classification using BERT Models and Ensembles

Markus Pobitzer*, Lucas Dodgson*, Lukas Mautner*, and Pierre Dorge

"The Transponsters", Department of Computer Science, ETH Zürich, Switzerland

***Abstract*—Sentiment classification is about the categorization of natural language by its underlying attitude. In this work, we present a number of approaches to solve this task specifically for tweet messages. Mainly, we explore different deep learning models, among them state-of-the-art Transformers and their combinations in different ensemble methods. Our best model was an ensemble of Transformer-based architectures and achieved a validation accuracy of 0.922.**

I. Introduction

The ability to understand the underlying sentiment in a person's speech is not only useful in daily social interactions, but also allows us to evaluate the things we offer – such as products, services or social policies. In the past decade, social media has been established as one of he leading means of large-scale human communication – Twitter being on the forefront of this revolution [1]. This is just as true for opinions on goods and services as it is for personal exchange. It is clear when considering these facts that it has become ever more important for companies and governments to be able to efficiently extract insights about the reception of their own products or policies from these large bodies of public text.

In this work, we present a general-purpose approach to the classification of tweets (Twitter posts) according to their sentiment. We do this through the use of state-of-the-art (SOTA), fine-tuned large language models (LLMs). By training our models on a dataset that is not restricted to any specific topic, we retain the option to filter tweets and similar messages by use cases later and then analyse the overall sentiment towards these use cases.

Our contributions are the following:

1) A comprehensive evaluation of various basic methods and SOTA pre-trained models for the given task.
2) An analysis of the effectiveness of three different ensemble approaches.

II. Related Work

The field of natural language processing (NLP) has gained popularity in recent years as the feasibility of large language models (LLMs) has significantly increased due to the rapid growth in commercially available GPU processing power [2].

LLMs refer to a class of deep learning models such as OpenAI's GPT-3 [3] that attempt to solve the tasks of natural language comprehension and transformation. Recently, the majority of noteworthy breakthroughs in this field has been based on the Transformer architecture [4]. Specifically, the BERT platform [5] is responsible for a plethora of SOTA language models, such as RoBERTa [6] and DistilBERT [7] (see [8] for more).

While LLMs can in principle be trained to handle almost any type of language and content (even programming languages; see [9]), we typically do not train them from scratch. Instead, we use a technique known as transfer learning or few-shot learning (see [10]) in order to adapt the publicly available pre-trained LLMs to our problem. We take the aforementioned pre-trained models from Huggingface.

III. Our Method

We used a variety of methods and models in our attempt to solve the sentiment analysis problem. In this section, we lead through the various explored approaches and outline our path towards the best-performing method.

See section IV for a quantitative comparison and discussion of each model's performance.

Dataset: Our dataset consists of 2.5 million tweets which are split by sentiment: either positive or negative (1.25 million each). Each tweet was sanitized, replacing all personal information and hyperlinks with standardized tags. All text was also converted to be lowercase.

This dataset provided multiple challenges. Firstly, it is a small dataset for LLMs[1]. Secondly, the language used in tweets is often very informal and may contain spelling errors, thus many pre-trained word embeddings may not know what to do with those words. Thirdly, the fact that casing was removed also means that valuable information has been lost. Finally, it contains certain tweets multiple times. We found removing the duplicates to not bring any advantage. Thus, for the remainder of this work we work with the unmodified dataset. The evaluation of the effect of the duplicates is in section VI-A in the appendix.

*Equal contribution

[1]For example, the BERTweet model was trained on over 850 million tweets [11].

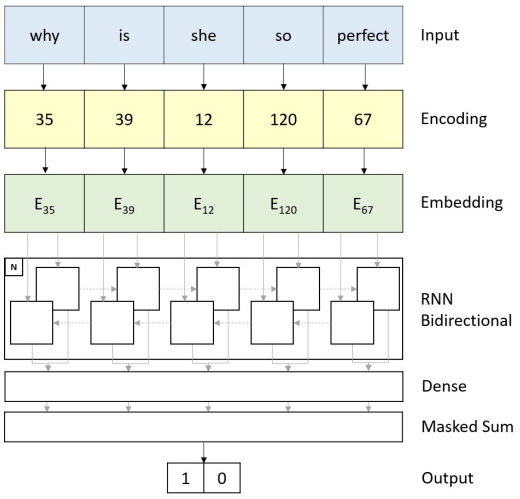

Figure 1. Overview of our RNN pipeline. Several RNN cells can be stacked after each other, denoted here by N. In our final model we set $N = 2$.

A. Baselines

We first attempted to solve the problem at hand with well-known and rather dated NLP approaches. In particular, we started off with bag-of-words and term frequency-inverse document frequency based approaches, where we used either Support Vector Machines or Logistic Regression. We also evaluated GloVe [12] and FastText embeddings. In comparison to models we explore in the following sections, the performance of these approaches is inconsequential, thus we omit further detail about these here and instead refer the reader to section VI-C in the appendix.

B. Recurrent Neural Networks

Our second set of approaches is based on recurrent neural networks (RNNs). We apply the recurrent nature of the RNN cell on an embedding of the input sentence to learn the semantic meaning of the words and their relationships to each other in the sentence. An overview of the architecture is shown in figure 1. Again, we found the performance of these approaches to be inconsequential, so we refer the reader to section VI-B in the appendix for an in depth discussion of our architecture and its performance.

C. Transformers

As Transformer-based architectures have dominated NLP – among many other areas of deep learning – in recent years, we explored their performance on our particular task as well.

One of the main challenges with this type of model is that it is extremely data-hungry. As such, we apply transfer learning to adapt and fine-tune pre-trained models to the task at hand. To do this, we compare the performance of a large variety of different pre-trained Transformer models on our task. We selected these models based on the following criteria: pre-trained on English text, uncased (agnostic to capitalization) and pre-trained on short and informal pieces of text.

We list the selected Transformer models here:

- Bert base model [13]
- DistilBERT base model [14]
- DistilBERT SST 2 (DistilBERT fine-tuned on SST-2)
- BERTweet Base (Based on RoBERTa [6]) [11]
- BERTweet Large (Based on RoBERTa [6]) [11]
- Bert Twitter Roberta Base [15]
- XLNet Base[16]
- Bart Base [17]
- XtremeDistilTransformers (l6-h256 checkpoint) [18]

The *BERTweet* and *Twitter RoBERTa* models deserve a special mention here, due to the datasets they were pre-trained on: The language is English and the contents are actual tweets. Consequently, they do not need to re-learn embeddings or semantic structures, but can be trained for our specific task quite effectively, using only a few epochs on a relatively small dataset.

D. Ensembles

The main idea behind the widely successful approach of ensemble learning is to increase generalization through the combination of multiple so-called base models, all of which must be able to solve the learning task on their own [19]. By combining their subsequently meaningful individual predictions, weak points of the base models may be "evened out". We evaluated three such approaches.

1) Majority Voting: In the most straightforward ensemble method for binary classification, one takes an odd number of independently trained models (we use 3 in our experiments) and lets them vote on what the output should be for each data point. Due to the odd number, there will always be a winner.

In order to determine interesting combinations of models, we measured the agreement between 8 of our best-performing base models (see figure 2). Our thinking is that less overall agreement should point to weak points in the participating models, which may be patched up by combining them. The reader is referred to section IV-E for the performance results of 8 such interesting combinations and to section VI-H in the appendix for the exact reasoning behind their selection.

2) Bagging: Bagging (short for "bootstrap aggregating") [20] is a special form of the majority voting approach discussed in section III-D1. While we talk about combining *different model types* in the previous section, bagging deals with combinations of *multiple models of the same type* that are trained on varying datasets. These variations of the original dataset are obtained through bootstrap sampling [21].

We selected the *DistilBERT SST* [14] and *XtremeDistilTransformers* [18] models for our bagging experiments. We refrained from training more models since the results

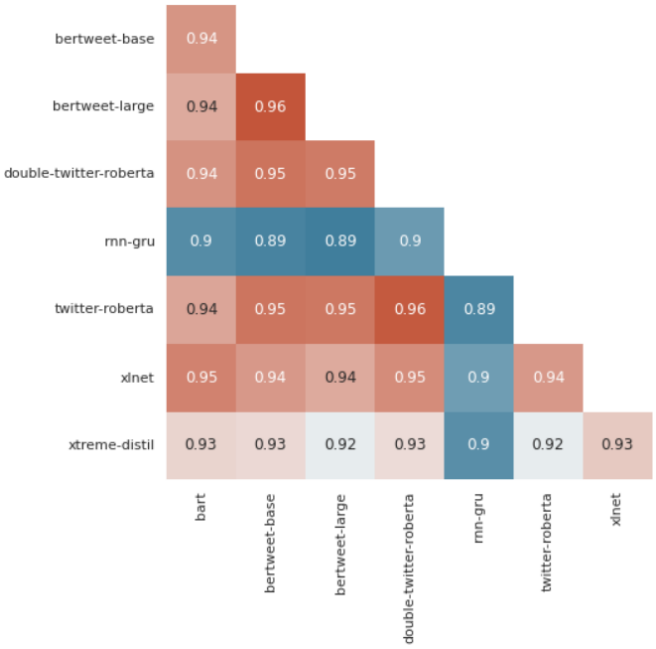

Figure 2. Agreement between a recent version of our highest-performing base models. Each score is the percentage $p_{i,j} \in [0, 1]$ of test-set predictions that the models i and j agree on.

on these two models suggest that bootstrapping might be harmful rather than beneficial in this context (see section IV-F).

3) Stacking: Stacking describes a paradigm of learning that concerns itself with the intelligent combination of outputs of pre-trained predictors, rather than with the act of prediction itself [22]. To this end, it utilizes a so-called "meta model", which is trained to combine the outputs of multiple base models into the correct final output in a supervised manner.

We implemented a variation of stacking that trains the stack of base models and meta model end-to-end, rather than separately. For the meta model, we used a simple MLP with only 1 hidden layer.

IV. Evaluation & Discussion

A. Overview

Model(s)	Accuracy ↑	Kaggle ↑	Main technique
Ensemble of 4 models	0.922	0.922	Stacking
BERTweet	0.920	0.920	Transformers
BERTweet B&L, RNN	-	0.920	Majority Voting
3x DistilBERT SST 2	-	0.896	Bagging
RNN	0.876	0.874	GRU
TF-IDF SVM	0.855	-	Simple Baselines
Official Baseline	-	0.804	-

Table I
Results of our best performing models. Rounded accuracies from both a validation split and the Kaggle leaderboard.

Table IV-A contains the scores of the best-performing instances of all the different methods described in section III. We list their respective performances both on a validation split of 0.5% (if applicable) and the *public* Kaggle leaderboard.

It is worth noting that we compare the methods' performances on their validation accuracies wherever possible, as we know the validation set to be balanced. By contrast, we do not know anything about the exact structure of the Kaggle public test set, which could be heavily biased.

B. Non-Transformers

1) Baselines: All our non-SOTA baselines are separated from our top-performing models by at least one major breakthrough in the field of NLP. For this reason, we believe that the differences in performance need no further explanation.

2) RNN: The RNN model introduced in III-B is not comparable with transformer-based approaches. However, with a bit of hyper-parameter tuning, we were able to achieve a decent accuracy (see VI-B).

C. Transformer-Based Models

Model	Accuracy ↑	Kaggle ↑
BERTweet Large	0.920	0.920
BERTweet Base	0.917	0.917
BERT Twitter RoBERTa Base	0.907	0.907
BERT base	0.903	0.902
XLNet Base	0.901	0.899
DistilBERT base 2	0.899	0.892
DistilBERT SST 2	0.898	0.898
Bart Base	0.895	0.893
XtremeDistil Transformers	0.893	0.884

Table II
Results of the evaluated Transformer-based models.

Transformer models pose a significant problem in training: They require an enormous amount of resources, even to merely fine-tune. For this reason, we were not able to run exhaustive hyper-parameter tuning for them. Instead, we trained each of them with only slightly varying learning rates (in the order of 10^{-6}), usually for 24 hours.

Despite these limitations, the strength of Transformer-based architectures is clear. They vastly outperform all of our baselines without any hyper-parameter tuning. Table IV-C contains the performance of all the evaluated Transformer-based models. Most of these observations are in line with our expectations; models trained on tweet data performed significantly better than those that were not. The one surprise was that despite being a cased model[2], XLNet performed very well.

[2]A cased model is not agnostic to capitalization. This could be problematic here since all text in the dataset is lower-case, which means that any information that stemmed from capitalization was lost and the model cannot utilize it.

D. BERTweet

The most important aspect of fine tuning BERTweet is the learning rate, which is far smaller than when training our RNN models. For a comparison of the learning rates on the Base model that worked best for us, see table VI-E in the appendix. A learning rate of $3 \cdot 10^{-6}$ showed the best result for the validation set; a fact which is in line with the findings in [23]. The differences seem small but were significant for the Kaggle leaderboard.

This became even more relevant when fine tuning the *BERTweet Large* model. Due to its long training time, we were not able to make a full ablation study for it. An evaluation of training such a model for several epochs can be found in section VI-D in the appendix.

E. Majority Voting

Combined Models	Kaggle ↑
bertweet-base, bertweet-large, rnn-gru	0.920
bart, bertweet-large, double-twitter-roberta	0.918
bertweet-base, bertweet-large, twitter-roberta	0.918
all MV bases except RNN	0.917
bart, bertweet-large, xlnet	0.914
bart, twitter-roberta, xlnet	0.913
bart, bertweet-large, xtreme-distil	0.912
bertweet-large, rnn-gru, xtreme-distil	0.909

Table III
RESULTS OF ALL SELECTED MAJORITY-VOTING COMBINATIONS.

Table IV-E contains the public Kaggle scores for the majority votes of the 8 most interesting combinations of top-performing base models. See sections III-D1 and VI-H for further information on their selection.

It is clearly visible that all combinations achieve rather high scores – even the ones containing the RNN model, which on its own only reaches validation accuracies of 0.876. This seems to confirm our theory that disagreement between models may be used advantageously.

However, it must also be noted that none of these combinations outperform the best base model (*BERTweet Large*) in its properly fine-tuned state. Also, seeing as the top majority-voting combination is between *BERTweet Large*, *BERTweet Base* and *RNN GRU*, the former two of which agree quite heavily, it is plausible that they simply overpowered the RNN in the voting process. This might additionally explain the similarity between the performances of this particular combination and *BERTweet Large* on its own.

Consequently, we must conclude that majority voting between different models does not seem to show any distinct advantages when compared to a SOTA BERT model on its own.

F. Bagging

For both base-model types outlined in section III-D2, we trained 3 base models on bootstrapped datasets for 5 epochs. The *DistilBERT SST 2* models reached a Kaggle score of 0.896 when combined, while the *XtremeDistilTransformers* models reached 0.880. Since neither of these scores surpass the base models' individual performances (0.8977 and 0.8836 respectively), we chose not to pursue bagging any further in the solution of this problem.

We hypothesize that the poor performance stems from the fact that bootstrapped datasets may be prone to imbalance, which would lead to an ensemble of too-heavily biased base models.

One possible way to alleviate this might be to train more classifiers in the hopes of balancing out the introduced biases in the final ensemble. Another approach would be to ensure balance in the bootstrapped datasets through stratified sampling [24] with repetition.

G. Stacking

We tried three different combinations for our variation on stacking. The first two were combinations of *XtremeDistilTransformers* and *XLNet* with *Bert Twitter Roberta Base*. But these did not show any improvements compared to the individual models, with validation accuracies of 0.904 and 0.898. Our final combination and also our final model combined four different models: *XLNet Base*, *BERTweet Large*, *XtremeDistilTransformers* and *DistilBERT Base*. This reached both a validation accuracy and a public Kaggle score of 0.922.

While this was our best performing model, two major caveats apply: We trained this model for almost 3 days and the performance benefits were very minor. Thus, we conclude that stacking on this task brings only small improvements.

V. SUMMARY

In this work we compared a multitude of deep learning approaches – both base models and ensembles – as applied to binary sentiment classification. It is clear from our analysis that Transformer-based architectures outperform other approaches by a significant margin. Interestingly, we found that they not only perform well compared to other models and learning paradigms (such as SVMs), but also compared to most ensemble methods that we tried. Specifically, the method of stacking is the only ensemble method we tried that was actually competitive. However, it still only outperforms our best base model by a very small margin and takes a multiple of the computational resources required to train any of the base models.

As for future work: We think it would be beneficial to explore further combinations of models with some of our ensemble techniques, in the hopes of identifying a meaningful consensus that we did not achieve in this work. Furthermore, some of our ensemble techniques could be tweaked slightly in order to adapt them better to the problem at hand.

REFERENCES

[1] D. Sayce. (2020) The number of tweets per day in 2020. [Online]. Available: https://www.dsayce.com/social-media/tweets-day/

[2] N. Evanson. (2020) 25 years later: A brief analysis of gpu processing efficiency. [Online]. Available: https://www.techspot.com/article/2008-gpu-efficiency-historical-analysis/

[3] T. B. et al., "Language models are few-shot learners," 2020, https://arxiv.org/abs/2005.14165.

[4] A. V. et al., "Attention is all you need," 2017, https://arxiv.org/abs/1706.03762.

[5] J. D. et al., "Bert: Pre-training of deep bidirectional transformers for language understanding," 2018, https://arxiv.org/abs/1810.04805v2.

[6] Y. L. et al., "Roberta: A robustly optimized bert pretraining approach," 2019, https://arxiv.org/abs/1907.11692.

[7] V. S. et al., "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," 2020, https://arxiv.org/abs/1910.01108.

[8] F. A. et al., "Transformer models for text-based emotion detection: A review of bert-based approaches," 2021, https://www.researchgate.net/publication/348740926.

[9] B. B. et al., "Tfix: Learning to fix coding errors with a text-to-text transformer," 2021, http://proceedings.mlr.press/v139/berabi21a/berabi21a.pdf.

[10] F. Z. et al., "A comprehensive survey on transfer learning," 2020, https://arxiv.org/abs/1911.02685.

[11] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "BERTweet: A pre-trained language model for English Tweets," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 2020, pp. 9–14.

[12] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.

[13] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," *CoRR*, vol. abs/1810.04805, 2018. [Online]. Available: http://arxiv.org/abs/1810.04805

[14] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *ArXiv*, vol. abs/1910.01108, 2019.

[15] D. Loureiro, F. Barbieri, L. Neves, L. E. Anke, and J. Camacho-Collados, "Timelms: Diachronic language models from twitter," *arXiv preprint arXiv:2202.03829*, 2022.

[16] Z. Yang, Z. Dai, Y. Yang, J. G. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *CoRR*, vol. abs/1906.08237, 2019. [Online]. Available: http://arxiv.org/abs/1906.08237

[17] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *CoRR*, vol. abs/1910.13461, 2019. [Online]. Available: http://arxiv.org/abs/1910.13461

[18] S. Mukherjee, A. H. Awadallah, and J. Gao, "Xtremedistiltransformers: Task transfer for task-agnostic distillation," 2021.

[19] L. Rokach, "Ensemble-based classifiers," 2009. [Online]. Available: https://link.springer.com/article/10.1007/s10462-009-9124-7

[20] L. Breiman, "Bagging predictors," 1996. [Online]. Available: https://link.springer.com/article/10.1007/BF00058655

[21] R. T. Bradley Efron, *An Introduction to the Bootstrap*. CRC Press, 1994.

[22] D. Wolpert, "Stacked generalization," 1992. [Online]. Available: http://www.machine-learning.martinsewell.com/ensembles/stacking/Wolpert1992.pdf

[23] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *China national conference on Chinese computational linguistics*. Springer, 2019, pp. 194–206.

[24] ScienceDirect. (2011) Stratified sampling. [Online]. Available: https://www.sciencedirect.com/topics/mathematics/stratified-sampling

[25] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv preprint arXiv:1409.1259*, 2014.

[26] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[27] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

VI. Appendix

A. Effect of Duplicate Tweets

Dataset	Validation Accuracy	Public test set accuracy
Full dataset	0.892	0.883
Partial dataset	0.881	0.886

Table IV
Comparison of the performance of the XtremeDistilTransformer model when training on the full dataset and when training on a cleaned dataset that doesn't contain duplicates.

To evaluate the effect that duplicate tweets in our dataset had on our models, we decided to run one of these models in the full dataset and in a dataset in which we had removed all the duplicate tweets (around 122000 positive and 107000 negative tweets). We run the following experiment: Using the *XtremeDistilTransofmers* model with the l6-h256 checkpoint [18], we run it on both the unmodified dataset and the one without duplicates. We use a validation split of 10% and let it run for 10 epochs. Table VI-A contains a comparison on both the validation set and on the public Kaggle score.

On the one hand we observe that in the validation split, the model trained on the full dataset performs better than the model trained on the dataset without duplicates. However, part of this effect can definitely be explained by the fact that with the duplicate tweets it can occur that the same tweet was in our training and in our validation set which would result in an increase in the performance. On the public test set accuracy, we observe a minor difference between the two models, with the partial dataset slightly outperforming the full one. But without knowing more about the makeup of the public test set and its correct labels, it's hard to make any decisions from this. In particular, the dataset without duplicates is no longer balanced between positive and negative tweets, which introduces a bias into our model. It is possible that it is this bias which actually results in the increase of the score on the public dataset. Due to this uncertainty and the minor differences in the performance of the model trained on the different datasets, we opted to work with the full and unmodified dataset for the remainder of our evaluation.

B. Recurrent Neural Networks

For many years, RNNs were amongst the best performing techniques for sentiment analysis. The advantages of RNNs are that they accept variable input length, leverage information over extended time intervals and are well studied [25], [26]. On the other hand, RNNs can face vanishing and exploding gradients that can make training unstable. To combat this problems, different extensions were proposed such as the Long Short-Term Memory (LSTM) [26] and Gated recurrent unit (GRU) [25] variants.

Our own RNN architecture is shown in figure 1 and outlined here. The first step is to represent every word in the sentence as a value. For this encoding step we use the tokenizer provided by Hugging Face, namely [15], which comes with a BERT tokenizer and was pre-trained on tweets. Afterwards, we train an embedding for the encoded words in the sentence and use this as the input for our RNN. Since the RNN computes an output for every embedded input word, we sum up all the logit outputs of a batch through a masked sum. A masked sum is used since we pad sentences in the encoding step to make batch processing easier.

In the final model we use a GRU or LSTM cell and stack it twice with a dropout probability of 0.2 in between (to combat overfitting). Additionally, we use a bidirectional RNN in the hopes of capturing non-sequential dependencies between the words.

We use a learning rate of 0.001 with Adam [27] as our optimizer. The full dataset with a validation split of 0.05 was used for training.

The RNN model is a lot smaller than the Transformer-based ones. Therefore, it was possible to train it completely from scratch in 5 hours on an NVIDIA GeForce RTX 2080 Ti. Considering that the model was not pretrained on millions of tweets, it achieved a reasonable accuracy.

Using a GRU or LSTM cell did not make any significant difference when it comes to the accuracy. The GRU cell reached the best validation accuracy after 6 epochs whereas the LSTM cell needed 8 epochs. Afterwards, they both started to overfit. Using only one cell – without stacking – did not significantly change the validation accuracy. On the validation split we were able to achieve an accuracy of 0.877 with a GRU cell and 0.876 with an LSTM cell.

C. Further Baselines

To have additional baselines to compare our model to, we also implemented and evaluated various other standard methods for text classification. In particular, we evaluated the following methods:

1) A logistic regressor using a bag of words vectorizer.
2) A logistic regressor using a Term Frequency and Inverse document Frequency (TF-IDF) based vectorizer.
3) A support vector machine (with both linear and non-linear kernels) using a TF-IDF based vectorizer.
4) A small neural network trained using either pretrained GloVe or FastText embeddings. For this, we tried out combinations in which the pretrained embeddings would also be trained or cases in which their embeddings were frozen.
5) A RNN using either pretrained GloVe or FastText embeddings. Same situation applies here again.

For each approach, we trained it using a 0.5% validation split, which we will use to compare the various models. We performed a large variety of different runs to tried and find

the best parameters for these. This did indeed increase the performance our models.

Even with the parameter tuning though, we found that they performed worse than Transformer-based models that were not tuned at all. In total we ran and logged over 1000 different runs with various parameters. These were logged using Weights and Biases, and more detailed information to each individual run can be found on https://wandb.ai/the-transponsters/CIL-2022Baselines.

Method	Validation Accuracy ↑
TF-IDF Non-Linear SVM	0.855
FastText with small NN	0.855
GloVe RNN	0.852
TF-IDF Linear SVM	0.835
TF-IDF logistic regression	0.826
Bag of Words logistic regression	0.822

Table V
RESULTS OF THE VARIOUS BASIC APPROACHES WE PERFORMED

Table VI-C contains an overview of the best performing variant of all each of the approaches. For both the Bag of Words and the TF-IDF based approaches, we performed extensive parameter searches. We did not perform this for the GloVe based approaches.

For the Bag of Words approach, we found that $10'000$ features performed the best. We observed the performance decreased if we removed stop-words. We also found that using sklearns cross validation performed better than a simple parameter search we performed.

For the TF-IDF logistic regression on the other than a minimum occurrence of 5 and a maximum occurrence of 60% performed the best, with a regularization constant of 2.0. Interestingly, varying the maximum frequency between 0.5 and 0.95 did not bring any major differences here for the validation accuracy.

For the Linear SVM, we again found a similar situation. A minimum occurrence of 15 performed the best, while varying the maximum between 0.6 and 0.95 brought no significant changes. Again, a regularization constant of 2.0 performed the best. For the non-linear SVM, we tried both using the Radial Basis Function (RBF) and a polynomial kernel. The RBF kernel performed better. Our best performing approach had a minimum occurrence of 5 and a maximum frequency of 0.5, again with a regularization constant of 2.0. Here, varying the maximum frequency had a significant effect. We also experienced significant over-fitting here, as for example in the best performing model the accuracy on the training set was 0.987.

Finally, for the GloVe versions, we unsurprisingly found the embedding pretrained on tweets to perform the best, where we further fine-tuned the embedding during the training. For all combinations, unfrozen embeddings performed better. While for the RNN, GloVe outperformed FastText, we found FastText to be better when trained using a small neural network.

D. Fine Tuning BERTweet for several epochs

Epoch	Validation Accuracy	Kaggle Leaderboard
1	0.917	0.915
2	0.918	0.914
3	0.913	**0.916**
4	0.915	0.914
5	**0.919**	0.914
6	0.917	0.910
7	0.913	0.908

Table VI
ROUNDED RESULTS OF LETTING BERTWEET LARGE FINE TUNE FOR SEVERAL EPOCHS.

The BERTweet models are memory intensive and especially *BERTweet Large* was challenging to fine tune. To fine tune a *BERTweet Large* model we used an NVIDIA GeForce RTX 2080 Ti so that the model would fit on the device. To find out if fine-tuning it for several epochs leads to an improvement of the accuracy, we let it run for 120 hours. We used a batch size of 8, a validation split of 20%, and a learning rate of $5 \cdot 10^{-6}$. Please note that we did not use any weight decay. An epoch of fine tuning took around 14 hours.

The results can be seen in Table VI-D. From this example we see that fine-tuning it for several epochs can lead to an improvement in accuracy on the validation set as well as on the Kaggle leaderboard. The accuracy on the Kaggle leaderboard does not seem to always follow the accuracy on the validation set, but without knowing the makeup of the public Kaggle leaderboard set, it is hard to reach any conclusions as to the reason for this. That being said, the difference in accuracy is not that big. By far the greater improvement regarding accuracy was achieved when we used a smaller learning rate instead of training it for more epochs.

E. BERTweet Learning Rates

Learning Rate	Test 1	Test 2	Val. 1	Val. 2
3e-06	0.917	**0.917**	**0.920**	**0.919**
2e-06	**0.918**	0.915	0.916	0.919
1e-06	0.914	0.915	0.912	0.916

Table VII
ACCURACIES (ROUNDED) OF DIFFERENT LEARNING RATES OF THE BERTWEET BASE MODEL CONSIDERING FIRST TWO EPOCHS. TEST CORRESPONDS TO THE ACCURACIES OF THE KAGGLE LEADERBOARD, VAL. TO THE ACCURACIES OF OUR VALIDATION SET. WE USED A BATCH SIZE OF 8, A VALIDATION SPLIT OF 0.1%, AND NO WEIGHT DECAY.

F. Summing up logits of different models

In our RNN approach we have seen that summing up the logits of each word worked. Therefore, we tried a similar approach with the output of different models that correspond to the whole sentence. We fine-tuned a *BERTweet Base* and a *Twitter-Roberta* model at the same time and summed up their outputs. After one epoch we achieved an accuracy of 0.915 on the Kaggle leaderboard, whereas a *BERTweet Base* model alone achieved an accuracy of 0.916 after one epoch of fine tuning. Considering the same hyper-parameters we did not notice any improvements.

G. Separate Models for Sentiment Classes

One idea to increase the accuracy was to use a model for each label. In this way a fine tuned model could better focus onto the assigned sentiment label. This means we fine tuned a pre-trained model for the positive sentiment and one for the negative at the same time. In our experiments we used the *Twitter-Roberta* model twice and achieved an accuracy of 0.906 on the Kaggle leaderboard after fine tuning it for one epoch. With the same hyper-parameters using only one Twitter-Roberta model we achieved an accuracy of 0.907. Therefore, this did not show any improvement. In addition, it took at least twice as long.

H. Combination Selection for Majority Voting

Note: For reasons of spacial efficiency, we will use the indices defined in table VI-H to refer to the 8 base models in table VI-H.

Model Name	Index
bart	1
bertweet-base	2
bertweet-large	3
double-twitter-roberta	4
rnn-gru	5
twitter-roberta	6
xlnet	7
xtreme-distil	8

Table VIII
INDICES FOR OUR MAJORITY-VOTING BASE MODELS.

In order to identify interesting combinations of three models each, we look to their pairwise agreements $p_{\cdot,\cdot}$ and compute a three-model agreement score as follows:

$$p_{i,j,k} = p_{i,j} \cdot p_{i,k} \cdot p_{j,k}. \tag{1}$$

We did this for all $\binom{8}{3} = 56$ possible combinations. The resulting scores are listed in table VI-H.

When considering figure 2, it is clear to see that the RNN model disagrees the most with all other models. This explains its prevalence in many of the lowest-score three-model combinations. However, this does not mean that only combinations involving the RNN model are interesting. In fact, one very plausible reason for its disagreement could be that its performance is just not quite on the same level as the other, Transformer-based models'. For this reason, we take only two combinations involving the RNN model: *[bertweet-large, rnn-gru, xtreme-distil]* (lowest agreement score) and *[bertweet-base, bertweet-large, rnn-gru]* (involv-

Model 1	Model 2	Model 3	Agreement Score ↓
3	5	8	0.7364
5	6	8	0.7384
2	5	8	0.7435
1	5	8	0.7464
4	5	8	0.7485
1	3	5	0.7494
1	5	6	0.7522
5	7	8	0.7524
3	5	7	0.7534
1	2	5	0.7545
3	5	6	0.7564
2	5	7	0.7579
2	5	6	0.7584
5	6	7	0.7584
1	4	5	0.7604
3	4	5	0.7606
2	3	5	0.7619
2	4	5	0.7634
1	5	7	0.7636
4	5	7	0.7654
4	5	6	0.7685
1	3	8	0.8076
1	6	8	0.8084
3	6	8	0.8095
3	7	8	0.8097
6	7	8	0.8128
3	4	8	0.8146
2	6	8	0.8161
1	2	8	0.8175
1	4	8	0.8177
2	7	8	0.8189
4	6	8	0.8208
4	7	8	0.8208
2	3	8	0.8222
2	4	8	0.8222
1	7	8	0.824
1	3	7	0.8378
1	3	6	0.8408
1	6	7	0.8419
3	6	7	0.8432
1	3	4	0.8437
3	4	7	0.8448
1	2	7	0.845
1	2	6	0.8453
2	6	7	0.8469
1	4	7	0.8478
1	2	4	0.8491
2	4	7	0.8494
2	3	7	0.8499
1	2	3	0.8506
1	4	6	0.851
4	6	7	0.8544
3	4	6	0.8641
2	3	4	0.8659
2	4	6	0.8661
2	3	6	0.8664

Table IX
THREE-MODEL AGREEMENT SCORES FOR ALL POSSIBLE COMBINATIONS OF 3 DIFFERENT MODELS OUT OF OUR 8 MAJORITY-VOTING BASE MODELS.

ing the very-best-performing base models).

The next-lowest agreement score without the RNN is achieved by *[bart, bertweet-large, xtreme-distil]*. Furthermore, since quite a lot of combinations following this one all include the *xtreme-distil* model, we jump to the next one that does not: *[bart, bertweet-large, xlnet]*.

In order to incorporate the *twitter-roberta* model and its double-variant, we also include *[bart, twitter-roberta, xlnet]* and *[bart, bertweet-large, double-twitter-roberta]* as the combinations with the respectively next-lowest scores.

Finally, we also take *[bertweet-base, bertweet-large, twitter-roberta]*, even though it has the highest agreement score. Our reasoning for this is simply that this particular combination contains all our best-performing base models.

On top of this, we also evaluate a combination of all 8 base models, except for the RNN (to achieve an odd number of voters).