Deduplication Is All We Did

Lorenzo Benedetti[1], Tommaso Di Mario[1], Andrea Sgobbi[1], Gabriel Tavernini[1]
Group: *cilling*
[1]Department of Computer Science, ETH Zurich, Switzerland

***Abstract*—This project aims at classifying positive and negative sentiment over a dataset of 2.5 million tweets, labeled using distant supervision based on textual emoji. The core of our work involves several novel contributions tackling the challenges of the dataset. Firstly, we investigate various data labeling techniques, including selecting the most frequent label, applying soft labels, and implementing label smoothing to address data duplication and label noise challenges. We then evaluate a broad spectrum of baseline models, including logistic regression, random forests, feedforward and convolutional neural networks, long short-term memory (LSTM) networks, and a non-parametric technique based on GZIP. Our principal contribution lies in the development of an advanced classification framework combining fine-tuned BERT models with a custom classification head within an ensemble architecture. This approach integrates model predictions through a final mixing regressor assigning importance weights to each ensemble model. Our experimental results demonstrate the effectiveness of deduplication and smoothing for our dataset, as well as the ability of ensembles to achieve performance superior to any single model within them. Finally, thanks to the wide spectrum of models studied, we provide an analysis of how model parametrization influences task performance.**

I. Introduction

In the age of social media, billions of people every day communicate in real-time and share opinions on public platforms; with over 500 million tweets posted daily, Twitter is practically a vast repository of user-generated content that reflects public opinion on a multitude of topics, from global events to personal experiences. Analysing such sentiment-rich text can offer valuable insights for various stakeholders, including businesses and politicians.

Sentiment analysis can be defined as a process that automates the mining of attitudes, opinions, views and emotions from text, speech, tweets and database sources through Natural Language Processing (NLP) [1]. Sentiment analysis involves classifying opinions in text into *positive* and *negative* (often a third category, *neutral*, is used but that's not the case in the dataset we worked on).

Accurate tweet sentiment analysis has numerous applications. For instance, businesses can leverage insights from sentiment classification to better understand customer sentiment and inform various aspects of their operations. Likewise, politicians can track public opinion on policies and events, enabling more effective election campaigns.

To achieve this, many techniques, ranging from simple NLP solutions to complex transformers [2], have been tried with different success levels. In Section II we delve into the details of some of the models we explored ourselves.

Given the noisy nature of the dataset, a notable amount of work is also spent trying to find the optimal way to preprocess the tweets before training our models.

II. Models and Methods

In this study, we employed the following approach for solving the task. Firstly, we implemented the various baselines experimenting with both machine learning and deep learning techniques together with different embeddings. Secondly, we explored pretrained large language models of the BERT [2] family, mainly `bertweet`[3] and `twitter-roberta` [4], where we introduced a slimmer classification head. Additionally, upon noting that there were numerous duplicates in the dataset, we applied three different data deduplication techniques, namely *Hard*, *Soft* and *Smooth* improving label accuracy by reducing inconsistencies and redundancies, leading to more reliable data for training and analysis.

A. Data

The training dataset consists of a collection of 2.5M tweets and their corresponding class, either 1 for *positive* or -1 for *negative*. In particular, a tweet is labeled 1 if the tweet contains the emoji ':)', and it is labeled -1 if it contains the emoji ':('. The labels are then converted to $\{0, 1\}$ to be modeled as probabilities. The dataset comes with the user tags and URLs replaced with `<user>` and `<url>` respectively, in order to anonymize and standardize the text for further analysis and, ultimately, reduce the noise in the training set.

For local testing, we opted for a 95% training - 5% validation split of the data. We kept the validation set relatively small at only 125k samples since we wanted our models to be trained on as much data as possible for better results on the Kaggle test set.

Given the algorithmic-labeling nature of the dataset, it contains many duplicates, often with contrasting labels. In particular, we find that more than 9% of the training split entries are duplicates. On top of this, roughly 1% of the dataset has noisy labels, meaning there are duplicate entries with conflicting labels. Thus, to improve training and generalization of the models, we test different techniques for data deduplication and label conflict resolution.

Note that duplicates were left in the validation set, to more closely simulate the actual test set, and hence give us more indicative results on the performance of the different strategies.

We implemented and tested the following deduplication strategies:

- *None*. As a starting point for our tests, we used the dataset without any changes.
- *Hard*. Duplicate entries were collapsed into the class they are most frequently labeled as (either 0 or 1). If a tie is found in the labeling, the instance is assigned to the *positive* class.
- *Soft*. Duplicates were assigned a soft label defined as their average class $\in [0, 1]$. This avoids the loss of data distribution information when the label is kept binary.
- *Smooth*. Label Smoothing is an effective regularization tool that has been shown to yield benefits for model generalization, robustness, and calibration [5]. It's often used to reduce the overfitting problem of DNNs and further improve classification performance [6]. Starting with *Soft* labels, we apply the formula from [5] for $K = 2$ and $\epsilon = 0.01$ as we've observed noise in roughly 1% of the labels:

$$y \leftarrow y \cdot (1 - \epsilon) + \frac{1}{K} \cdot \epsilon$$

B. Baselines

We established a foundation for our study by implementing various baselines. These models, although generally simpler and poorer in performance, will serve as a reference point for the other models.

1) Machine Learning Baselines: Our first approach was to try different embeddings together with simple ML classifiers. We trained the following classifiers:

- Logistic Regression [7] is a model mainly used for binary classification tasks, given its simplicity and efficiency we decided to use this model as a first baseline.
- Random Forest [8] is an ensemble method that uses multiple decision trees to get a more accurate and stable prediction after merging the results of each tree.

The models mentioned above were trained using two-word embedding techniques:

- GloVe [9] (Global Vectors for Word Representation) is an unsupervised learning algorithm that is used to obtain dense vector embeddings.
- TF-IDF [10] (Term Frequency-Inverse Document Frequency) measures the importance of a word to a document in a corpus, the embeddings are computed directly at the document level and they are highly sparse.

2) FFNN: Feed-Forward Neural Networks or Multi-Layer Perceptron consist of multiple layers of fully connected nodes with a non-linear activation function.

We implemented a FFNN that consists of an embedding layer followed by three fully connected layers interleaved by a dropout layer to prevent overfitting.

Lastly, we use the sigmoid function to perform binary classification.

3) TextCNN: Following [11] we developed a textCNN baseline. The model we implemented consists of an embedding layer followed by three 1D convolutional layers and as many max pooling layers to reduce the sentence to a single vector. Lastly, after a fully connected layer, we perform binary classification.

For what concerns the training configuration, we trained the model for 30 epochs, with increasing filter sizes (3,4,5).

4) LSTM: Recurrent neural networks are particularly well suited for the analysis and classification of sequences. A widely used improvement on vanilla RNNs is the Long Short-Term Memory Cell [12] which tries to mitigate exploding and vanishing gradients.

Inspired by [13] we used a tokenizer and implemented a model using an embedding layer followed by 3 LSTM layers and finally 2 dense layers. The tokenizer we used is the one from BERTweet.

Our model used an embedding dimension of 128 and 3 LSTM layers of 512 units each. The MLP uses ReLU as an activation function for the hidden layer, which has 16 units and a sigmoid for the single unit output.

We trained the model for up to 60 epochs but noticed a plateau in performance after only 5 epochs.

5) BiLSTM: Using the same configuration as the above-mentioned LSTM model we also tried using a Bidirectional LSTM architecture. The intuition behind this approach is that by processing data in both directions, the model can better understand the relationship between tokens.

The only difference is the input size of the MLP, which doubles given it now takes 2 hidden states as its input, one from the forward LSTM and one from the backwards LSTM.

We again trained the model for up to 60 epochs but noticed a plateau in performance after only 5 epochs.

6) GZIP-KNN: Another approach we explored is the one introduced by [14]. This method stands out for its simplicity, being parameter-free and more data-efficient compared to the methods we have considered so far.

The algorithm iterates through all pairs of tweets x_1 and x_2, where x_1 is in the test set and x_2 is in the training set, and computes the Normalized Compression Distance (NCD):

$$\text{NCD}(x_1, x_2) = \frac{C(x_1 x_2) - \min(C(x_1), C(x_2))}{\max(C(x_1), C(x_2))},$$

where $x_1 x_2$ denotes the concatenation of x_1 and x_2, and $C(x)$ equals the length of x after being compressed with GZIP. Finally, we use K-Nearest Neighbors [15] to classify the tweets in the test set.

Due to limited computational resources, we performed the classification using only a portion (approximately 500k tweets) of the training set.

C. Transformers

In recent years, Large Language Models (LLM) have shown impressive capabilities in generating and understanding human-like text. These models have been pre-trained on large amounts of data and fine-tuned to perform specific tasks. Among those tasks LLMs such as BERT [2], which are based on the transformer architecture [16], have been outperforming previous architectures on sentiment analysis and various models have been specifically pre-trained on Twitter datasets. Furthermore, APIs and libraries developed by organizations like HuggingFace [17] have significantly streamlined the development and deployment of LLMs, enabling us to efficiently experiment with various models.

We chose to use models in the BERT family of encoders for two main reasons: these models already have a dedicated classification token `[CLS]` which can easily be adapted to our task, and their comparably small size allows them to fit within the 11GB of VRAM of the GTX 1080TI we have available. We identified two models in this family which were pretrained on Twitter datasets: `twitter-roberta` and `bertweet`, both of which were trained following the RoBERTa [18] training procedure.

As part of our contribution, we replaced the default classification head employed by the HuggingFace library with a "slimmer" alternative. Instead of having an extra dense layer with both input and output dimensions equal to the hidden state size, we perform the down-projection on the `[CLS]` token output right away. Furthermore, whilst the standard model resorts to a regression loss when the output is one-dimensional, we opted to use a binary cross-entropy loss instead. Thanks to these changes we noticed substantial improvements in accuracy over the base model.

The models were trained over 3 epochs, beyond which we experienced no noticeable improvement, and using a batch size of 32 samples due to hardware limitations. We also employed Weight Decay, which for large overparametrized models has been suggested to play a critical role in the regularization mechanisms that enable generalization [19]. Although this proved effective for the `bert-base` models with roughly 135M trainable parameters, the `large` models with 355M parameters exhibited some unexpected behavior within the first couple thousand training steps, with the loss quickly climbing and eventually regressing to a random guesser. We attributed this either to overfitting due to overparametrization of the model, or alternatively to the gradients being overly noisy due to the smaller batch size imposed by the VRAM limitations.

To solve this problem we enhanced our model with LoRa [20] adapters, more specifically using the revised scale parameter suggested in rsLoRa [21]. LoRa is a Parameter-Efficient Finetuning technique which adds trainable low-rank matrices alongside the normal transformer layers. We inject rank 16 adapters for Query, Key and Value matrices of the Self-Attention layer, as well as for the dense linear layer of the piecewise MLP. All other model parameters, save for the classification head, are frozen during training, resulting in only 7.1M trainable parameters, less than 2% of the original amount.

Dedup Strategy	LSTM	bertweet-base
None	87.10	91.24
Hard	87.34	91.40
Soft	87.38	91.52
Smooth	**87.52**	**91.62**

Table I
KAGGLE PUBLIC SCORES USING THE VARIOUS DEDUPLICATION STRATEGIES PROPOSED.

D. Ensemble

Ensemble models leverage the combined strengths of diverse individual models to achieve superior performance compared to any single model. In this section, we explore the implementation of an ensemble model built upon the transformer models discussed in the previous section. By integrating several LLMs, our ensemble approach harnesses the varied insights and predictive capabilities of each constituent model: this is corroborated by the fact that, although some models in the ensemble may underperform compared to others, their inclusion within the ensemble will lead to improved collective performance.

We make use of 7 different models within our ensemble: 2 based on `twitter-roberta-base`, 3 on `bertweet-base` and 1 version of each `large` model trained using LoRa. A simple approach would be to average the model outputs to obtain the final estimate, however this completely ignores the fact that some models have proven more effective than others and should thus be weighed accordingly. We therefore opted to train a final *mixing* layer by leveraging the 5% of held-out data used for validation. Due to the small dataset and low data dimensionality (we have a single floating-point output per model) the use of complex methods such as Random Forest or XGB classifiers proved prone to overfitting. We thus limited ourselves to the three simplest solutions possible: linear, ridge and logistic regression, which effectively assign an importance weight for each model to contribute to the ensemble distribution.

III. RESULTS

As introduced in section II, we started our experiments with various baselines. For all models trained we showcase Kaggle public scores, as well as validation accuracy, in table III.

First, we considered the simple models coming from standard machine learning that do not involve DNNs. Models

Model	# Param.	Val. Acc.	Kaggle Acc.
GZIP-KNN	0	-	74.50
Logistic + GloVe	200	78.78	77.00
Logistic + TF-IDF	1k	80.66	79.92
RF + GloVe	189.4k	74.41	72.46
RF + TF-IDF	4.6M	76.39	74.92
FFNN	6.4M	85.03	84.20
CNN	6.7M	86.88	86.74
LSTM	11.9M	88.09	87.52
BiLSTM	21.6M	88.19	87.98
twitter-roberta-base	124.6M	91.30	91.30
bertweet-base	134.9M	91.87	91.62
twitter-roberta-large	362.5M	91.46	91.14
bertweet-large	362.5M	92.10	91.80
Ensemble	1378.9M	**92.44**	**91.98**

Table II
COMPREHENSIVE RESULTS FOR ALL MODELS TESTED, REPORTING PARAMETERS, VALIDATION ACCURACY AND KAGGLE PUBLIC SCORE.

Mixing Strategy	Validation Accuracy	Kaggle Accuracy
Averaging	92.308	91.80
Linear Regression	92.432	91.82
Logistic Regression	**92.444**	**91.98**
Ridge Regression	92.420	91.84

Table III
KAGGLE PUBLIC SCORES AND VALIDATION ACCURACY FOR DIFFERENT ENSEMBLE MIXING LAYERS.

using GloVe embeddings represent a tweet as the average embedding. Despite the low computational cost and number of parameters, logistic regression shows impressive results, especially with TF-IDF embeddings.

Next, we show the results for DNN-based models. For each model, we also report the respective number of trainable parameters in table III. Furthermore, we report in table I the achieved accuracy on the public Kaggle test set of the LSTM baseline and `bertweet-base` under the different deduplication strategies proposed in section II-A.

We show the correlation between model parameters and achieved accuracy in figure 1. When using LoRa, we chose to include the untrained parameters as they remain indicative of training performance. Notice that although more parameters is nearly always better, the amount of parameters needed scales exponentially with accuracy, with larger models clearly showcasing diminishing returns.

Finally, we explore the effectiveness of our mixing layers in table III. We observed that while simply averaging model outputs does not greatly improve over the single best model, the use of a trained mixing layer allows the ensemble to better leverage the individual model strengths and perform better than the sum of its parts.

IV. DISCUSSION

In this discussion, we analyze the impact of various techniques on model performance, parameter efficiency, and present a small ablation on the ensemble model. Our findings highlight the following key points:

1) Implementing data deduplication strategies (*Smooth*, *Soft* and *Hard*) resulted in improved accuracy and consistency of the training data. Out of the 3 strategies *Smooth* stood out, leading to improvements of up to 0.4% in test accuracy for the models trained with it. This indicates that addressing duplicate entries and label conflicts is crucial for training effective sentiment analysis models.
2) Despite the optimal performance, the ensemble does not fare well in terms of parameter efficiency. On the other hand, training with LoRa techniques offers very similar performance by optimizing a fraction of the parameters, falling just 0.18% short of the ensemble. The other stand-out in parameter efficiency is the GZIP-KNN method, which although less performant than the other baselines at 74.5% accuracy, does so without any conditioning on the data.
3) Since the mixing parameters can be interpreted as importance weights, an idea would be to "prune" the models deemed least important by the final layer. Testing this, we instead found that removing any of the models from the ensemble would degrade performance, which could be interpreted as evidence that the ensemble is indeed able to effectively reconcile conflicting information.

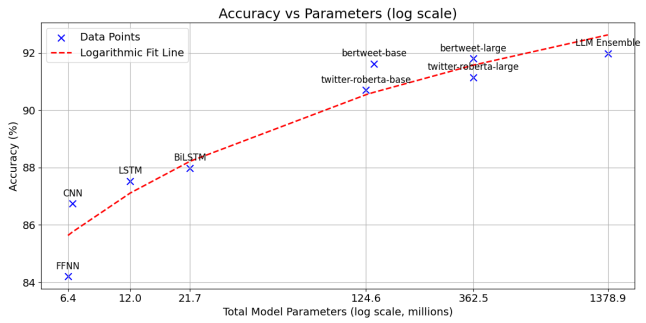

Fig. 1. **Parameter Efficiency.** Kaggle public scores of neural network-based models as a function of their total parameters in logarithmic scale. LoRa models are reported as the total number of parameters as opposed to only those that are trainable. We fit a logarithmic line showing the diminishing returns of larger models.

V. SUMMARY

We demonstrated the effectiveness of transformer models in the field of sentiment analysis, particularly large language models in the BERT family. By leveraging the strength of multiple models we developed a high-performing ensemble model with a novel mixing strategy. Additionally, we introduced and tested various deduplication strategies which proved their importance in improving the quality of training data and robustness of models.

Finally, we analysed the trade-offs between model performance and parameter efficiency. While the ensemble model achieved superior accuracy, it required significantly more parameters, whereas LoRA adapters offered a more efficient training approach.

References

[1] V. A. and S. Sonawane, "Sentiment analysis of twitter data: A survey of techniques," *International Journal of Computer Applications*, vol. 139, no. 11, p. 5–15, Apr. 2016. [Online]. Available: http://dx.doi.org/10.5120/ijca2016908625

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019. [Online]. Available: https://arxiv.org/abs/1810.04805

[3] 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.

[4] D. Loureiro, F. Barbieri, L. Neves, L. Espinosa Anke, and J. Camacho-collados, "TimeLMs: Diachronic language models from Twitter," in *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 251–260. [Online]. Available: https://aclanthology.org/2022.acl-demo.25

[5] R. Müller, S. Kornblith, and G. Hinton, "When does label smoothing help?" 2020. [Online]. Available: https://arxiv.org/abs/1906.02629

[6] C.-B. Zhang, P.-T. Jiang, Q. Hou, Y. Wei, Q. Han, Z. Li, and M.-M. Cheng, "Delving deep into label smoothing," *IEEE Transactions on Image Processing*, vol. 30, p. 5984–5996, 2021. [Online]. Available: http://dx.doi.org/10.1109/TIP.2021.3089942

[7] D. R. Cox, "The regression analysis of binary sequences (with discussion)," *J Roy Stat Soc B*, vol. 20, pp. 215–242, 1958.

[8] T. K. Ho, "Random decision forests," in *Proceedings of 3rd International Conference on Document Analysis and Recognition*, vol. 1, 1995, pp. 278–282 vol.1.

[9] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[10] K. Sparck Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, vol. 28, no. 1, pp. 11–21, 1972.

[11] Y. Kim, "Convolutional neural networks for sentence classification," 2014. [Online]. Available: https://arxiv.org/abs/1408.5882

[12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, pp. 1735–80, 12 1997.

[13] D. Murthy, S. Allu, B. Andhavarapu, and M. Bagadi, "Text based sentiment analysis using lstm," *International Journal of Engineering Research and*, vol. V9, 05 2020.

[14] Z. Jiang, M. Yang, M. Tsirlin, R. Tang, Y. Dai, and J. Lin, ""low-resource" text classification: A parameter-free classification method with compressors," in *Findings of the Association for Computational Linguistics: ACL 2023*, A. Rogers, J. Boyd-Graber, and N. Okazaki, Eds. Toronto, Canada: Association for Computational Linguistics, Jul. 2023, pp. 6810–6828. [Online]. Available: https://aclanthology.org/2023.findings-acl.426

[15] E. Fix and J. Hodges, *Discriminatory Analysis: Non-parametric Discrimination: Consistency Properties*. USAF School of Aviation Medicine, 1951. [Online]. Available: https://books.google.ch/books?id=4XwytAEACAAJ

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023. [Online]. Available: https://arxiv.org/abs/1706.03762

[17] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing." in *EMNLP (Demos)*, Q. Liu and D. Schlangen, Eds. Association for Computational Linguistics, 2020, pp. 38–45. [Online]. Available: http://dblp.uni-trier.de/db/conf/emnlp/emnlp2020-d.html#WolfDSCDMCRLFDS20

[18] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," 2019. [Online]. Available: https://arxiv.org/abs/1907.11692

[19] M. Andriushchenko, F. D'Angelo, A. Varre, and N. Flammarion, "Why do we need weight decay in modern deep learning?" 2023. [Online]. Available: https://arxiv.org/abs/2310.04415

[20] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "Lora: Low-rank adaptation of large language models," 2021. [Online]. Available: https://arxiv.org/abs/2106.09685

[21] D. Kalajdzievski, "A rank stabilization scaling factor for fine-tuning with lora," 2023. [Online]. Available: https://arxiv.org/abs/2312.03732