Smiley prediction on Twitter data

Georgios Fotiadis (271875)*, Paola Mejia (322552)† and Angelika Romanou (309854)‡
School of Computer and Communication Sciences, EPFL
Lausanne, Switzerland
Email: *georgios.fotiadis@epfl.ch, †paola.mejia@epfl.ch, ‡angelika.romanou@epfl.ch

***Abstract*—In this paper, we apply machine learning methods to Twitter data to predict if a tweet message used to contain a positive or a negative smiley. We present four different models: a simple machine learning baseline model; two long-short term memory (LSTM) models using word2vec and GloVe embeddings respectively; a zero-shot learning inspired model; and a BERT model. Our main contribution is the incorporation and comparison of state-of-the-art data representations, transformers and classification models, as well as the use of zero-shot learning for data efficiency. Our proposed model is the one that uses CT-BERT language model which achieves 0.906 accuracy and 0.905 f1-score in the test set and it was placed at the third position of the respective AIcrowd competition.**

I. Introduction & Problem Definition

With the keep increasing volumes of user complex textual data, text classification is a relevant challenge that requires a deeper understanding of machine learning methods to be able to accurately classify texts in many applications. The success of these learning algorithms relies on their capacity to understand complex models and non-linear relationships within data [1]. An interesting domain of text classification and natural language understanding is the analysis of textual data in terms of the opinion they express and the sentiment they have. Hence, sentiment analysis in Twitter is a field that has been attracting research interest with various published scientific works [2]. This is due to the platform's popularity and the special characteristics of the published tweets such as their short length, their rich metadata, and their informal syntactic structure.

In this work, the problem we are intending to solve is to predict the existence of a positive or negative smiley face which is similar to the sentiment analysis problem. Naturally, the existence of a positive or negative smiley might mean that the text has positive or negative sentiment respectively. However, there might be other emotions like irony or sarcasm which we do not take into consideration in our approach and in the used dataset.

Moreover, recent studies in text operationalization and representation have shown novel ways to represent an input sequence in a multidimensional space, making classification algorithms perform more efficiently on the input data. In our implementation, we use different word embedding representations, from simple tf-idf to more dense ones such as GloVe and word2vec. Furthermore, we also perform experiments with textual representations created by transformer models such as BERT.

The rest of the document is structured in the following manner: Section II presents the universal data preprocessing steps that we took which were applied in all the proposed models; Section III explains the methodology for each implemented model; Section IV presents the results of the models and the comparison among them; Section V analyzes the errors and finally, Section VI concludes this work.

II. Dataset & Data preprocessing

The dataset is consists of two classes: positive tweets and negative tweets. The words and punctuation are properly tokenized and all words are in a lowercase format. Unfortunately, the later characteristic of the dataset makes impossible the extend of the existing hashtags by converting camel case hashtags into the respective phrases (e.g. #ThisIsAPhrase to "this is a phrase"), and thus, we might have lost some interesting textual patterns.

The dataset is balanced with 1,250,000 positive tweets and 1,250,000 negative ones. It is very important to note that almost 12% (288,789) of the tweets are duplicated. For this analysis, the duplicated tweets were removed in order to avoid information likage from training to validation set.

Inspired by the data preprocessing done for tweets in [3], the following data transformations were explored:

- Standardizing the texts by replacing accented characters by non-accented characters (example: smïléy to smiley), removing tabs, changes of line, and control characters.
- Removing twitter special syntax like indications of "RT", $< url >$ or $< users >$.
- Translating emojis from punctuation characters to words like positive, negative, neural and heart. In addition, the library emoji [4] was used to translate other emoticons to words.
- Removing extraaaa letters at the end of the words.
- Expanding contractions into full words (example: won't to will not).
- Expanding slang contractions into the complete words (example: gr8 to great).

Data preprocessing was different for models using transformers because transformers do benefit from stop words, conjugated verbs and punctuation. On the contrary, for transformers models, the punctuation was not removed but simplified to keep the last punctuation sign when multiple consecutive punctuation signs occur. Furthermore, for the rest of the models, additional data preprocessing was made with the

punctuation and existing stop words removed and all the words were lemmatized to transform them to their simplest form.

III. Methodology

In this section we present the methodology followed in order to provide meaningful text representations for the model input, define the different baseline models and propose state-of-the-art inspired models based on the existing literature.

A. Model A: Tf-idf Baseline

[++] After the text transformation into numerical vectors using bag-of-words representations, remaining is the application of classification algorithms. For this, we employed several machine learning algorithms from more simple to more advanced ones. The classifiers used are the following: Logistic Regression, Naive Bayes, Support Vector Machines and Random Forest. Result comparisons and further analysis on the experiments are presented in Section IV.

B. Model B: Deep learning LSTM model with word embeddings

Bag-of-word representations are highly sparse with feature size dependent on the vocabulary. Furthermore, in the bag of words approach all words synonyms and words with related meaning are treated as completely independent features. For those reasons, more dense text representations such as word embeddings were implemented. With word embedding, each word is represented by a vector that maps the word into a multidimensional space. In that way, words that are close to each other have a closer meaning. In our work, we train word2vec [5] embeddings and we also use pre-trained GloVe [3] embeddings that are trained on Twitter data.

Since our previous models did not capture the sequential nature of the text data, for this model we use Bi-directional long short-term memory (BiLSTM) [6] neural network that takes as input the complete sentence and provides as output a predicting label. BiLSTM allows the network to have both backward and forward information about the sequence at every time step. The input data is three-dimensional: number of tweets, tweet's sequence length, word embedding length. In order to keep the second dimension the same for all tweets, we found the max words contained in a tweet, and we padded the sentences with zero vectors until they reach the desired length. Figure 1 depicts the network structure of the BiLSTM model used. For the case of word2vec representations, an embedding layer prior to biLSTM was used.

C. Model C: Bidirectional Encoder Representations from Transformers (BERT) model

Transformers are networks built with attention [7]. They apply attention mechanisms to gather information about the relevant context of a given word, and then encode that context in a rich vector that smartly represents the word. State-of-the-art results in Natural Language Processing (NLP) have been obtained with Bidirectional Encoder Representations from Transformers (BERT) [8]. Our models are built based on [9].

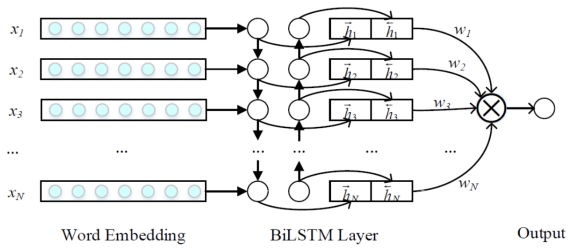

Fig. 1. Architecture of the BiLSTM

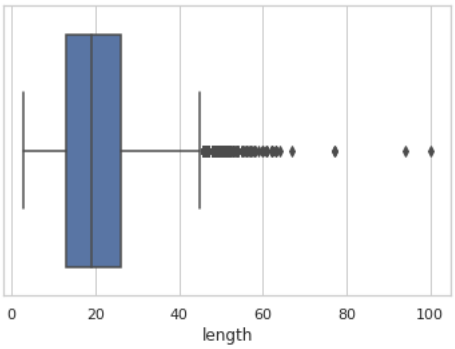

Fig. 2. Length of the tensors with BERT encoding

Pre-training BERT is very expensive (4 days on 4 to 16 Cloud TPUs), nevertheless, **fine-tuning** is much more accessible. The pipeline used to fine-tune the models was the following:

1) Firstly, the data was cleaned and preprocessed with the transformations mentioned in II.
2) The cleaned tweets were tokenized and then encoded. In order to use BERT for classification the special token [CLS] must be added at the beginning of each tweet and the token [SEP] indicates the end of the tweet. In addition, every sentence/sequence must have the same length. The maximum possible value is 512 but smaller values speed up training. Figure 2 shows the length of the tensors without setting a limit, the maximum length was set to 100 given that it is the longest sequence in the training set. If the test set contains longer sequences they will be truncated. In order for BERT to differentiate between tokens and padding, attention masks were created with the encodings using the function encode_plus.
3) Thirdly, the data was loaded in batches and the pre-processed model is loaded. The models explored were transformers for feature classification. These transformers have an extra linear layer on top of the pooled output. The different models tested were DistilBERT [10], RoBERTa [11], and BERT (base and large) [8] and CT-BERT [12]. To prevent the gradients from "exploding" the norm of the gradients is clipped to 1 as seen in [9] [1]

[1]https://github.com/huggingface/transformers/blob/master/examples/text-classification/run_glue.py#L170

D. Model D: Zero and Few shot learning model

Zero and few shot learning models is a very recently created class of models where the learner can classify object into classes that it has not been trained with. This is the opposite of how neural network classifiers are traditionally used, meaning that they are only able to classify objects in a set of strictly predefined classes. In our case, we used the TARS classifier [13] in the context of the Flair framework [14]. With Flair, we had the option to either use the model directly (zero-shot learning) or additionally train it with a small subset of our data, using a very small learning rate (few-shot learning). We created two classes, (positive and negative) and asked the model to classify the preprocessed tweets in one of them. It's worth emphasizing again that the model itself has not been trained for classifying text in these categories and we could change them to whatever we wanted to (ex. happy, sad, plus, minus etc.).

We observed that when ran in zero-shot learning mode, the model had difficulties classifying tweets whose text was comprised of a couple of words and resulted in run-time errors. That's why we decided to proceed with the few-shot learning approach and varying the amount of data we trained the model with. Specifically, we trained the model with 1% and 5% of the data and you can see how this affected the model's accuracy in the next chapter.

E. Implementation

Colaboratory [15] was used to facilitate neural nets training. The modes were trained on a single GPU at the time. The GPUs available in Colab vary over time but often include Nvidia K80s, T4s, P4s and P100s. The main disadvantage of using Colab is that Notebooks disconnect from VM when left idle for too long. The strategy used by the authors was to save the models after every epoch so that when the connection timed-out, training could be restarted from the previous checkpoint. It is relevant to note that training the models was very challenging and the authors encountered errors like CUDA out of memory, sessions expiring and reaching Google's GPU usage limits.

IV. Experiments & Results

This section presents and analyses the experimental results of the different classification methods used.

A. Model A: Tf-idf models

For the training of the machine learning models we perform a k-fold cross validation with 10 folds. The results for the machine learning models using tf-idf features as input were depicted in Table I. We observe that regardless of the baseline, all implemented models have very close to each other f1-scores. However, models such as Naive Bayes and Random Forests favour recall... [++]

TABLE I
PERFORMANCE OF DIFFERENT MACHINE LEARNING MODELS WITH TF-IDF INPUT

Model	Accuracy	Precision	Recall	F1-score
Logistic regression	**0.760**	**0.763**	0.759	0.761
Naive Bayes	0.751	0.713	**0.843**	0.773
SVM	0.752	0.741	0.779	0.760
Random Forest	0.758	0.728	0.830	**0.775**

TABLE II
PERFORMANCE OF DIFFERENT DEEP LEARNING MODELS

Model	Learning Rate	Epochs	Embedding Size	F1-score
Word2Vec BiLSTM	1-e05	100	64	0.000
Pre-trained Glove BiLSTM	1-e04	5	100	0.761

B. Model B: Deep learning LSTM model with word embeddings

For the word embedding models, we performed hyper-parameter tuning on the learning and dropout rates and the size of the word embeddings. We set the batch size to 250 and the epoches to 100 for all experiments. The final hyper-parameter settings along with the results are depicted in Table II. [++]

C. Model C: Bidirectional Encoder Representations from Transformers (BERT) model

The pre-trained models tested were DistilBERT [10], RoBERTa [11], and BERT (base and large) [8] and CT-BERT [12]. 90% of the data (1,990,089 tweets) was used to train the models and 10% (221,121 tweets) to vaidate. A further enhancement to their project would be to use k-folds cross-validation. The same seed (123) was used for replicability and the following hyperparameters were explored:

- [8] recommends exploring different values of learning rates (5e-5, 3e-5, 2e-5), however, given that 2e-5 gave the best results in some models, lower learning rates were also explored (1e-5 and 5e-6).
- Adaptive schedulers were used to adapt the learning rate. The following schedulers were explored constant scheduler, linear scheduler, cosine scheduler with warm-up and cosine with hard restarts scheduler.
- Batches or 16 and 32 were explored.
- Models were run for 4 epochs each.

Table III shows the best hyper-parameters and accuracy for each model. The first model explored was BERT base uncased (all letters in lowercase) which consists of 12 layers, 12 attention heads and 110 million parameters. As seen on table III, the accuracy was 0.890 (MISSNG above the baseline). The accuracy of BERT large was 0.09 points higher. The difference between both models is that BERT large was almost 3 times as many parameters (340 million parameters).

TABLE III
PERFORMANCE & SETTINGS OF DIFFERENT BERT IMPLEMENTATIONS

Model	Epochs	Learning rate	Batch size	Scheduler	Accuracy
BERT (base)	3	1.00E-05	32	cosine	0.890
BERT (large)	2	1.00E-05	16	cosine	0.899
DistilBERT	4	2.00E-05	32	linear	0.866
roBERTa (large)	2	2.00E-05	16	cosine with hard reset	0.784
CT-BERT	3	5.00E-06	16	cosine	**0.904**

Much of BERT's success relies on the fact that it has been trained with a large corpus (3.5 billion words from Wikipedia and 0.8 billion words from free book corpus), nevertheless, the texts in Twitter have a culture of its own. The texts differ from classical text because of the brevity, multiple abbreviations, use of emojis and hashtags. To address this, we tried CT-BERT, a model trained on a corpus of 160 million tweets. Despite the tweets being related to COVID-19, the model had the highest accuracy out of all the models we explored.

To improve runtimes, DistilBERT was explored, this model runs 60% faster than BERT and in some cases preserves 95% of BERT performance. In our project, the accuracy dropped but good results were still obtained. Then we explored RoBERTa, different from BERT, RoBERTa removes the Next Sentence Prediction (NSP) task from BERT's pre-training and introduces dynamic masking so that the masked token changes during the training epochs, nevertheless, with this dataset it does not achieve better results.

V. ERROR ANALYSIS

It is interesting to note that on the validation set CT-BERT does slightly better at predicting correctly "happy" tweets (f1-score 0.917) than "unhappy" tweets (f1-score 0.893).

The false-negative and false-positive tweets were analyzed, these are a few possible explanations with examples.

A. Duality

The following three tweets exemplify the duality of some tweets. They were wrongly classified as positive although in reality, they had a sad face.

1) *i love my psychology class , too bad its almost over !*
2) *i am good ! how are you ? and i know right*
3) *i love you do not be angry .*

The challenge with this type of tweets is that they are composed of two parts: a positive and a negative part. Although these tweets had a sad face, it could have been very likely that for some parts there had been a happy face too. For example:

1) *i love my psychology class :) , too bad its almost over :(!*
2) *i am good :) ! how are you ? and i know right :(*
3) *i love you :) do not be angry :(.*

B. Lack of context

The following three tweets were also false negatives. Meaning they had a sad face and were classified as having a happy emoji. In these three cases, depending on the context, the two emojis would fit.

1) *on my way to school*
2) *looks that way*
3) *yes please*

C. Sarcasm

Some tweets have a sarcastic tone hard for the classifier to detect. For example: *dear all my followers , i will be dead in minutes nice knowing you [true label :()* was wrongly classified as having a happy emoji perhaps because of the first and last part of the sentence. There might be other cases in which the emojis are used in a counter-intuitive way, for example, *why should i hate it ? ! nevermind i am not going to talk [true label :)].*

VI. CONCLUSIONS

In this work, we showcase the importance of the data and model understanding

One big contribution of this project is applying CT-BERT to a corpus unrelated to COVID-19 and still outperforming BERT-large. This indicates that the contributions of CT-BERT are not only on the health field but improvements can also be seen on other social media tasks. In the future, this work ad hyperparameter exploration could be a nice addition to [12]. In [12], the authors finalize saying there is room for improvement by tuning the hyperparameters. In this work we tuned the hyperparameters and a smaller learning rate (5e-06) gave better results than the default used by the authors (2e-05).

The source code of this project is available on GitHub[2] along with its detailed documentation.

[2]https://github.com/geofot96/MLproject2

REFERENCES

[1] K. Kowsari, K. Jafari Meimandi, M. Heidarysafa, S. Mendu, L. Barnes, and D. Brown, "Text classification algorithms: A survey," *Information*, vol. 10, no. 4, p. 150, 2019.

[2] A. Giachanou and F. Crestani, "Like it or not: A survey of twitter sentiment analysis methods," *ACM Computing Surveys (CSUR)*, vol. 49, no. 2, pp. 1–41, 2016.

[3] 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.

[4] T. Kim and K. Wurster. (2020, November) Emoji. [Online]. Available: https://github.com/carpedm20/emoji

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[6] Z. Huang, W. Xu, and K. Yu, "Bidirectional lstm-crf models for sequence tagging," *arXiv preprint arXiv:1508.01991*, 2015.

[7] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in neural information processing systems*, 2017, pp. 5998–6008.

[8] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[9] 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 *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[10] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[11] 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," *arXiv preprint arXiv:1907.11692*, 2019.

[12] M. Müller, M. Salathé, and P. E. Kummervold, "Covid-twitter-bert: A natural language processing model to analyse covid-19 content on twitter," *arXiv preprint arXiv:2005.07503*, 2020.

[13] J. K. R. V. Kishaloy Halder, Alan Akbik. (2020) Task-aware representation of sentences for generic text classification. [Online]. Available: https://kishaloyhalder.github.io/pdfs/tars_coling2020.pdf

[14] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in *COLING 2018, 27th International Conference on Computational Linguistics*, 2018, pp. 1638–1649.

[15] G. Research. (2020) Frequently asked questions: Gpus. [Online]. Available: https://research.google.com/colaboratory/intl/en-GB/faq.html