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 message has a positive or a negative smiley. We present four different types of models: a set of simple machine learning baseline models; two long-short term memory (LSTM) models using word2vec and GloVe embeddings respectively; transformer models; and a few-shot learning model using TARS. 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 the few-shot learning method for data efficiency. Our proposed model is the one that uses the 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 (submission ID: 107963).**

I. Introduction & Problem Definition

With the ever increasing volumes of complex user textual data, text classification is a relevant challenge that requires a deep understanding of machine learning methods to be able to accurately classify texts from different domains. The success of these learning algorithms relies on their capacity to understand 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 probably indicates that the text has positive or negative sentiment respectively. However, there might be other emotions like irony and sarcasm which will not be taken into consideration for this emoji classification problem.

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 data preprocessing steps that were applied to 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 missed predictions of the proposed model and finally, Section VI concludes this work.

II. Dataset & Data preprocessing

The dataset consists of two classes: positive and negative tweets. The words and punctuation are properly tokenized and all words are in a lowercase format. Unfortunately, the latter characteristic of the dataset makes it impossible to extract the hashtag phrase by splitting words based on camel case (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 duplicates. For this analysis, the duplicated tweets were removed in order to avoid information leakage from the training to the validation set.

Inspired by the data preprocessing done for tweets by Pennington [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, neutral and heart. In addition, the library emoji [4] was used to translate other emoticons to words.
- Removing extra letters at the end of the words (example: "extraaaaa" to "extra").
- 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. Transformers do benefit from stop words, conjugated verbs and punctuation and thus, for these models only, the punctuation was not removed but simplified to keep the last punctuation sign when multiple consecutive punctuation signs occurred. For the rest of the models, punctuation and stop

words were 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, to define the different baseline models, and to propose state-of-the-art inspired models based on the existing literature.

A. Model A: TF-IDF Baseline

One of the most popular and intuitive approach of transforming the textual data into numerical is the bag-of-words (BoW) method. A BoW is a representation of text that describes the occurrence of words within a document. In this work we use a weighted case of BoW which is the Term Frequency - Inverse Document Frequency (TF-IDF) [5]. TF-IDF determines the relative frequency of a word in a specific tweet compared to the inverse proportion of that word over the entire collection of tweets. This statistical measure reflects how important a word is to a tweet. The created input matrix will have $N \times |V|$ dimensions, with N the size of the data samples and $|V|$ the vocabulary size.

After transforming the text into numerical vectors, we applied several machine learning classification algorithms with different complexity. The classifiers used are the following: Logistic Regression, Naive Bayes, Support Vector Machines and Random Forest.

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

Bag-of-word representations are highly sparse with the 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 embeddings, 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 trained word2vec [6] embeddings and we also used 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 a Bi-directional long short-term memory (BiLSTM) [7] neural network that takes as input the complete sentence and provides as output the predicted label. BiLSTM layers allow 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, and word embedding length. In order to keep the second dimension the same for all tweets, we found the maximum number of tokens contained in tweets from the training set, 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.

Specifically, the BiLSTM used for word2vec was a sequential network, having an embedding layer, a dropout layer

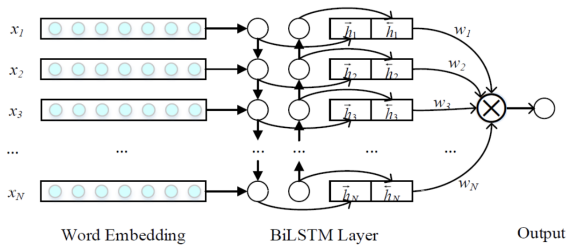

Fig. 1. Architecture of the BiLSTM

to prevent over-fitting, a bidirectional LSTM layer, and a final dense layer with sigmoid activation to perform the final classification. The BiLSTM used for GloVe embeddings was also a sequential network with two stacked bidirectional LSTM layers, and a final dense layer with sigmoid activation. Both models were trained with the Adam optimizer, having binary cross-entropy as the loss function.

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

Transformers are networks built with attention [8]. 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) [9]. Our proposed models are built based on [10]. To fully train a BERT language model would require an enormous amount of computational power that exceeds the scope and requirements of this assignment (4 days on 4 to 16 Cloud TPUs), and for that reason we performed fine-tuning on the pre-trained BERT language model. The pipeline used to fine-tune the models was the following:

1) Firstly, the data was cleaned and preprocessed with the transformations mentioned in Section II.
2) Then, the cleaned tweets were tokenized and 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] at the end of it. In addition, every input sequence must have the same length. Similar to what was done with the BiLSTM, the maximum input length was set to 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.
3) Thirdly, we trained the models. The used transformers were models with an extra linear layer on top of the pooled output for classification purposes. The different models tested were DistilBERT [11], RoBERTa [12], BERT (base and large) [9] and CT-BERT [13]. To prevent the gradients from "exploding" the norm of the gradients is clipped to 1 as seen in [10].

D. Model D: Zero and Few shot learning model using TARS

One of the biggest disadvantages of transformers is how computationally expensive fine-tuning is. Hence, we searched for alternatives that would allow us to get good results while significantly speeding the process up.

Zero and few shot learning models is a very recently created class of models where similarly to transformers, they are pre-trained models, ready to be used in unseen datasets, with potentially new labels. This makes them suitable solution for what we wanted to achieve. Specifically, we decided to use Task-Aware Representation of Sentences (TARS), which was introduced by [14] as a simple and effective method for few-shot and even zero-shot learning for text classification.

We used the TARS classifier [14] in the context of the Flair framework [15]. The difference between zero-shot learning and few-shot learning is that, zero-shot learning does not use data at all, whereas, in few-shot learning we can additionally train it with a subset of our data. We created two classes, (positive and negative) and then classified the preprocessed tweets. We varied the amount of input data until we found a good balance between training time and validation accuracy.

E. Implementation

Colaboratory (Colab)[16] 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 the Notebooks disconnect from the connected VM when left idle for too long. The strategy we used 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 we encountered errors like CUDA out of memory, sessions expiring and reaching Google's GPU usage limits. For this reason, multiple chunking operations were implemented especially in the data loading and preprocessing steps in order to perform these operations in a batched way. The source code of this project is available on GitHub[1] along with its detailed documentation.

IV. Experiments & Results

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

For the training of the machine learning and deep learning (BiLSTM) models, we performed a k-fold cross validation with 10 folds. For the transformer models, cross-validation was very expensive so we used 90% of the data (1,990,089 tweets) to train the models and 10% (221,121 tweets) to validate them.

A. Hyperparameter tunning

For the machine learning models which use the TF-IDF input, the default hyperparameters of the scikit-learn library were used:

- Logistic regression: Penalty = l2, C = 1
- Naive Bayes: alpha = 1
- SVM: C = 1, kernel = Gaussian (rbf)
- Random Forest: estimators = 100

For the BiLSTM with word2vec embeddings, we tuned the dropout rate (0.25, 0.50, 0.75), the embedding size (64, 128, 256) and number of training epochs (25, 50, 100). The best results were obtained with a dropout rate of 0.5 and embedding length of 128 and 100 epochs.

For the rest of the models (BiLSTM with GloVe, transformers and TARS), the following hyperparameters were explored:

- Devlin[9] recommends exploring different values of learning rates (5e-5, 3e-5, 2e-5), however, given that it some cases, the limits gave the best results, higher (2e-02, 2e-03, 1e-04) and lower values were explored (1e-05).
- Batches of sizes 16 and 32 were explored.
- Models were ran for 5 epochs each.

The best hyper-parameter settings along with the respective scores are presented in Table II.

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**

B. Results

The results for the machine learning models using TF-IDF features as input are depicted in Table I. We observe that regardless of the baseline, all implemented models have very similar F1-scores. However, models such as Naive Bayes and Random Forests perform better in terms of true positives and thus achieve better recall. Overall, the TF-IDF representation can provide some good baselines but without modeling the correlation and the sequential aspect of the words, this approach reaches its limitations. Given that the classes are balanced, for the rest of the analysis we will focus only on the accuracy.

Regarding the GloVe embeddings, pre-trained vocabulary always leads to better results compared to the trained-from-scratch ones (word2vec), which was expected due to the amount of data used for the pre-trained vocabulary. From the results in Table II, we observe that exploiting the sequential nature of the data, we achieved almost 10% increase in the model performance compared to the BoW approach.

The models that achieved the best scores overall were the transformers. As mentioned previously, the pre-trained models tested were DistilBERT [11], RoBERTa [12], BERT (base and large) [9] and CT-BERT [13].

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 II, the accuracy was 0.890 (0.13 above the TF-IDF logistic regression

[1]https://github.com/CS-433/cs-433-project-2-mlakes

TABLE II
PERFORMANCE & SETTINGS OF DIFFERENT MODEL IMPLEMENTATIONS

Model	Epochs	Learning rate	Batch size	Accuracy
Word2Vec BiLSTM	100	1-e05	32	0.842
Pre-trained Glove BiLSTM	5	1-e04	32	0.864
Few shot learning	5	2-e02	16	0.855
BERT (base)	3	1-e05	32	0.890
BERT (large)	2	1-e05	16	0.899
DistilBERT	4	2-e05	32	0.866
roBERTa (large)	2	2-e05	16	0.784
CT-BERT	2	1-e05	16	**0.904**

baseline). The accuracy of BERT large was 0.09 points higher. The difference between both models is that BERT large had almost 3 times as many parameters (340 million parameters).

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). However, as mentioned in the introduction, Twitter text differ from any other 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 training time efficiency, DistilBERT was explored. Benchamarks on this model show a 60% faster training than BERT while in some cases it preserves 95% of BERT performance [11]. In our project, the accuracy dropped but good results were still obtained. Moreover, we explored RoBERTa, which compared to BERT, 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. However, fine tuning RoBERTa on our data did not yield better results.

Regarding the TARS few-shot learning model, as mentioned in Section III, we varied the amount of data we trained it with until we found a good balance between training time and model performance. We trained the model with 0.1%, 1%, 5%, 10%, 25%, 50% and 100% of the data and we observed that even though naturally the more data the better the accuracy, the increase in the accuracy was insignificant after a certain proportion of data, but the training time grew disproportionately. Specifically, after using more than 25% of the data, the accuracy was reached a plateau of around 85.5%. In Table II we present the model trained on 25% of the data.

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" ones (f1-score 0.893). The false-negative and false-positive tweets were analyzed, and this section provides a discussion around possible explanations of model's false negatives and false positives.

A. Duality

The following three cases exemplify the duality of some tweets. They were wrongly classified as positive where 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 different text representations and machine and deep learning models. It is relevant to note that TARS and CT-BERT were released in 2020 and to our knowledge it is the first work that compares their performance. Even though the CT-BERT achieved the highest performance, the performance of TARS was remarkable given the little amount of data it was trained with. Another main contribution of this project is the application of CT-BERT to a Twitter corpus of different context and still outperform the state-of-the-art BERT-large. This indicates that CT-BERT is not only applicable on health (COVID) related tweets, but improvements can also be seen on other social media topics. The hyperparameter exploration that our work presents could also be an addition to Muller [13], since in their work, the authors conclude by saying that there is room for improvement for tuning the hyperparameters. We show in our work that a smaller learning rate (1e-05) gave better results than the default used by the authors (2e-05) in [13].

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] K. S. Jones, "A statistical interpretation of term specificity and its application in retrieval," *Journal of documentation*, 1972.

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

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

[8] 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.

[9] 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.

[10] 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

[11] 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.

[12] 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.

[13] 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.

[14] 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

[15] 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.

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