Yet another project on Twitter Sentiment Analysis

Giorgio Mannarini, Maria Pandele, Francesco Posa
École Polytechnique Fédérale de Lausanne, Switzerland

***Abstract*—Tweets are known to be extremely complex data to interpret, because of the language used, free and not attentive to grammar. The algorithms of text classification and sentiment analysis, tasks in which even many human beings would have difficulty, are numerous and varied. The goal of this paper is to compare most of these models on a classification task, adapting and improving them, and exposing their strengths and critical points. In particular, we will guide you in the development of a baseline composed of classic ML methods, up to complex Neural Networks, such as BERT (Bidirectional Encoder Representations from Transformers) [1], with which we achieved an accuracy of 90.2% (90.1% F1-Score) on the AIcrowd platform [2], on a testing set of 10,000 tweets.**

I. Introduction

The great abundance of tweets on the web is a gold mine for computer scientists, who have used this data for the most diverse purposes: marketing, profiling, sentiment analysis, etc. In particular, the challenge that this paper aims to solve, starting from tweets from which all positive *:)* and negative *:(* smileys have been voluntarily removed, is to predict which of the two emoticons each tweet used to contain. For the rest of this paper, we will call the two classes **positive** and **negative**. All data are provided by the AIcrowd platform as part of the *EPFL ML Text Classification* challenge [2]. The platform has also the role to evaluate predictions made on test data, providing their accuracy and F1-Score.

To deal with the classification task, we develop different Machine Learning models starting from more classic methods up to the implementation of complex Neural Networks. The baseline is implemented with Scikit Learn [3], while concerning Deep Learning, we make use of a Recurrent Neural Network (RNN) based on GRU (Gated Recurrent Units) [4] and, subsequently, BERT [1], a fine-tuned pre-trained model provided by Google AI Language and based on Transformers [5]. These Deep Learning models are implemented using TensorFlow [6], with Keras [7] as a wrapper.

II. Exploratory Data Analysis

The tweets are organized in 2 labeled datasets: a small one, consisting of 200,000 tweets and a larger one of 2,500,000 tweets. The datasets are somehow balanced: there are the same number of tweets from both classes. However, after further investigation, we find that the files contain *duplicates* - same tweet and label multiple times - which might introduce skewness and bias in the data. Dropping these duplicates, we have 1,142,838 negative examples and 1,127,644 positive ones (larger datasets). They are still close enough to treat them as balanced classes. A key part of any Natural Language Processing task is data pre-processing. So, before applying any binary classification algorithm, we explore the data to understand what text pre-processing techniques to apply and what text representation to use. Analyzing the labeled data, we discover that:

- **Technical details**: Tweets have a maximum of *140 characters*.
- **Language**: They are comprised of mostly *english words* and *slangs* which may or may not have spelling errors.
- **Tags**: Most tweets have <*user*> and <*url*> tags to substitute an user or a reference to an external link. However, there are other tags (words in angle brackets) but they are significantly scarcer than the 2 mentioned before. For example, in the extended dataset, the first 3 most common tags are: <*user*> (1,605,595 times), <*url*> (526,862 times) and <*b*> (only 27 times).
- **Hashtags**: The larger dataset has 114061 unique hashtags (ex: *#ilc2012, #sofunnymom, ...*) which may contain multiple words, slangs and spelling mistakes.
- **Emoticons**: In writing, Twitter users sometimes use emoticons (ex: :p, <3, (*_*), ...) to accentuate an idea. Using the List of emoticons from Wikipedia [8] and merging with other common ones we knew, we find 132 unique emoticons present in the dataset.
- **Tweet endings**: We notice that some tweets end in 3 dots (...) or in a series of repeated parentheses, either) or (.

III. First attempt with classic NLP - Baseline

A. Preprocessing

Applying a non-Deep-Learning model requires a careful and extended pre-processing. Moreover, the order in which the text is modified matters as, for example, removing punctuation before analyzing emoticons might delete them. As a preprocessing pipeline for applying classic NLP techniques, we choose to firstly *remove tags* such as <*user*> or <*url*> and split *hashtags* in actual words (#sofunnymom becomes so, funny, mom). We use SymSpell [9], a library which implements an algorithm for spelling correction and word segmentation. Secondly, we convert *slangs* to words using a list of common english slangs adapted from the code of an online blogpost [10]. To deal with *emoticons*, we remove spaces between them (introduced either by the user or by a

previous tokenization) and add a space between emoticons and the rest of the text if there was none. Our reasoning is that some emoticons are appended at the end of a word: *hello:)*. Only after this extensive preprocessing we detect the emoticons by using the list mentioned in Section II and replace them with GloVe specific categories (smile, lolface, heart, neutralface, sadface) [11]. Moreover, following our observations about repeated final parenthesis, we transform those as well in either smile or sadface. At the end, we remove all *numbers* and *punctuation*, convert the text to lower case letters only, perform a *spelling correction* using the before mentioned SymSpell's algorithms and *lemmatize* the text by using nltk library [12] with part of speech tagging (PoS). Then we remove all *stopwords*. We notice that at the end of this preprocessing pipeline, some tweets end up being empty (ex: <user> theres no b?). Instead of leaving them blank, we add a tag which marks them as <EMPTY>.

B. Tweet representation and feature engineering

To represent the tweets and feed them in ML models, we use the TF-IDF representation. Since this is computationally expensive, we are building it using only the small dataset. To further reduce the dimensionality of the representation (the entire vocabulary has 30,282 words) we apply Latent Semantic Analysis, a statistical approach based on the idea that words used in similar contexts are similar in meaning [13]. It applies singular value decomposition on the TF-IDF matrix with a specified desired output length, resulting in having a final vector of dimension 500 for each tweet. Along the above representation, we add some statistics about the Part of Speech used in the tweet. For example, verbs, adjective and adverbs are most commonly used to express emotions, however some nouns too. Following a widely used approach [14], for each tweet we use a weighting scheme in which we count how many words are in each category and multiply it with the strength of the category: *emotion* - adjective, adverb and verb - with strength *2*, *normal* - noun - with strength *1.5* , *remain* - everything else - with strength *1*. The last feature we use is the output of another classifier, Vader [15], which performs sentiment analysis on tweets based on modelling rules on lexical features with respect to expressing sentiments in text. Another classifier, TEXTBLOB [16], which takes a lexicon approach by assigning sentiment polarity to words and aggregating them, was also considered. Vader is specifically trained for social media or mini blogs texts such as our tweets, so we feed in the raw tweet without any preprocessing.

As a recap of this section: each tweet is represented by a total of 504 features.

C. Models fittings

We choose to train 6 classic classifiers: K-Nearest Neighbors, Naive Bayes Classifier, Logistic Regression, SVM with linear kernel, Random Forest and a fully connected Multi-Layer Perceptron with one hidden layer. The hyper parameters are tuned using a grid search with 5-fold cross validation scoring the accuracy of the models. To implement this, we heavily use Scikit Learn and parallelization. The most important hyper parameters we find for each model are (for an extensive list consult the code): KNN [$K = 7$], Naive Bayes Classifier [$var\_smoothing = 1e-12$], Logistic Regression [$C = 0.00398$ (regularization strength)], SVM linear [$C = 0.06309$ (regularization strength)], Random Forest [$n\_estimators = 1000$ (number of trees)], Multi-Layer Perceptron [$alpha = 0.25118$, $hidden\_layer\_size = 5$ (regularization term and size of the hidden layer)].

IV. Gated Recurrent Unit

A GRU is a gating mechanism in Recurrent Neural Networks. Neural Networks belonging to this family can be thought of as a set of networks linked to each other, where connections between nodes form a directed graph along a temporal sequence, allowing them to solve problems for which the desired output depends on inputs at times far in the past (i.e. long-term dependencies). RNNs proved to be particularly suitable for complex tasks, such as the one we are facing, but considering only a standard RNN is not enough, since they tend to suffer from the vanishing gradient problem [17]. That is why gating mechanisms such as GRU and LSTM (Long Short Term Memory) [18] have been proposed. These mechanisms, indeed, allow gradients to propagate more easily through the structure of the Neural Network, using gates to directly encode the long-distance information (fig. 1). We choose GRU over LSTM because it turns out to achieve similar results with a smaller training time [19].

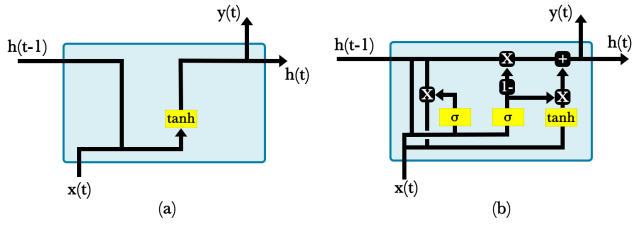

Figure 1. (a): a standard RNN cell. (b) a Gated Recurrent Unit. The symbol (X) represents a point-wise multiplication, (+) a point-wise addition. The junction of two lines represents a vector concatenation. Activation functions are reported in the yellow boxes. X(t) represents the input at time t, H(t) the status of the cell and Y(t) the output.

A. Model structure

The model we propose consists of 4 layers:

- **Embedding**: for this type of task, it is often convenient to use embedding pre-trained layers. We use 100-dimensional word vectors, pre-trained on Twitter data and obtained with GloVe (Global Vectors for Word Representation), provided by Stanford University

[11]. The total number of word vectors in the file is 1,193,514.

- **Bidirectional GRU**: a GRU bidirectional layer, with 100 units, a dropout of 0.2 and a recurrent dropout of 0. With respect to the activation functions, we followed the guidelines provided by the Keras implementation of GRU [20], using a *hyperbolic tangent* as the first activation function, and *sigmoid* for the recurrent steps.
- **Dense**: a densely connected layer, consisting of 100 units with a *Rectified Linear Unit (ReLU)* activation function, followed by a one-unit dense layer, with a *sigmoid* activation function, that provides the probability for the output to have label 1 (positive).

As model parameters, we use an Adam optimizer [21], with learning rate set to 0.001. The optimization is done on the accuracy of the model and the loss function is set to be binary cross-entropy. We train the model for 10 epochs on the largest dataset, with a batch size of 128, using a 80/20 split between the training and the validation data.

B. Preprocessing

Since we use GloVe, it is necessary to comply with the guidelines that Stanford University issued about preprocessing because the word vectors were obtained on a corpus preprocessed in the same way. In particular, different situations can occur within a tweet and each of them is encoded with a specific tag [22]. We report just some examples for brevity:

- Elongated words are converted in *normal_word* <*elong*>. E.g. *hellooo* to *hello* <*elong*>.
- Emoticons are replaced with tags. E.g. <*3* to <*heart*>.
- Punctuation repetitions are converted in *single_symbol* <*repeat*>. E.g. *!!!* to *!* <*repeat*>.

We keep the two tags <*user*> and <*url*> in the tweets, as they are recognized by the embedding. To have a fixed size for each tweet they are right-padded with 0s after the tokenization, to reach a maximum number of 120 words per tweet. Tweets longer than that are cropped. As mentioned before some tweets present three dots at the end of them, meaning they have been cropped before being put in the dataset. We remove this punctuation as it does not carry any information. After dealing with this pattern, we focus on the tweets that end with repeated parentheses: when a tweet ends with one or more "*(*" or "*)*" we replace them with the corresponding emoticon tag as done in the baseline. With this pre-processing, we obtain a vocabulary of 439,822 words. 172,879 of these are correctly recognized by GloVe, while, unfortunately, 266,943 are missed, and thus represented as a zero-vector, exactly as each padding token. This ratio is due to the numerous spelling and grammatical errors present in the dataset. We try to apply the same correct spelling algorithm used for the baseline, but given the high number of rows in the dataset, this turns out to be not feasible in terms of performance.

V. BERT

BERT (Bidirectional Encoder Representation from Transformers) is a pre-trained language representation model developed by Google AI Language [1] and based, as expressed by its name, on a Transformers architecture [5]. The strength of a this architecture is that every word is represented by a vector, just like in a normal embedding, but this vector changes with respect to the context of the word itself. This leads to different representations of the same word according to the meaning of the entire tweet (fig. 2). There exist many versions of BERT, categorized according to the number of trainable parameters, and every version has been pre-trained on a large corpus of English text by masking 15% of the tokens with the goal to guess them. Although our task is not the same, results show that BERT is also suitable for text classification, often reaching the state-of-the-art [23].

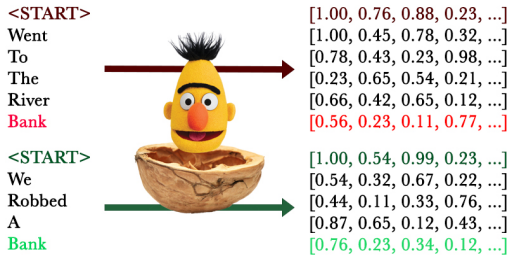

Figure 2. BERT in a nutshell [24]

A. Model structure

Firstly we choose to fine-tune the pre-trained model *bert-base-uncased* as a trade-off between the number of trainable parameters (110M) and expected accuracy. Every test is conducted on a Virtual Machine from Google Cloud Platform with a *dual-core* CPU, *13 GB* of RAM and a *NVIDIA Tesla P100* as GPU. The model is implemented using the official library provided by Hugging Face [25]. As global parameters, we use a variable learning rate that increases linearly from 0 to 2e-5 during the first 10% of the total training steps (warm-up phase), decreasing then linearly back to 0 during the remaining ones. As optimizer we use the Adam implementation provided by the library, clipping the norm of the gradient to 1 to avoid exploding gradient problem. Even if our task falls under the category of binary classification, BERT cannot work with a binary cross-entropy loss function, thus we use a more general sparse categorical cross-entropy. As evaluation metric, as before, we consider the accuracy. The batch size is set to 24. This increases the training time of the model, with respect to higher batch sizes, but also its accuracy. The ratio between training and validation data is set to 90/10 because, as said before, each word is represented differently according to its context, and so it is important to have as many training datapoints as possible. On the other hand, we also need

a considerable number of validation data to infer about the training phase. With respect to the number of epochs, after many tests we notice that the more the data is pre-processed, the sooner the model tends to overfit. After some tests with this configuration we move to *bert-large-uncased*, with 336M of parameters, to improve the model accuracy.

B. With different preprocessings come great responsibilities

1) Our first attempt: Our first idea is to tune BERT on well pre-processed data. Knowing that it has been trained without using the special tags of GloVe, we start by removing the <*user*> and <*url*> tags, then we delete the three dots at the end of the tweets. After this step, we use the same emoticons categorization used in GRU, but, instead of using tags when replacing the symbols, we use normal English words, such as: *sad*, *lol*, *neutral*, *smile*, *heart*. We do the same for the final parentheses, replacing them with the word corresponding to the inferred smiley. We then convert elongated words as done in GRU but without adding the related tag. As last step, we remove the remaining symbols in this list: [$&+=@#|<>:*()%-]. We do not remove exclamation and question marks as they can be good indicators of the mood of a person. In the end, we make sure that every tweet is lowercase, as requested by the model.

2) Pre-processing the data: a downside?: This preprocessing seems good for classic models, but there is a downside that needs to be considered: a significant amount of information is lost. At this point, instead of trying to perform a more advanced data manipulation, we do the opposite: we train a model on a dataset on which we only remove the <*user*> and <*url*> tags, the three dots at the end of the tweets, replacing then the final parenthesis with the corresponding smiley but keeping the emoticons as they are, together with all the other symbols. With this smaller pre-processing we achieve better results (Section VI). We try to increase the number of training epochs from 2 to 3, but overfitting becomes visible for the third. We do not suggest, then, to train BERT for more than two epochs with this configuration. At this point we start training the *large* model on the same data, reducing the number of epochs back to 2.

VI. Results and discussions

A. Baseline

Table I shows the results obtained by applying classic NLP to train 6 standard classifiers. Not surprisingly, the best model is the Multi-Layer Perceptron, however its performances are similar to the Linear SVM, Random Forest and Logistic Regression ones. We remind that these results are obtained by training on the smaller dataset: meticulously applying the same preprocessing techniques to the entire dataset is computationally expensive and would not greatly improve the results.

Table I
Validation and Test scores. Baseline models.

Model	Validation acc.	AIcrowd	
		Acc.	F1 Score
KNN	65.6%	67.4%	65.1%
Naive Bayes	64.1%	63.0%	59.8%
Logistic Regression	76.4%	73.9%	73.4%
Linear SVM	76.5%	73.7%	73.2%
Random Forest	76.5%	77.1%	77.9%
Multi-Layer Perceptron	77.6%	78.1%	78.7%

B. Advanced models

Table II shows the results obtained with GRU and the various BERT models tested. It is interesting to notice that for these NN less data pre-processing is necessary, and it even turns out to be deleterious in some cases. We also report the result of the weighted voting among the four DL models, and among all the models, baseline included. The weight for each model is given by its validation accuracy.

Table II
Validation and Test scores. DL models and weighted voting.

Model	Validation acc.	AIcrowd	
		Acc.	F1 Score
GRU (GloVe Embedding)	85.3%	85.6%	85.8%
BERT base (advanced prepr.)	88.8%	89.1%	89.1%
BERT base (basic prepr.)	89.4%	89.6%	89.7%
BERT large (basic prepr.)	89.7%	90.2%	90.1%
Weighted voting (DL models)	/	90.1%	90.2%
Weighted voting (all models)	/	85.7%	85.9%

AIcrowd accuracy is always higher than the validation one because of the duplicates in the test dataset: labeling correctly one of them means guessing also the others. Regarding the weighted voting, when we include the baseline models the accuracy drops significantly, while, when considering only the DL models, it is comparable to the best one.

VII. Conclusion

In this paper, we build a baseline methodology based on a classic NLP approach and simple ML models which require extensive data pre-processing, attention to details and feature engineering. To improve on them we shift completely and turn to DL models which are computationally more expensive but yield better results. Moreover, we experiment with weighted ensembles relying on the validation score without a significant improvement. Our best model is based on transfer learning with BERT, achieving 90.2% accuracy and 90.1% F1 score on the test set. As future work, one can extend on our ideas by continuing fine tuning BERT with multiple Twitter datasets or, since the field of DL in NLP is continuously expanding, one can shift the view altogether and use other pre-trained language models such as XLNet [26] and GPT-2 [27].

REFERENCES

[1] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[2] (2020) EPFL ML Text Classification. École Polytechnique Fédérale de Lausanne, Switzerland. [Online]. Available: https://www.aicrowd.com/challenges/epfl-ml-text-classification

[3] P. F. *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[4] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," 2014.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017.

[6] M. Abadi *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, 2016, pp. 265–283. [Online]. Available: https://www.usenix.org/system/files/conference/osdi16/osdi16-abadi.pdf

[7] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[8] (2020) List of emoticons. [Online]. Available: https://en.wikipedia.org/wiki/List_of_emoticons

[9] Symspell. [Online]. Available: https://github.com/wolfgarbe/SymSpell

[10] R. Verma. Python script to turn text/message abbreviations into actual phrases. [Online]. Available: https://github.com/rishabhverma17/sms_slang_translator

[11] 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

[12] E. Loper and S. Bird, "Nltk: The natural language toolkit," in *In Proceedings of the ACL Workshop on Effective Tools and Methodologies for Teaching Natural Language Processing and Computational Linguistics. Philadelphia: Association for Computational Linguistics*, 2002.

[13] P. Foltz, "Latent semantic analysis for text-based research," *Behavior Research Methods*, vol. 28, pp. 197–202, 02 1996.

[14] Chris Nicholls and Fei Song, "Improving sentiment analysis with part-of-speech weighting," in *2009 International Conference on Machine Learning and Cybernetics*, vol. 3, 2009, pp. 1592–1597.

[15] C. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," 2014. [Online]. Available: https://www.aaai.org/ocs/index.php/ICWSM/ICWSM14/paper/view/8109

[16] S. Loria, "textblob documentation," *Release 0.15*, vol. 2, 2018.

[17] R. Pascanu, T. Mikolov, and Y. Bengio, "On the difficulty of training recurrent neural networks," 2013.

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

[19] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio, "Empirical evaluation of gated recurrent neural networks on sequence modeling," 2014.

[20] Gru layer. [Online]. Available: https://keras.io/api/layers/recurrent_layers/gru/

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.

[22] R. Paulus. Script for preprocessing tweets - GloVe. [Online]. Available: https://nlp.stanford.edu/projects/glove/preprocess-twitter.rb

[23] Bert used for sentiment analysis, Tensorflow. [Online]. Available: https://www.tensorflow.org/tutorials/text/classify_text_with_bert

[24] (2020) EPFL ADA course - lecture 11. École Polytechnique Fédérale de Lausanne, Switzerland - DLab. [Online]. Available: https://dlab.epfl.ch/teaching/fall2020/cs401/

[25] (2020) Hugging Face - BERT library. huggingface. [Online]. Available: https://huggingface.co/transformers/model_doc/bert.html

[26] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds., vol. 32. Curran Associates, Inc., 2019, pp. 5753–5763. [Online]. Available: https://proceedings.neurips.cc/paper/2019/file/dc6a7e655d7e5840e66733e9ee67cc69-Paper.pdf

[27] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever, "Language models are unsupervised multitask learners," 2019.