Text Sentiment Classification

Marie Biolková **Sena Necla Çetin** **Robert Pieniuta**
Department of Computer Science, EPFL, Switzerland
{marie.biolkova, sena.cetin, robert.pienuta}@epfl.ch
AIcrowd Team: gLove_Actually

***Abstract*—The goal of this project was to build a classifier for predicting the sentiment of a given tweet, i.e. whether it included a positive or negative smiley, from the remainder of the tweet. Consulting recent relevant literature, we experimented with preprocessing, various word representations (including self-trained embeddings), classical machine learning models, as well as neural network architectures. The best result was obtained using an ensemble of recurrent neural networks and fastText models with pre-trained GloVe embeddings, yielding an accuracy of 88.6% on test data and $F1$-score of 88.8% – an improvement of 4.4% over the baseline support vector machine with TF-IDF.**

I. Introduction

Sentiment classification is a task in natural language processing (NLP), in which the goal is to assign a label – positive, neutral, or negative – to text by detecting emotions or opinions expressed by its contents. Although an abundant and valuable resource, text represents unstructured data, making it difficult to automate the process of information extraction. Such information can be a great value to businesses and other entities who can use it to motivate their decisions (such as marketing strategies) and improve their services, for instance through more efficient customer support. Nowadays a lot of communication is done online where the form of text can be very different from general language. In particular, social media data are very challenging to process due to the presence of slang, emojis, initialisms, neologisms and other out-of-vocabulary forms. Messages are often sent as an immediate response to another message which increases the risk of typos. Thus, social media data are often of great interest in NLP, both due to the utility of the information that can be retrieved from them and due to the challenge it represents for automatic processing of text.

In this study, we will focus on the inference of sentiment from a corpora of tweets. The tweets were assigned a binary label depending on whether they contained a positive smiley :) or a sad smiley :(, before the smileys were removed from the tweet's body. The goal is to predict a sentiment of the message based on the text without smiley, which makes our task an example of a typical machine learning classification problem. We will explore a variety of approaches to text representation and a multitude of models, from traditional classifiers to more complex ones, including neural networks. We will also try to improve the accuracy by performing various preprocessing steps as these could help reduce noise and the dimensionality of the vocabulary.

In the following, we start by data exploration and description of preprocessing (Section II) . In Section III we describe the choice of features for the classification problem. Then, we introduce the models considered in Section IV and present the results in Section V. We conclude in Section VI by discussing the performance of the models and the word representations used, as well as the relevance of data preprocessing for this task.

II. Data description and preprocessing

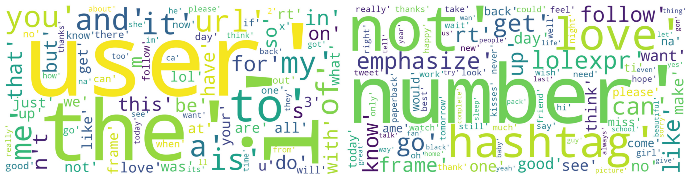
Fig. 1. Word clouds of the most common words created before (on the left) and after (on the right) preprocessing.

The full dataset contained 2.5M tweets, with equal representation of the positive and negative classes. Due to the amount of the set and resource constraints, hyperparameter tuning of the baseline models was carried out on a development set of 200,000 tweets. The model was subsequently re-trained with the optimal parameters and evaluated on the AIcrowd test set.

While some of the modern approaches to text processing prefer to work with raw data, a careful preprocessing can help to emphasize the sentiment of a tweet as well as reduce noise and the dimensionality of the vocabulary. We decided to perform the following preprocessing steps in the attempt to improve learning, and to contrast the performance with results on the raw data. Note that the tweets have already been tokenized.

Auxiliary tags: As some tags (e.g. `<user>`, `<url>`) and abbreviations (e.g. *rt* derived from the word *retweet*) appear in a great number of tweets and do not contain an emotional message as such, we decided to remove them.

Contractions: We found useful to expand the contractions existing in informal writing (e.g. *isn't* $\rightarrow$ *is not*) to recover their sentiment.

Numbers: In order to unify the information about the presence of numbers in the tweets, we have decided to replace them all by the `<number>` tag.

Emojis: A popular way to indicate the sentiment of a message is to use emojis. As they can occur in many variants, it is impossible to process them all. We combined some of the emojis in one tag to indicate a similar meaning. For example, emojis :d and :p were replaced by `<laughface>`.

Stopwords: One of the most popular ways to reduce the noise in the messages is to eliminate so-called 'stop words', e.g. 'a', 'an', 'the', 'or', 'then', etc. Usually they do not indicate the sentiment of a sentence, so with a subtle approach they can be removed. However, we have decided to preserve some of these, including those that convey denial, believing that they carry valuable information about the emotional character of a tweet.

Repeated punctuation: We decided to extract repeated punctuation from tweets, keeping one occurrence only and adding the tag `<emphasize>`.

Extended words: In order to reduce the dimensionality of the space, we replaced the multiple occurrences, i.e. more than 3 times, of a character by a single character.

Hashtags: Although hashtags may convey sentiment (e.g. '#worstdayever'), splitting these words accurately and efficiently can be a difficult task. Therefore, we decided to remove the '#' symbols and replace them with the tag `<hashtag>`. This indicates the presence of a hashtag and may inform of the sentiment of the words it contained.

Laughs: We merged different variations of laughs (e.g. hahaha, ahaha, lol) into a single `<lolexpr>`.

Lemmatization: It is a process of grouping together the different inflected forms of a word so they can be analysed as a single item. It transforms words like 'tried' to 'try', 'worse' to 'bad', 'coats' to 'coat'.

Finally, we removed multiple whitespace occurences and filtered out single character tokens that have not yet been processed, excluding the exclamation and question marks.

During data exploration, we noticed that negative tweets tend to be longer. This property was preserved during preprocessing, which reduced the vocabulary size by $\approx 25\%$.

III. Features

To use text as input to machine learning algorithms, one needs to at first find numerical representations for its components (words or tokens) which can be used as features. In this section, we present techniques commonly used in NLP that we decided to apply to our dataset.

Bag-of-words: As the simplest method for representing words, we used the bag-of-words model which extracts features from text by counting the frequency of words in a document. Each sentence is represented by a frequency vector that summarizes the number of occurrences of each word in a tweet. While this method is extremely simple to use, it has a number of key drawbacks which prevents it from achieving better results. First of all, based solely on the frequency of words, it completely fails to recognize the context of a message. Secondly, similarities between words are lost. Lastly, the most common words such as 'the', 'a', 'an' become the most important ones, which does not improve sentiment prediction ability.

TF-IDF: To address the problem with domination of most popular words in BoW method, one can give less weight to tokens that occur in many tweets. Penalization of common words is the key idea of Term Frequency - Inverse Document Frequency (TF-IDF). In this project, we investigated four different n-grams, where $n \in \{1, 2, 3, 4\}$ of both, words and characters. We found that the character analyzer, using those four types simultaneously generally worked the best.

BoW and TF-IDF methods result in the creation of a very high-dimensional, sparse (with most of the elements equal zero) feature spaces. This fact plays a key role in the selection of hyperparameters, which will be discussed later.

GloVe embeddings: In contrast, word embeddings provide a dense word representation in a vector space. They aim to capture the semantic meaning of the word, i.e. similar words should have similar representations. Global Vectors, or GloVe [1], is one of the most popular models for building word embeddings. It is trained using an unsupervised algorithm which factorizes the co-occurrence matrix in a way that minimizes the reconstruction loss. Two options were considered:

1) Self-trained GloVe embeddings: Using the pre-processed data, we trained 200-dimensional GloVe embeddings with stochastic gradient descent (SGD) and adaptive learning rate for 10 epochs. Index terms with a frequency less then 5 were not considered when constructing the co-occurrence matrix. For simplicity, we also omitted scalar bias terms from the loss function – an investigation of whether their inclusion significantly improves the performance might be explored in the future. The learning rate was updated according to the Adagrad algorithm [6] which takes into account the gradient at previous iterations, with initial value 0.01. Note that overfitting is generally not considered as an issue during embedding training as the goal is to get solid representations rather than make predictions.

2) Pre-trained GloVe embeddings: Another possibility is to use embeddings that were pre-trained on large corpora and could therefore provide better word representations. Pennington et al. (2014) provide GloVe embeddings trained on large corpora, one of which was a corpus of 2B tweets. We opted for the 200-dimensional version.

An alternative word embedding model is word2vec [5]. Rather than using the co-occurences, it trains a shallow neural network with the words as input. The network can either predict a word from a context (CBoW) or predict the context of a given word (skip-gram). However, we decided not to include this approach in our experiments.

When mapping the data to embeddings, unknown tokens might be encountered. In such cases, we assign them the <unk> tag and the corresponding embedding, which is the mean of all remaining embeddings combined. The downloaded embeddings already contained the pre-trained <unk> tag.

Many of the algorithms we considered can only accept fixed-length input, so once we have obtained the word vectors, we need to find a way to represent tweets which may have different lengths. To do this, we simply take an average of the embedding vectors of its components.

***n*-grams:** As mentioned, the bag-of-words representation is invariant to word order. However, word ordering plays a significant role in the meaning of sentences, though taking the whole order into account can be computationally very expensive. The n-grams model makes the trade-off between efficiency and localization by taking only the previous n words into account, while computing the probability of the occurrence of a word [7].

Our final model, fastText, uses bag-of-words representation and n-grams as additional features to capture some partial information about the local word order while being very efficient [2]. The hashing trick [9] utilized in fastText allows n-grams to be mapped fast without being memory-intensive [2].

IV. Models

Baselines: Several commonly used models were considered and used as a reference point when evaluating more complex models. These included random forests, which calculate the average prediction of individual trees, so-called weak learners, logistic regression, a natural choice for binary classification, and support vector machines (SVMs), a popular alternative to logistic regression. The hyperparameters were tuned on the development set using grid search and a 80:20 training-validation split. To accelerate hyperparameter tuning for SVM and logistic regression, we used stochastic gradient descent. We restricted SVMs to the linear kernel. However, other (nonlinear) kernels might be also suitable. This was mostly due to the limited computational capacity – despite using only kernel approximation only, the model took a lot of memory to fit when generating predictions from the full data. We also considered implementing Naïve Bayes classifier but decided that the assumptions of feature independence were too strong to provide a good prediction.

Recurrent Neural Networks: In recent years, deep learning methods, such as multi-layer perceptron (MLP), convolutional neural network (CNN), and recurrent neural network (RNN), have become the main focus for NLP applications, achieving state-of-the-art performance. While the two former methods are very popular, they accept fixed-size inputs. However, in many NLP tasks, the input sizes may vary (e.g. due to different text lengths) and one has to tackle such situations by using padding. However, RNNs were specifically designed for handing sequential data, which makes them the perfect candidate for problems that make use of text. The reason why they are well-suited for sequences is because they contain cycles, which allows them to retain important past inputs in memory, thereby to capture context. Unfortunately, RNNs struggle with vanishing gradient when dealing with long sequences, and thus suffer from short memory. We therefore considered two variants that address this problem by introducing gates into the architecture – these can determine which parts of the sequence are important and which can be forgotten, hence extending the network's memory.

1) GRU: The Gated Recurrent Unit (GRU) consists of two gates – reset and update. We implemented the bidirectional version – passing the inputs both forward and backward allows the network not only to remember the past, but also to obtain information from the future inputs. This typically leads to better performance. The dimension of the GRU layer was adapted to the input features which were the 200-dimensional GloVe embeddings. We experimented with one and two-layer GRU networks followed by a single-layer feed-forward network and tried multiple hidden layer dimensions (between 50 and 250 per direction). The dropout rate was set to 10% as the model was not very complex and did not show any signs of overfitting.

2) Long Short-Term Memory (LSTM): LSTMs have three gates (input, output and forget) and an additional memory unit. So, theoretically, they should outperform GRU on longer sequences, but might be less efficient due to this increased complexity. We opted for an identical architecture as for GRU.

We implemented these methods in PyTorch [10]. Both models were trained using the cross-entropy loss with Adam optimizer [8]. We opted for batch-size equal to 32.

fastText: Although deep neural networks can achieve superior performance compared to classical machine learning models, they are computationally very expensive due to their high number of parameters. fastText is an alternative classifier, developed by Facebook research, that allows to learn text representations and to do text classification using linear models with a rank constraint and a fast loss approximation [2]. It achieves performance similar to neural networks while being much faster [2].

To maximize the accuracy of the fastText classifier, we performed tuning of the learning rate and of the length n of n-grams. We experimented using both raw and preprocessed data while keeping in mind that neural networks are generally negatively affected by preprocessing since they benefit from having more data to learn from.

FastText also provides an automatic hyperparameter optimization feature that automatically optimizes the hyperparameters by the precision and recall measures on the validation data. Therefore, we considered it as an alternative to classical tuning. Furthermore, we tried a variety of ensemble fastText models on n-grams, with $n \in \{2, 3, 4, 5, 6\}$, and the automatically optimized model using majority voting.

Ensembles: Having trained a wide range of models, we naturally wanted to test how well these would perform if their predictions were combined. We decided to take the average of prediction probabilities from the best 3 models. While there are other, more sophisticated approaches to ensemble learning (such as bagging [3] or boosting [4]), this method conveniently did not require any further optimisation.

V. Results and Discussion

The results of our experiments for classical and deep models (including fastText) are collected in Table I and II, respectively.

First of all, as the common language used in social media is full of errors of various kinds, it feels appropriate to subject it to far-reaching preprocessing. However, instead of emphasizing the sentiment, it seems that our preprocessing caused a significant loss of information. We suspect that lemmatization must have been a particularly crude step, and also the numerous filter we applied might have drastically reduced the length of tweets, leaving little data for inference.

TABLE I
PERFORMANCE OF THE BASELINE MODELS USING DIFFERENT FEATURES AS INPUT IN TERMS OF ACCURACY ON THE TEST SET. THE VALUES IN PARENTHESES CORRESPOND TO ACCURACY OBTAINED ON PREPROCESSED DATA.

Features / Model	Bag-of-words	TF-IDF	GloVe (pre-trained)	GloVe (self-trained)
Logistic Regression	80.6 (79.6)	83.6 (80.0)	77.4 (76.4)	68.7 (68.0)
Random Forrest	73.5 (73.3)	79.0 (75.8)	77.7 (77.4)	72.9 (72.8)
SVM	80.9 (79.5)	84.2 (80.3)	77.9 (77.0)	69.4 (69.2)

It would be interesting to see whether a less invasive approach would have done any better. Even subtle changes tend to make the predictions worse, so one needs to empirically check the efficiency of any modification.

Without preprocessing and using four different word representations with the baselines, we quickly achieved results that exceeded 80% accuracy. Neither logistic regression nor SVC required regularization, suggesting that the models were robust (and could potentionally be made more complex while avoiding overfitting). We also noticed that the optimal regularization constant was higher for the dense embeddings than for sparse vectors (although it was still in the order of 10^{-3}). The baselines did particularly well with the Tf-Idf – in fact, logistic regression and SVMs outperformed 2 out of 3 fastText models when this weighing scheme was applied.

It is not surprising that the self-trained GloVe embeddings did not do as well as the pre-trained ones. Indeed, in order to learn how to capture sematic relationships, they require large volumes of data and computionally expensive training, neither of which we had. Furthermore, the self-trained embeddings were fit to the pre-processed data with the intention to speed up computation and improve their performance at the same time. As the pre-trained embeddings were significantly better and training neural networks is computationally expensive, we only used these embeddings as features for RNNs.

fastText enabled us to reach high accuracy without compromising efficiency. The best manually optimized classifier (*accuracy* = 86.2%) was obtained using raw data with hyperparameters $lr = 0.1$, *n-grams* = 3, and *epochs* = 2. Moreover, fastText's automatic hyperparameter optimization feature improved our manually optimized model by 0.2% and gave us the best result (*accuracy* = 86.4%) achieved by a single fastText classifier. The ensemble of *n*-grams and the automatically tuned model improved our accuracy by only 0.1% (*accuracy* = 86.5%) since these models are very similar in their predictions. Note that fastText gives slightly different results at each run due to the utilized optimization algorithm, i.e. asynchronous stochastic gradient descent [2].

Finally with RNNs, we were able to achieve a much higher accuracy. Upon a brief exloration to verify our hypothesis that bidirectional LSTM and GRU are far more powerful, we found out that the there is also not much need for dropout – in fact reducing it to 1% made the 2-layer LSTM our best-performing (single) classifier. The validation loss was persistently lower than the training loss, so one might think about further increasing the complexity of the network. Unfortunately, this would not be feasible given the constraints on the GPU we had. It would be tempting to try stacking more LSTM or GRU layers – for instance Google made use of 8 LSTM layers for their Google Neural Machine Translation (GNMT) tool [11].

Model	Specifications	Accuracy (%)
fastText	0.1 learning rate, 2 epochs	83.2
fastText	0.1 learning rate, 2 epochs, preprocessed data	82.9
fastText	0.1 learning rate, 2 epochs, 3-grams, auto. opt.	86.4
RNN	1 LSTM layer, 50 hidden units, 0.1 dropout	87.1
RNN	2 LSTM layers, 250 hidden units, 0.1 dropout	87.9
RNN	2 LSTM layers, 250 hidden units, 0.01 dropout	88.2
RNN	1 GRU layer, 50 hidden units, 0.1 dropout	86.7
RNN	2 GRU layers, 250 hidden units, 0.1 dropout	87.1
Ensemble	best fastText, LSTM and GRU above	88.6

TABLE II
PERFORMANCE OF THE NEURAL NETWORK-BASED MODELS IN TERMS OF ACCURACY ON THE TEST SET.

The best accuracy overall was achieved by the ensemble of 3 of the best-performing models, one of each of the following type: fastText, LSTM and GRU. Ideally, we would have liked to include many more models in the ensemble and train them specifically to be part of an ensemble.

As the field of natural language processing is evolving fast, there are more sophisticated methods to be considered, such as BERT and ELMo to name a few. For instance, ELMo does not make use of a fixed embedding for each word but instead scans the entire sentence before assigning an embedding to a word. This way, it can tackle the problem of polysemy [12].

VI. CONCLUSION

Performing textual analysis on corpora involving language used online, therefore being full of peculiarities and errors, is not straightforward. We apply a variety of classification models with diverse word representation schemes. The results showed that that TF-IDF is a reliable and convenient method for word encoding, allowing to achieve relatively high accuracy with baseline classification models, including logistic regression and support vector machines. FastText with automatic optimization using 3-grams yielded good performance with efficiency. As expected, recurrent neural networks yielded even better performance on the test set, in particular the LSTM architecture which has received a lot of attention in recent years. We were able to further lift our test accuracy to our ultimate best, 88.6%, by combining the probabilistic predictions of multiple models into an ensemble.

References

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

[2] Joulin, Armand and Grave, Edouard and Bojanowski, Piotr and Mikolov, Tomas (2016). Bag of Tricks for Efficient Text Classification. arXiv preprint arXiv:1607.01759.

[3] Breiman, L. (1996). Bagging predictors. *Machine learning*, 24(2), 123-140.

[4] Freund, Y., and Schapire, R. (1996). Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 148-156

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

[6] Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive Subgradient Methods for Online Learning and Stochastic Optimization. *Journal of Machine Learning Research*, 12, 2121–2159.

[7] Jurafsky, D. & Martin, J. (2008). Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition. (Ch. 3)

[8] Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980.*

[9] Mikolov, T., Deoras A., Povey, D., Burget, L., and Cernocky, J.. (2011). Strategies for training large scale neural network language models. *Workshop on Automatic Speech Recognition and Understanding.* IEEE.

[10] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. (2019) PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in neural information processing systems*, 8026-8037.

[11] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., & Klingner, J. (2016). Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144.*

[12] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., & Zettlemoyer, L. (2018). Deep contextualized word representations. *arXiv preprint arXiv:1802.05365.*

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