Preprocessing, Word-Vector Embedding, and Classification for Twitter Sentiment Analysis

Andrea Canonica
canandre@student.ethz.ch

Oliver De La Cruz
deoliver@student.ethz.ch

Claudio Röthlisberger
rclaudio@student.ethz.ch

Simon Scherrer
simonsch@student.ethz.ch

***Abstract*—The automatic analysis and interpretation of text is an active area of research in natural language processing (NLP) and machine learning (ML). We strive to contribute to the problem of *sentiment anlysis*, i.e. the identification of the general emotion expressed in a text.**

We developed a multi-step method for sentiment analysis of tweets [1], consisting of preprocessing, word-vector embedding, and classification. For every step, we borrowed concepts from related literature, combined them with self-devised approaches, and optimized the partial solution for one step at a time. Thereby, we were able to create a model that improved the classification accuracy by nearly 30 percentage points compared to the naive logistic regression baseline solution.

I. Introduction

Sentiment analysis is an intensely pursued research topic at the intersection of data science and computational linguistics. In general, sentiment analysis is understood as the deduction of an author's emotional characteristics on the basis of her text.

Research in sentiment analysis is fueled and facilitated by large amounts of available and easily usable data. One such source, which was used in this paper, are *tweets*. Twitter [1] provides large amounts of tweets labelled as either positive or negative, where the label is determined on the basis of emoticons in a tweet.

In this paper, we draw on 2.5 million such tweets to train a classification algorithm that aims at identifying positive and negative tweets, doing so in absence of emoticons. In advance of classification, we apply preprocessing and word-vector embedding methods to ease classification. The paper at hand also provides an evaluation of different such methods.

II. Method

This section explains the methods of sentiment analysis we designed in this project. Figure 1 illustrates the method steps. First, *preprocessing* is applied to the raw data files, which is described in Section II-A. Second, *word-vector embeddings* are produced (cf. Section II-B). Third and finally, a *classification* algorithm is trained to acquire prediction capabilities, as explained in Section II-C.

A. Preprocessing

The given dataset presents a set of challenges which have to be addressed in order to extract meaningful data. First, the constraint of 140 characters gives rise to a plethora of abbreviations and acronyms. Second, orthography is not observed. We evaluated several methods to address at least some of these issues (as did Bao et al. [2]).

[1]Messages posted on the social network service Twitter.

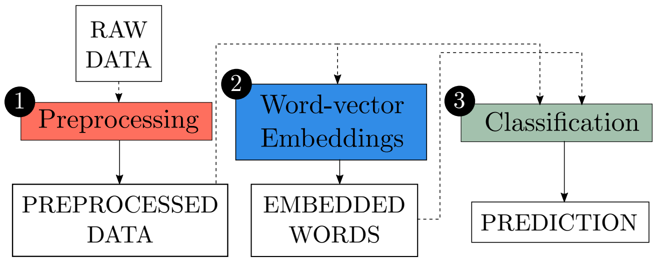

Figure 1. Illustration of method steps

All these methods aim to reduce the size of our dictionary. They either remove what we considered sentiment-neutral words, or they map differently spelled words onto one. Additionally, inspired by the papers [3] and [2] we implemented the methods *Same-Letter Sequence Removal* and *Negation Connection.*

1) Negation Replacement (N): Negation replacement substitutes common negated terms with their opposite expressions.

2) Word Cancellation (S): Word Cancellation removes frequent words that we assumed to be sentiment-free. Examples include: '*i*', '*you*', '*am*'. For the complete list, please contact the authors.

3) Tag Removal (R): Tag removal eliminates the tags *<user>* and *<url>* from every tweet.

4) Negation Connection (C) and Negation Disconnection (D): Negation connection implies that '*n't*' becomes '*nt*', whereas negation disconnection transforms '*n't*' into ' *not*'.

5) Short Word Removal (3): Short Word Removal removes a word if it consists of two characters or less.

6) Same-Letter Sequence Removal (M): If the word contains a letter more than twice in a row, that sequence is shortened down to two characters. For instance, occurrences like '*looól*' are shortened to '*lool*'.

7) Word Type Filtering (W): This method categorises every word and then keeps or discards it based on its type. Words were kept if they belonged to one of the types noun, verb, adverb, and adjective.

B. Word embeddings

On the basis of preprocessed files, *word-vector embeddings* are produced. Such embedding methods map each word in the vocabulary to a vector in a multi-dimensional space. Semantic relations between words should then be captured by geometric relations between the corresponding vectors, e.g., distance.

In this project, we used and evaluated two well-known embedding methods: *GloVe* [4] and *word2vec* [5]. Coarsely put, both methods work by adapting vectors iteratively such that the dot product of two word vectors is proportional to the probability that the two respective words occur in the same context.

C. Classification

Since a simple and effective approach to learn distributed representations of words was proposed [5], neural networks advance sentiment analysis substantially.

The first such network models, among which are Recursive Neural Network ([6], [7], [8]), Recursive Neural Tensor Networks ([9]), and Recurrent Neural Networks ([10]), performed sentiment analysis by utilizing syntax structures of sentences. However, such methods may suffer from syntax parsing errors. In addition, Recurrent Neural Networks (RNN) do not allow information to persist in the long-term. A common solution to this problem was given by LSTM (Long Short-Term Memory) networks, which contain special gates that control the information flow [11].

Since LSTM networks are capable of learning long-term dependencies, they are well-suited for natural language modeling. Furthermore, they are less prone to encounter gradient vanishing problems than RNN models. More formally, each cell in LSTM can be computed as follows:

$$X = \begin{bmatrix} h_{t-1} \\ x_t \end{bmatrix} \tag{1}$$

$$f_t = \sigma(W_f \cdot X + b_f) \tag{2}$$

$$i_t = \sigma(W_i \cdot X + b_i) \tag{3}$$

$$o_t = \sigma(W_o \cdot X + b_o) \tag{4}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot tanh(W_c \odot X + b_c) \tag{5}$$

$$h_t = o_t \odot tanh(c_t), \tag{6}$$

where $W_f, W_i, W_o, W_c \in \mathbb{R}^{d\times(n+d)}$ are the weighted matrices and $b_f, b_i, b_o, b_c \in \mathbb{R}^d$ are biases to be learned during training, parameterizing the transformations of the input, forget and output gates, respectively. σ is the sigmoid function and $\odot$ stands for element-wise multiplication. $x_t \in \mathbb{R}^n$ includes the inputs of LSTM cell unit, representing the word embedding vectors w_t (cf. Figure 2). The vector of hidden layer t is $h_t \in \mathbb{R}^d$, where d is the number of hidden units in a cell. Then different approaches can be applied to obtain a vector representation of a sentence. The last hidden vector h_N can be considered as the feature representation and also the average of the hidden vectors.

A fully connected hidden layer calculates the transformation $\delta(W_s h_N + b_s)$, where $W_s \in \mathbb{R}^{K,d}$ is the weight matrix, $b_s \in \mathbb{R}^K$ the bias, and δ an activation function such as the rectified linear unit or $tanh$. The output vector of this layer corresponds to logits that are normalized through a softmax function. Finally, the output returns the probabilities for each class, $y \in \{-1, 1\}$ and $K = 2$.

A potential issue with this setup is that a neural network needs to be able to compress all the necessary information of a sentence into a fixed-length vector that is fed to the fully connected layer. Inspired by Bahdanau et al. [12], regarding machine translation, we propose to implement an attention mechanism that does not attempt to summarize a whole text into a single vector. Instead, it encodes the input sentence into a sequence of vectors and chooses a subset of these vectors adaptively. Each time that model generates a prediction, it searches for a set of positions in the sentence where the most relevant information is concentrated. Hence, the vector representation is based on the concatenation of the last hidden unit and the context vector computed as weighted sum of all the hidden units. Let $H \in \mathbb{R}^{d\times N}$ be a matrix consisting of hidden vectors $[h_1, ..., h_N]$ that the LSTM produced, where z is the size of hidden layers for the attention mechanism and N is the length of the given sentence.

$$M = tanh(W_h \cdot H) \tag{7}$$

$$\alpha = softmax(w^T \cdot M) \tag{8}$$

$$r = H \cdot \alpha^T \tag{9}$$

$$h^* = tanh(W_p \cdot r + W_x \cdot h_N) \tag{10}$$

$$y = softmax(W_s \cdot h^* + b_s), \tag{11}$$

where $W_h \in \mathbb{R}^{z\times d}, W_p \in \mathbb{R}^{d\times d}, W_x \in \mathbb{R}^{d\times d}, w \in \mathbb{R}^z$ are the new matrices of parameters estimated by minimizing the cross-entropy error between y and $\hat{y}$ for all sentences.

Luong et al. [13] proposed an interesting modification method by categorizing various attention-based models into two broad categories, global and local. The global attention described above computes the scores or alignment (α) by taking into account all the hidden states from a sentence. On the other hand, the local attention offers a mechanism that focus on a smaller subsets of words. Compared to the suggested implementation stated in the original paper, we adapted the formulas to perform text classification tasks. In other words, the algorithm selects a window where the alignments decay exponentially around the center. More formally, our model predicts the midpoint of defined interval as follows:

$$p_t = N \cdot sigmoid(v_c tanh(W_c \cdot h_N)) \tag{12}$$

$$\alpha = softmax(w^T \cdot M) \cdot \exp\left(-\frac{(s - p_t)^2}{2\sigma^2}\right). \tag{13}$$

W_c, v_c are the model parameters which will be learned to predict positions. N is the source sentence length. As a result of sigmoid, $p_t \in [0, N]$. To favor alignment points near p_t, we place a Gaussian distribution centered around p_t.

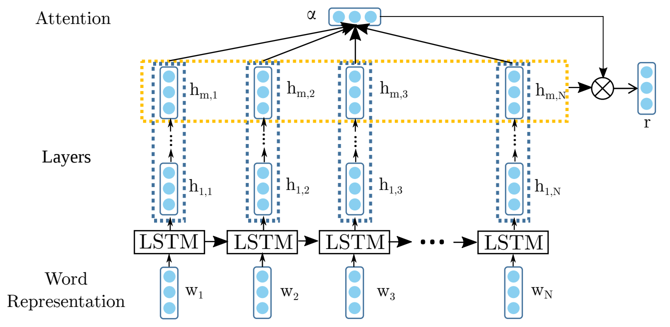

Figure 2. Illustration of global attention mechanism

The network parameters are learned using Adam optimizer [14]. We set the hyper-parameters to $\eta = 0.001$(learning rate), $\beta_1 = 0.9$, $\beta_2 = 0.999$ and we linearly decreased η whenever the loss function did not improve in the previous 10'000 iterations with a batch size of 64. In Section III, we will assess the accuracy of each of those different architectures.

III. Results

A. Baselines

We will use the following two algorithms as baselines throughout Section III:

- ***BL1:*** Logistic regression classifier, *GloVe* embeddings produced from the programming assignment script, no preprocessing (Score: 59.92%)
- ***BL2:*** Logistic regression classifier, *word2vec* embeddings, no preprocessing (Score: 77.21%)

B. Preprocessing

1) Setup and Process for Evaluation: We used two classifiers to test the influence of our preprocessing methods. First, we applied a linear classifier that builds on logistic regression (LR). Second, we evaluated our preprocessing method on a recurrent neural network (RNN) (LSTM cells, 1 layer, 150 hidden units, static).

The goal is to improve classification accuracy (fraction of correctly classified tweets over all tweets); this is therefore the metric used to evaluate our preprocessing methods. Each method was evaluated in repeated experiments. We applied each preprocessing method individually and compared the scores to baseline. Next, we ran all combinations of the methods that had improved accuracy.

2) Results for Logistic Regression: In Table I, we show the accuracy effect of each method compared to the baseline score *BL2*.

The methods 3 and W reduced accuracy, the other methods improved it, likely because they reduce the number of embeddings and create indicative word associations. The method C worked better than D, congruent with the findings in [2]. The impact of N is almost negligible, probably because it modifies tweets only marginally. Lastly, R yields a relatively big improvement for the small changes it makes.

Preprocessing method	**Score** (%)	**Effect**	
None (*BL2*)	77.21	0.00	
3 (Short Word Removal)	76.65	-0.56	−
M (Same-Letter Removal)	77.44	+0.23	+
D (Negation Disconnection)	77.47	+0.30	+
C (Negation Connection)	77.65	+0.49	+
R (Tag Removal)	77.80	+0.64	+
N (Negation Replacement)	77.20	+0.04	+
W (Word Type Filtering)	76.47	-0.69	−
S (Word Cancellation)	77.49	+0.33	+
MD	77.61	+0.44	+
MR	77.69	+0.53	+
DR	77.77	+0.60	+
DS	77.72	+0.56	+
CR	77.66	+0.49	+
RS	77.77	+0.61	+

Table I
EFFECTS OF PREPROCESSING METHODS IN THE LOGISTIC REGRESSION CLASSIFIER

Preprocessing method	**Score** (%)	**Effect**	
None	81.24	0.00	
3 (Short Word Removal)	80.45	-0.79	−
M (Same-Letter Removal)	81.89	+0.64	+
D (Negation Disconnection)	81.05	-0.20	−
C (Negation Connection)	81.28	+0.03	+
R (Tag Removal)	80.90	-0.34	−
N (Negation Replacement)	80.94	-0.30	−
W (Word Type Filtering)	80.98	-0.26	−
S (Word Cancellation)	81.72	+0.47	+
MC	81.30	+0.06	+
MS	81.96	+0.71	+
CS	81.96	+0.71	+
MCS	81.92	+0.68	+

Table II
EFFECTS OF PREPROCESSING METHODS IN THE RNN CLASSIFIER

In Table I, a subset of the results for two preprocessing methods are shown. For the full results, please refer to the appendix.

Note that the pairs involving R improved accuracy the most. However, it is unclear to us why these combinations lead to worse accuracy compared to R alone, especially since the combined preprocessing methods do not interfere with each other. Contrary to pairs involving R, the remaining combinations improved nearly by the sum of their single-scores. We assign this effect, again, to the reduced word embeddings and strengthening of common word associations.

3) Results for RNN: Table II contains information on how the classification accuracy of a Recurrent Neutral Network (RNN) is influenced by our preprocessing methods. One quickly notices that, unlike in the case of a logistic regression, preprocessing the tweets leads to a decrease in accuracy in more cases.

We notice that word cancellation is not necessarily a source of inaccuracy, since both improving as well as damaging methods (S and 3) perform some kind of word

Embedding	Classifier	Preprocessing	Score (%)
GloVe	LR	None	67.82
		M	68.42
	RNN	None	75.36
		M	75.81
word2vec	LR	None	77.21
		M	77.44
	RNN	None	81.24
		M	81.89

Table III
EFFECTS OF PERFORMANCE ON CLASSIFICATION ACCURACY

cancellation. Accuracy effects may thus depend on *which* words are removed.

Unexpectedly, the gains achievable when using method M alone are virtually neutralized when using it in combination with method C. Surprising on this note is that method C significantly helps method S.

4) Comparison of Classifiers: Our results show that the effect of a preprocessing method is heavily dependent on the used classifier. The following examples illustrate this finding:

- Negation replacement (N) greatly affects RNN whereas LR barely notices it.
- Negation connection (C) greatly helped LR but is negligible for RNN.
- Tag removal (R) has a positive effect on LR accuracy but a negative effect on RNN accuracy.

In conclusion, handling same-letter sequences is a safe improvement, given that both LR and RNN seem to benefit from it. They also agreed on negation connection, although care should be taken when combining it with other preprocessing method. Lastly, our method to remove simple, common words also helps either model.

C. Word-Vector Embeddings

We evaluated the effects of the different embedding methods from Section II-B on classification accuracy. In these experiments, the results in Table III were obtained: Clearly, *word2vec* performs considerably better than *GloVe*, independently of classifier and preprocessing used.

D. Classification

Selected results of classification are displayed in Table IV. All classification evaluation experiments were performed using a skipgram model of window size 5 and filtering words that occur less than 5 times with an embedding dimension of 100, a dropout of 0.75, and a vocabulary size of 89'000 on the test set.

It is enlightening to analyze which model more accurately predicts the sentiment given the results obtained on the test set in the ranking system (Kaggle). In our study, we observe that in general, precision is increased by inserting more layers and keeping approximately the same number of parameters in our classifiers. Notwithstanding, the marginal benefit from incrementing the number of layers clearly diminishes. Secondly, adding an attention mechanism sightly boots the accuracy. However this pattern is no longer observable when using four layers. Surprisingly, the local attention does not obtain a better score than the global one.

IV. Discussion

Judging from the results in the previous section, our step-wise optimization approach yielded counter-intuitive findings as well as notable improvements in accuracy.

We want to highlight two unexpected findings and give possible explanations for them:

First, we wrongly assumed that most words removed by preprocessing methods 3 and W are neutral regarding sentiment and thus not helpful in classification. Possibly, removing many words could have the effect that context windows become more diverse. If co-occurrence counts are smaller, vector embeddings become more imprecise, which would explain the drop in classification accuracy.

Second, the benefit of preprocessing methods is heavily dependent on the used classifier. Source of these differences could be the categorically different *modus operandi* of the two classifiers: Whereas LR condenses all vectors relating to tweet words into a single vector, RNN processes individual word vectors in the word order of the tweet. Structure may render even insignificant words more relevant. For example, prepositions are not very meaningful on their own but become more meaningful when incorporated in a sentence structure.

On the contrary, the benefit of single embedding methods is very unambiguous. Not only did *word2vec* embeddings lead to better results, but their production was also far faster. However, this observation can be partially explained by the fact that we used the C code to produce $word2vec$ embeddings and Python code to produce $GloVe$ embeddings.

V. Summary

We described our deep learning approach to Twitter sentiment analysis on phrase level. We gave a detailed description of our 3-step process consisting of preprocessing, word-vector embeddings production, and classification. In each step, we evaluated several different approaches and combined the best-performing solutions from every step. The resulting model demonstrates state-of-the-art performance and provides a novel approach by introducing an attention mechanism to our multi-layer LSTM model.

Classifier	**Units**	**Layers**	**Embeddings**	**Score** (%)
Logistic Regression (*BL1*)	-	-	GloVe	59.92
Logistic Regression (*BL2*)	-	-	word2vec	77.21
LSTM	252	1	word2vec	87.22
LSTM	252	2	word2vec	87.38
LSTM + Global Attention	200	2	word2vec	87.52
LSTM + Local Attention	200	2	word2vec	87.56
LSTM	190	3	word2vec	87.20
LSTM + Global Attention	190	3	word2vec	87.74
LSTM + Local Attention	190	3	word2vec	87.28
LSTM	150	4	word2vec	87.96
LSTM + Global Attention	150	4	word2vec	87.72
LSTM + Local Attention	150	4	word2vec	87.67

Table IV
ACCURACY RESULTS OF RNN CLASSIFIER

REFERENCES

[1] Twitter Inc. Twitter. [Online]. Available: http://www.twitter.com/

[2] Y. Bao, C. Quan, L. Wang, and F. Ren, "The Role of Preprocessing in Twitter Sentiment Analysis," in *International Conference on Intelligent Computing*. Springer, 2014, pp. 615–624.

[3] I. Hemalatha, G. S. Varma, and A. Govardhan, "Preprocessing the informal text for efficient sentiment analysis," *International Journal of Emerging Trends & Technology in Computer Science (IJETTCS)*, vol. 1, no. 2, pp. 58–61, 2012.

[4] 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

[5] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient Estimation of Word Representations in Vector Space," *CoRR*, vol. abs/1301.3781, 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[6] R. Socher, C. C. Lin, C. Manning, and A. Y. Ng, "Parsing Natural Scenes and Natural Language with Recursive Neural Networks," in *Proceedings of the 28th international conference on machine learning (ICML-11)*, 2011, pp. 129–136.

[7] L. Dong, F. Wei, C. Tan, D. Tang, M. Zhou, and K. Xu, "Adaptive Recursive Neural Network for Target-Dependent Twitter Sentiment Classification." in *ACL (2)*, 2014, pp. 49–54.

[8] Q. Qian, B. Tian, M. Huang, Y. Liu, X. Zhu, and X. Zhu, "Learning Tag Embeddings and Tag-Specific Composition Functions in Recursive Neural Network," in *ACL (1)*, 2015, pp. 1365–1374.

[9] R. Socher, D. Chen, C. D. Manning, and A. Ng, "Reasoning with Neural Tensor Networks for Knowledge Base Completion," in *Advances in neural information processing systems*, 2013, pp. 926–934.

[10] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, "Recurrent Neural Network Based Language Model," in *Interspeech*, vol. 2, 2010, p. 3.

[11] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[12] K. C. Dzmitry Bahdanau and Y. Bengio, "Neural Machine Translation by Jointly Learning to Align and Translate," *International Conference on Learning Representations*, 2015. [Online]. Available: https://arxiv.org/pdf/1409.0473.pdf

[13] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-Based Neural Machine Translation," *arXiv preprint arXiv:1508.04025*, 2015. [Online]. Available: http://aclweb.org/anthology/D15-1166

[14] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.