

Computational Intelligence Lab: Tweet Sentiment Classification

Lukáš Jendele*
ETH Zürich
jendelel@ethz.ch

Larissa Laich*
ETH Zürich
llaich@ethz.ch

Ondrej Skopek*
ETH Zürich
oskopek@ethz.ch

Michael Wiegner*
ETH Zürich
wiegnerm@ethz.ch

Abstract—Sentiment analysis is a key stepping stone on the path to natural language understanding. In this paper, we show that deep neural networks can learn to classify sentiment even on very short and noisy texts without any preprocessing, by evaluating our models on the Twitter dataset and obtaining competitive results (88.5% test set accuracy). We investigated the performance of different models Recurrent and Convolutional Neural Network based models on the binary sentiment classification problem and compared all our approaches to simpler baselines.

I. INTRODUCTION

With the rise of Internet technologies, micro-blogging sites like Twitter have become increasingly popular to share and discuss information and express personal opinions. Used by millions of people on a daily basis, Twitter has become a source of data and offers huge opportunities for analysis. One of many applications is for example market analysis: how users perceive certain companies' products. Due to the large amount of data, reliable automated sentiment analysis is essential to be able to process and extract information from it efficiently. Recent developments in machine learning, like deep neural networks have shown promising results when performing sentiment analysis. However, there is still room for improvement, which is what we want to investigate in this paper. In our task, we classify tweets based on positive or negative sentiment.

Tweets are short messages on Twitter which were originally limited to a length of 140 characters. Due to this length limitation and the fact that it is a fast-paced micro-blogging platform, most people use abbreviations, emoticons, hash-tags, or "@username" to refer to other users. The dataset is therefore relatively structured and allows us to determine sentiment by checking emoticons occurring in tweets. Tweets with a "😊" smiley are considered to be positive tweets while negative tweets contain a "😞" smiley. This allows us to label the data automatically as the smileies are removed from the training data and used as a label. Determining tweet sentiment simply based on an emoticon contained in the tweet introduces some noise to the labels, as there sometimes is not a direct correlation between sentiment and smiley (e.g. irony, sarcasm, etc.). On the other hand, the use of colloquial language makes this task different from analyzing

text from books or articles, and reflects the rough real world conditions of language use.

For this project, we designed and implemented multiple machine learning models for sentiment analysis. The results are evaluated based on classification accuracy. Our contributions are the following:

- 1) Improved accuracy from the best baseline models by approximately 7%.
- 2) By using ensembles, improved performance by another 0.5%.
- 3) Achieved 3rd best result on both public and private Kaggle test datasets.

II. RELATED WORK

Sentiment analysis is one the most basic text classification tasks, and therefore, has been a subject of interest to many researchers [1], [2], [3], [4], [5]. While part of the research focused on data preprocessing [4], others were more concerned with the classification model itself [1].

Before deep learning models, researchers extracted features from the text and ran a binary classifier on these features {**TODO: Find paper that does this.**}. Another simple approach was to sum or average learned word embeddings and classify them with standard machine learning algorithms, as recommended in the CIL course tutorial.

With the recent success of deep neural networks in natural language processing problems, researchers tackled sentiment analysis by applying various convolutional [1] and recurrent neural networks [6] to the task.

In our approach, we implemented both classical, as well as modern neural network approaches, apply them to sentiment analysis and compare them with each other.

III. MODELS AND METHODS

We use several different approaches to classify sentiment. In this section, we briefly describe each model and explain our motivation for its usage.

A. Baselines

1) *Glove + Logistic regression*: This baseline is taken from the CIL lecture. We use GloVe word embeddings [7] and average them for each tweet. On the resulting vector, we apply logistic regression. For this baseline, we either compute the Glove embeddings based on our training data by

* All authors contributed equally.

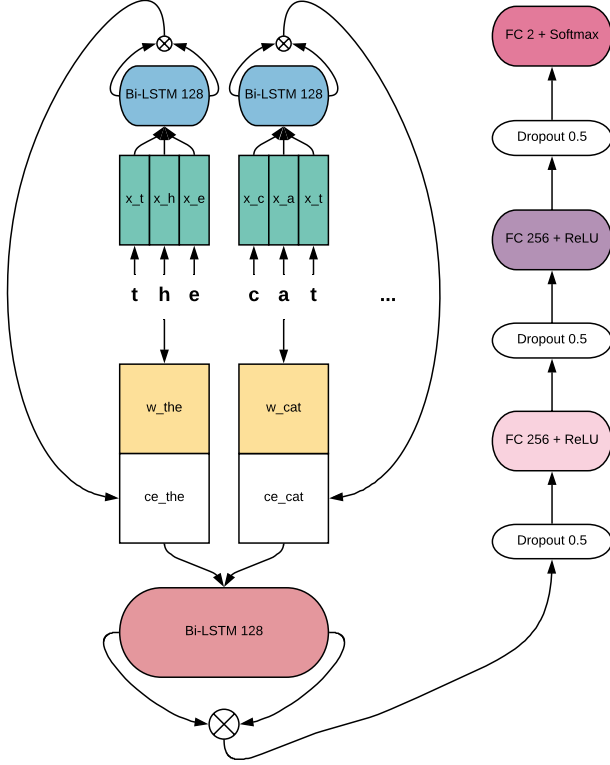


Figure 1. Sketch of our recurrent neural network architecture LSTM128. Equal colors indicate shared weights. x_* represent individual character embeddings, w_* represent word embeddings, and ce_* represent character-based word embeddings.

using scripts from the CIL tutorial, or use precomputed ones from Stanford University¹. Both variants of the embeddings have 200 dimensions.

2) *Glove + Random Forest*: Similarly to the previous baseline, we average the GloVe word embeddings for each tweet. However, this time we use a Random Forest (RF) classifier [8] with 128 trees. We report scores for evaluation using both the self trained and Stanford pre-trained Glove embeddings. The same dimensionality as above was used.

3) *FastText*: FastText [9] is a command-line tool developed by Facebook AI Research. As it is simple to set up and runs fast, we decided to run it on our dataset and consider it as one of the baselines. In one step, it learns word embeddings in an unsupervised way, and trains a simple classifier on them.

B. Common preprocessing

For our RNN and CNN models, we apply a common loading and preprocessing step. As the input data is already lowercased and spaces are normalized, we only apply a vocabulary downsize and padding step for computational reasons. We only add the 20 000 most common words (in

the training data) to our vocabulary. All other words are mapped to an UNK token. Moreover, we clip tweets at 40 words (only very few tweets are longer than that).

We also experimented with more heavy preprocessing similar to [4], but did not manage to achieve significant improvements in performance compared to the un-preprocessed variant of our models.

C. RNN models

Recurrent neural networks are one of the most used network architectures for natural language processing tasks such as machine translation, language modeling, and text classification.

Our model architecture is depicted in Figure 1. First, we run a bi-directional RNN on each of the tweet words' embedded characters. We concatenate the final hidden states with randomly initialized whole-word embeddings, and use the concatenated vectors as the final word embeddings for the next layer. Another bi-directional RNN network is run on the final word embeddings and its hidden states are fed into two 256-unit fully connected layers with ReLU activation functions. Finally, we append a fully connected layer with Softmax that classifies into positive and negative labels, and train the network using a cross-entropy loss function.

We use only two RNN cells with dimensionality of the hidden state(s) 128, one for the character RNN and one for the word RNN. A single cell is used for both directions. Our character and word embeddings have a dimensionality 200.

We also briefly experimented with both Luong [10] and Bahdanau [11] attention, but only observed higher training times and equivalent performance to models without attention.

The performance of the model depends largely on the recurrent cell. Thus, we ran the described architecture with minor changes (each described below) and report the performance.

1) *LSTM128*: Long-Short-Term-Memory [12] cells are the most frequently used recurrent units used to process text. They are well-known for their ability to capture long-range contextual features in text. In our implementation, the cell has a dimensionality 128, and we concatenate both the cell and hidden states of the final cell before passing them to the fully-connected layers.

2) *GRU256*: Gated Recurrent Unit [13] has proven to be the best compromise of learning capabilities and computational complexity. It contains less gating (only one hidden state) than the more popular and older LSTM cell. Therefore, we ran an experiment in which we replaced the LSTM cell with a GRU cell of dimensionality 256.

3) *LSTM128 without character embeddings*: In this experiment, we removed the first RNN network. That way, we completely ignore word morphology, but acquire a significantly simpler model.

¹<https://nlp.stanford.edu/projects/glove/>

4) *LSTM128 without word embeddings*: Conversely, we only use the character based part computed by the first RNN here. This enforces a more hierarchical and end-to-end learned model.

5) *Stacked LSTM*: To increase the model capacity (its power to learn), we stack multiple LSTM cells on top of each other, which is a common practice in machine translation models [14]. For our purposes, we use two LSTMs, with dropout 0.5 [15], [16] on all individual LSTM cell inputs and outputs.

D. Other models

1) *TextCNN*: Recurrent neural networks are well known for their capabilities to capture long context. However, tweets are very short and therefore a limited fixed receptive field is most likely capable of learning the text sentiment. Thus, we replaced the word RNN with convolutions similar to [1]. We used kernel sizes 3, 4, 5, and 7, each with 512 channels.

2) *Transformer Tiny*: Transformer [3] is a state of the art method in machine translation. It uses gating similar to RNN cells, while processing tokens in parallel similar to CNNs. We utilized the Tensor2Tensor [17] implementation, and trained the “tiny” version (2 hidden layers) of the model’s encoder for classification.

3) *Ensemble Majority Vote*: Motivated by most data science competitions, an ensemble of several strong models usually yields to better performance. Thus, to lower the prediction variance, we combined the test predictions of several different approaches using majority voting: LSTM128, Stanford GloVe+RF, Transformer, Stacked LSTM, and CNN. We decided to select five models as an odd number always leads to a majority vote for one of the two labels. Additionally, a single model still has some weight when we choose a relatively small number of models.

E. Implementation

All neural network models are implemented in TensorFlow [18], and trained with the Adam optimizer [19] with a learning rate of $3 \cdot 10^{-4}$. All variables were initialized with Glorot initialization [20]. The GloVe embedding implementation was adapted from the coding exercise solution to CIL tutorial 6. When training GloVe embeddings, we train for 20 epochs. For the Random Forests and Logistic Regression, we used the NumPy [21] and scikit-learn [22] Python libraries. For FastText and Transformer, we used the original implementations available online². The ensemble majority voting was done in a simple spreadsheet.

All code is available in the repository³ and is also attached to our paper submission.

²<https://fasttext.cc/>, <https://github.com/tensorflow/tensor2tensor>

³<https://github.com/oskopek/cil>

Model	Evaluation Acc. %	Test (public) Acc. %	Test (private) Acc. %
GloVe + LR	61.69	61.56	62.10
Stanford GloVe + LR	78.08	77.06	76.94
GloVe + RF	71.00	67.84	69.04
Stanford GloVe + RF	79.97	77.00	77.94
FastText [9]	81.31 ± 2.21	80.88 ± 2.03	80.50 ± 1.57
knilt		89.08	88.48
Satisfaction		88.78	88.62
LSTM128	88.49	87.90	87.82
GRU256	88.45	87.66	87.52
LSTM128 w/o WE	?	?	?
LSTM128 w/o CE	88.18	87.82	87.90
StackLSTM	88.58	88.06	87.68
TextCNN	87.71	87.70	87.10
Transformer Tiny	86.00	85.34	83.84
Ensemble Majority Vote		88.54	87.78

Table I

EVALUATION AND TEST SET RESULTS. VALUES LEFT BLANK ARE UNKNOWN. ACC. MEANS ACCURACY, LR IS LOGISTIC REGRESSION, WE/CE ARE WORD/CHARACTER EMBEDDINGS RESPECTIVELY, AND RF DENOTES A RANDOM FOREST. THE TOP SECTION DESCRIBES BASELINES THAT WE RAN ON OUR DATA SPLIT. MIDDLE SECTION CONTAINS THE TWO TOP TEAMS ON KAGGLE FOR THIS PROJECT. BOTTOM SECTION SHOWS OUR RESULTS. VALUES THAT HAVE CONFIDENCE INTERVALS WERE UNSTABLE ENOUGH TO BE HAVE TO BE RUN 5 TIMES.

IV. EXPERIMENTS

To achieve reproducible results, we trained all experiments with the same fixed random seed. Moreover, we split the data into a training and evaluation set which we used for all the experiments. The evaluation set contained 100 000 tweets, and the remaining 2.4 million tweets were used for training.

All neural network experiments were trained on the Leonhard ETHZ cluster on Nvidia GeForce GTX 1080 Ti. All of these experiments ran for a maximum of 24 hours, 15 epochs, with batch size 32. We evaluated on the full evaluation set every 7500 steps (independent on the batch size), and as the final predictions, used the ones that were created at the point where the model had the highest evaluation accuracy. Most models obtained best results around epochs 9–12, which corresponds to about 21 hours in training time.

The Stacked LSTM ran with batch size 256 for about 12 hours. The Transformer {**TODO: Lukas finish this**}. All baselines ran for a very short time on commodity CPU hardware. Due to the instability of FastText with respect to the random seed, we perform 5 runs, and report the standard deviations.

V. RESULTS

The results of all our experiments are summarized in Table I. The reported evaluation accuracy is the best one obtained during training (and hence the version of the model with which we submitted the predictions to Kaggle). The public and private test scores correspond to the scores those

Model	Parameters
LSTM128	?
GRU256	?
LSTM128 w/o WE	?
LSTM128 w/o CE	?
StackLSTM	?
TextCNN	?
Transformer Tiny	{TODO: Lukas}
Ensemble Majority Vote	?SUM?

Table II
NUMBER OF PARAMETERS OF OUR MODELS.

% Same	LSTM128	SF GloVe+RF	Transformer	Stacked LSTM	CNN	Ensemble
LSTM128	100.00%	81.12%	89.74%	94.28%	94.13%	96.47%
SF GloVe+RF	81.12%	100.00%	81.36%	81.10%	82.01%	83.55%
Transformer	89.74%	81.36%	100.00%	89.30%	89.75%	92.25%
Stacked LSTM	94.28%	81.10%	89.30%	100.00%	93.33%	95.95%
CNN	94.13%	82.01%	89.75%	93.33%	100.00%	96.40%
Ensemble	96.47%	83.55%	92.25%	95.95%	96.40%	100.00%

Table III
PAIRWISE LABEL AGREEMENT BETWEEN THE VARIOUS MODELS USED IN THE ENSEMBLE ON THE TEST SET. NOTE THAT THE TABLE IS SYMMETRIC.

submissions obtained on the Kaggle data splits. The values left blank are unknown.

In Table II, we present the number of parameters for our models (including the learned embeddings).

VI. DISCUSSION

We can observe that all our models perform consistently better than all baselines, and most also perform better than the best submissions of all teams below us on the Kaggle leaderboard.

An interesting observation is that some models generalize to the test set better than others. For example, the accuracy of TextCNN only drops marginally on the test sets compared to the evaluation set, as opposed to all our RNN models. Generally, we observe that the simpler the model, the better it generalizes (among neural models, e.g. LSTM128 w/o WE, LSTM128 w/o CE).

As can be seen in Table II, the number of parameters in a model has limited effect on its performance. We can see that above a model complexity, adding more parameters only helps marginally. {TODO: (oskopek): Describe/fix/make more concrete the description of model perf based on number of params.}

An interesting comparison of how surprisingly diverse the models we trained are can be seen in Table III. Interestingly, our well performing models (LSTM128, CNN512, and StackLSTM) only agree with each other in about 94% of the cases (pairwise). We can also see the effects of lower performance of the Transformer and GloVe+RF models on the pairwise agreement with other models. {TODO: Expand this if we add FastText and GRU256, might be interesting to compare them to Glove and LSTM, respectively.}

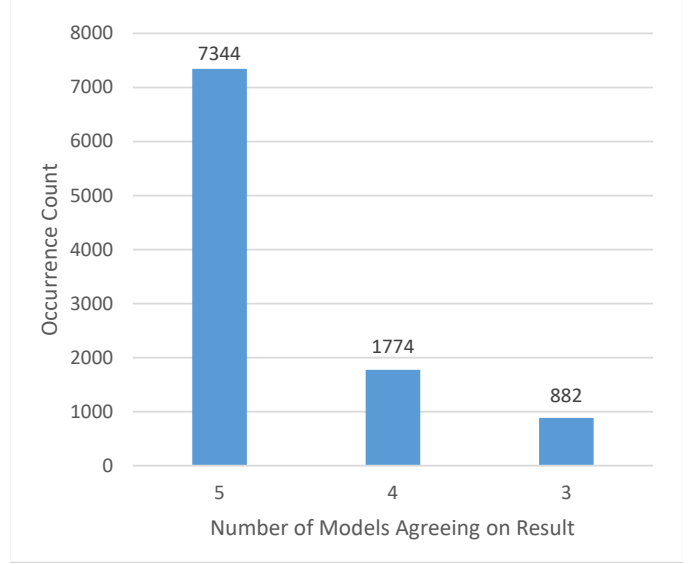


Figure 2. Bar plot of the number of models that ended up voting for the final majority result.

Our ensemble model leverages this diversity. The majority voting results are summarized in Figure 2. We can see that in a total of about 26.5% of the cases, at least one model was unsure about the chosen prediction.

VII. SUMMARY

To summarize, we have presented several approaches that perform competitively on the task of Twitter sentiment classification. They consistently outperform the baselines, behave in a stable fashion during training, are simple to implement, and need almost no preprocessing (on top of the already provided and processed data).

We have also shown that the models are diverse, and combining them in an ensemble reduces prediction variance and thus improves generalization.

ACKNOWLEDGEMENTS

We would like to thank the Data Analytics Lab and ETH Zürich for computational resources for this project on the Leonhard cluster, and answering all our questions on Piazza or otherwise.

REFERENCES

- [1] Y. Kim, “Convolutional neural networks for sentence classification,” *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: <http://arxiv.org/abs/1408.5882>
- [2] I. Mozetič, M. Grčar, and J. Smailović, “Multilingual twitter sentiment classification: The role of human annotators,” *PloS one*, vol. 11, no. 5, p. e0155036, 2016.
- [3] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, “Attention is all you need,” *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: <http://arxiv.org/abs/1706.03762>

- [4] D. Effrosynidis, S. Symeonidis, and A. Arampatzis, "A comparison of pre-processing techniques for twitter sentiment analysis," 09 2017.
- [5] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in Neural Information Processing Systems* 28, C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, Eds. Curran Associates, Inc., 2015, pp. 649–657. [Online]. Available: <http://papers.nips.cc/paper/5782-character-level-convolutional-networks-for-text-classification.pdf>
- [6] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *arXiv preprint arXiv:1605.05101*, 2016.
- [7] 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>
- [8] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: <https://doi.org/10.1023/A:1010933404324>
- [9] A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, "Bag of tricks for efficient text classification," in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*. Association for Computational Linguistics, April 2017, pp. 427–431.
- [10] M.-T. Luong, H. Pham, and C. D. Manning, "Effective Approaches to Attention-based Neural Machine Translation," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 2015. [Online]. Available: <https://arxiv.org/pdf/1508.04025.pdf>
- [11] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-End Attention-Based Large Vocabulary Speech Recognition," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2016. [Online]. Available: <https://arxiv.org/pdf/1508.04395.pdf>
- [12] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.
- [13] K. Cho, B. van Merriënboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014. [Online]. Available: <http://arxiv.org/abs/1406.1078>
- [14] M. Johnson, M. Schuster, Q. V. Le, M. Krikun, Y. Wu, Z. Chen, N. Thorat, F. B. Viégas, M. Wattenberg, G. Corrado, M. Hughes, and J. Dean, "Google's multilingual neural machine translation system: Enabling zero-shot translation," *CoRR*, vol. abs/1611.04558, 2016. [Online]. Available: <http://arxiv.org/abs/1611.04558>
- [15] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," 2012. [Online]. Available: <http://arxiv.org/abs/1207.0580>
- [16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: <https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf>
- [17] A. Vaswani, S. Bengio, E. Brevdo, F. Chollet, A. N. Gomez, S. Gouws, L. Jones, L. Kaiser, N. Kalchbrenner, N. Parmar, R. Sepassi, N. Shazeer, and J. Uszkoreit, "Tensor2tensor for neural machine translation," *CoRR*, vol. abs/1803.07416, 2018. [Online]. Available: <http://arxiv.org/abs/1803.07416>
- [18] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng, and G. Brain, "TensorFlow: A System for Large-Scale Machine Learning," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 2016, pp. 265–284. [Online]. Available: <https://www.usenix.org/conference/osdi16/technical-sessions/presentation/abadi>
- [19] D. P. Kingma and J. L. Ba, "Adam: A Method for Stochastic Optimization," *International Conference on Learning Representations (ICRL)*, dec 2015. [Online]. Available: <http://arxiv.org/abs/1412.6980>
- [20] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," in *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, 2010, pp. 249–256.
- [21] T. E. Oliphant, *A guide to NumPy*, 2006, vol. 1.
- [22] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.



Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

COMPUTATIONAL INTELLIGENCE LAB: TWEET SENTIMENT CLASSIFICATION

Authored by (in block letters):

For papers written by groups the names of all authors are required.

Name(s):

Jendele
Laich
Skopek
Wiegner

First name(s):

Lukas
Larissa
Ondrej
Michael

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zürich, 26th June

Signature(s)

Lukas
Larissa
Skopek
Michael Wiegner

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.