Computational Intelligence Lab: Tweet Sentiment Classification
Group: Laich et al. (2018)

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 with almost no preprocessing, by evaluating our models on the Twitter dataset and obtaining competitive results (88.5% test set accuracy). We investigated the performance of different 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 became 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, hashtags, 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 smilies 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, ...). 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 [6], 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 [7] 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 [8] and average them for each tweet. On the resulting vector, we

* All authors contributed equally.

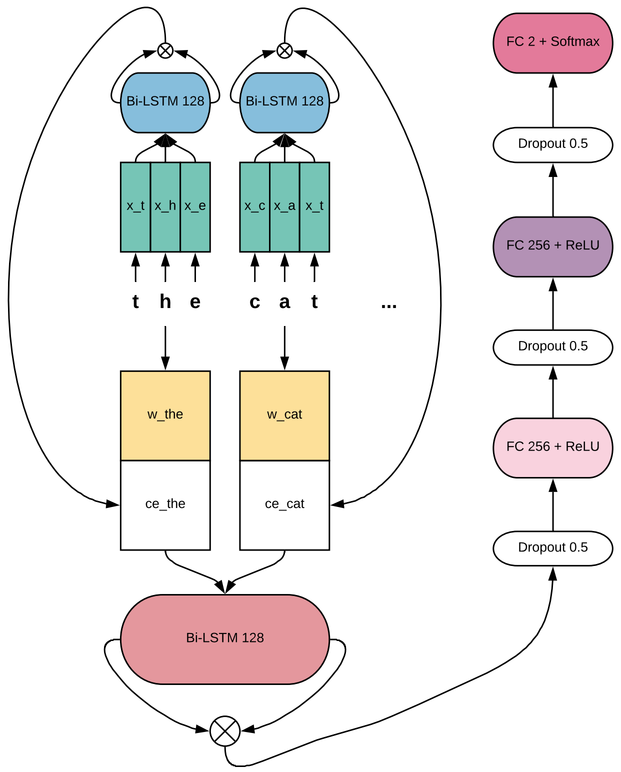


Figure 1. Sketch of our recurrent neural network architecture LSTM128. Equal colors indicate shared weights. $x\_$ represent individual character embeddings, $w\_$ represent word embeddings, and $ce\_$ represent character-based word embeddings. The $\otimes$ symbol indicates concatenation.

apply logistic regression. For this baseline, we either compute the GloVe embeddings based on our training data by using scripts from the CIL tutorial, or use precomputed ones from Stanford University[1]. 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 [9] 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 [10] 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 and dropout [11], [12] in between them (with keep probability 0.5). 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 200 dimensions.

We also briefly experimented with both Luong [13] and Bahdanau [14] 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 [15] 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's hidden state has 128 dimensions, and we concatenate both final cell state and final hidden state of the cell. Then we concatenate the forward and backward states of the bi-directional RNN and pass the result to the fully-connected layers.

2) GRU256: Gated Recurrent Unit [16] 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.

[1]https://nlp.stanford.edu/projects/glove/

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.

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's capacity (its power to learn), we stack multiple LSTM cells on top of each other, which is a common practice in machine translation models [17]. For our purposes, we use two LSTMs, with dropout with a keep probability of 0.5 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 (model implementation is called CNN512).

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 [18] 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 [19], and trained with the Adam optimizer [20] with a learning rate of $3 \cdot 10^{-4}$. All variables were initialized with Glorot initialization [21]. The GloVe embedding implementation was adapted from the coding exercise solution of CIL tutorial 6. When training GloVe embeddings, we train for 20 epochs. For the Random Forests and Logistic Regression, we used the NumPy [22] and scikit-learn [23] Python libraries. For FastText and Transformer, we used the original implementations available online[2]. The ensemble majority voting was done in a simple spreadsheet.

All code is available in the repository[3] and is also attached to our paper submission.

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 a single Nvidia GeForce GTX 1080 Ti GPU. All of our experiments ran for a maximum of 24 hours, 15 epochs, with batch size 32. Every 7500 steps, we evaluated on the full evaluation set (independently of the batch size). For the final predictions, we used the ones that were created at the point when the model had the highest evaluation accuracy.

Most models obtained best results around epochs 9–12, which corresponds to about 21 hours of training time. The Stacked LSTM ran with a batch size of 256 for about 12 hours.

The Transformer was trained for 40 000 steps in mini-batches of 8 000 words. The model trains for about an hour, which is significantly faster than all of the RNN based models.

FastText ran for a very short time on commodity CPU hardware. Due to its instability, with respect to the random seed, we perform 5 runs, and report the standard deviations.

Our GloVe implementation consumes a lot of memory (peak at ≈ 40 GB), and runs for about 12 hours on 4 CPUs (all four experiments in total).

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 this is the version of the model we used to submit predictions to Kaggle). The public and private test scores correspond to the scores those 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

[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 [10]	**81.31 ± 2.21**	**80.88 ± 2.03**	**80.50 ± 1.57**
knilpt		**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	87.88	86.98	86.56
LSTM128 w/o CE	88.18	87.82	**87.90**
Stacked LSTM	**88.58**	88.06	87.68
TextCNN	87.71	87.70	87.10
Transformer Tiny	86.00	85.34	83.84
Ensemble Maj. 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.

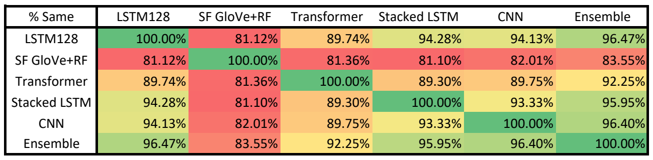
Model	Parameters
LSTM128	4.8 M
GRU256	5.3 M
LSTM128 w/o WE	0.7 M
LSTM128 w/o CE	4.4 M
Stacked LSTM	21.1 M
TextCNN	11.7 M
Transformer Tiny	1.5 M
Ensemble Majority Vote	≈ 51.1 M

Table II
NUMBER OF PARAMETERS OF OUR MODELS.

it generalizes (among neural models, e.g. LSTM128 w/o WE, LSTM128 w/o CE).

As can be seen in Table II, our models had very different numbers of parameters but perform very similarly in terms of accuracy. Adding more parameters increases the performance, but only marginally. LSTM models clearly show the best trade-off between model complexity and performance. Despite weaker performance, the Transformer's scores are impressive considering the duration of training ($\approx 20\times$ faster) and lower parameter count. Given enough parameter tweaking, the model has considerable potential.

Table III shows an interesting comparison how much our models agree on the predictions and reveals how surprisingly diverse our trained models are. Interestingly, our well performing models (LSTM128, TextCNN, and Stacked LSTM) only agree with each other (pairwise) in about 94% of the cases. We can also see the effects of lower performance



% 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.

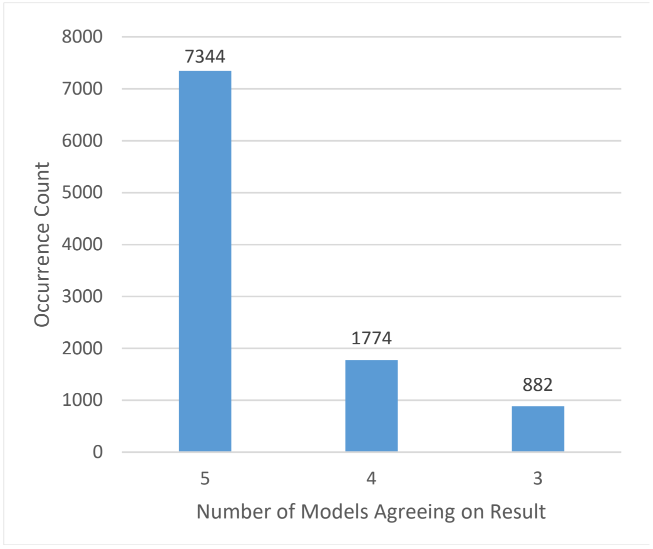


Figure 2. Bar plot of the number of models that ended up voting for the final majority result.

of the Transformer and GloVe+RF models on the pairwise agreement with other models.

Our ensemble model leverages this diversity. The majority voting results are summarized in Figure 2. In about 26.5% of the cases, at least one model chose to predict a different class than the rest.

VII. SUMMARY

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 comparatively 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] A. Moschitti and R. Basili, "Complex linguistic features for text classification: A comprehensive study," in *Advances in Information Retrieval*, S. McDonald and J. Tait, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, pp. 181–196.

[7] P. Liu, X. Qiu, and X. Huang, "Recurrent neural network for text classification with multi-task learning," *arXiv preprint arXiv:1605.05101*, 2016.

[8] 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

[9] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, Oct. 2001. [Online]. Available: https://doi.org/10.1023/A:1010933404324

[10] 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.

[11] 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

[12] 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

[13] 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

[14] 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

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

[16] K. Cho, B. van Merrienboer, 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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.

[22] T. E. Oliphant, *A guide to NumPy*, 2006, vol. 1.

[23] 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.