EPFL Machine Learning Text Classification 2019
CS-433 Machine Learning – Project 2

Louis Amaudruz, Andrej Janchevski, Timoté Vaucher
IC, EPFL, Lausanne, Switzerland

***Abstract*—In this report, we take a look at the binary classification of tweets. We need to predict whether the original tweet contained a positive or a negative emoji. To this end, we first use state-of-the-art data preprocessing, identify task-specific important features and devise four models: a classic ML baseline, a GRU model using GloVe embeddings and two transfer-learning models based on ULMfit and BERT respectively. The best classifier we found is the BERT model which yields a 0.904 accuracy and F-1 score on the test set in the competition.**

I. Introduction

Sentiment analysis refers to the use of natural language processing, text analysis and computational linguistics to systematically identify, extract, quantify, and study affective states and subjective information. The simplest type of problems of this nature is the task of identifying the "polarization" of a text.

For this project, the goal was to utilize text classification techniques to categorize statements originating from the social network Twitter, as either positive or negative. These classes had been inferred from the tweets by the presence of the positive emoji ":)" or the negative emoji ":(". As such, this task can not be completely considered as a classical sentiment analysis problem, however classical techniques from this area of research are sufficiently applicable. The full data provided contained 2.5 million tweets from anonymous users, with 1.25 million tweets for each class.

The team decided to employ four different methods to approach the challenge. First, we utilized more classical feature-engineering machine learning to construct a baseline model, in order to analyze whether there exist characteristics of the tweets inherently indicative of their class. Then, we compared state-of-the-art deep-learning models and architectures used for text classification in order to improve on the first method and see the trade-off between interpretability of the results and model performance. In the following sections, we first present the preprocessing methods used for each model in section II, then the four devised procedures in section III and finally the results in section IV.

II. Data processing

We observed that the preprocessing steps applied have an effect on our models' performance, however we also note that the structure of the tweet preprocessing pipeline required slight modifications to accommodate the differences of the four approaches.

For the baseline method, it was required to perform more extensive transformation of the tweets to facilitate the feature extraction. First, the tweet text is parsed to replace common character patterns and Twitter specifics with special tokens: cropped tweet endings, hashtags, timestamps, emojis and numbers. For the hashtags, it was decided not to preserve the hashtag text. Using predefined lists commonly-used emojis were identified as positive or negative. After splitting on whitespace, stopwords (excluding "not") were removed from the obtained tokens, on which contraction of repeated characters and lemmatization was performed as final steps.

For the model using GloVe embeddings [1], we extend on the previous preprocessing by matching it with the one of the pretrained embeddings as it uses special tokens. We don't perform Stopword removal or Lemmatization here as there is enough training data. We use *Spacy* for the tokenization with the *en_core_web_sm* model [2].

The tweet preprocessing pipeline for the transfer learning approaches uses the same preprocessing used the model using GloVe embeddings with on top of that, processing of word repetition and adding special tokens at the start and the end of the tweets. This is especially useful for the language model.

III. Methodology

A. Classic ML baseline

1) Data: In order to reduce computational complexity, we only considered the smaller provided sample of 200,000 tweets for building the baseline model. In addition, we observed that a considerable number of tweets had been duplicated in the dataset, so to further reduce the data size and to avoid training a model on artificially skewed feature values it was decided only to preserve unique tweets in the dataset.

2) Feature Extraction: Five different types of features were extracted from each tweet text, using classical text analysis methods to extract sufficiently powerful descriptors.

Language style: In many past research studies it has been shown that simple word statistics can provide very good insight for the text classification problem, because they can efficiently model the writing style of the author of the text. As such, from the preprocessed tweet tokens we calculate:

the fraction of true English word tokens present, the mean word length, the fraction of dictionary words present and the fraction of unique tokens present.

TF-IDF vectors: After building a vocabulary of the 1000 most frequent tokens in the tweets, we calculate the TF-IDF-normalized Bag-of-Words matrix for our data, whose rows are used as the feature vectors for the tweets.

Morphology: We model grammar usage in the tweet text by calculating Penn Treebank Part-of-Speech (POS) tags for the word tokens, then extracting simple morphological statistics: the fraction of nouns, verbs, adjectives and adverbs present.

Vader sentiment: The nature of the task at hand requires that the emotional polarity of the tweet text is analyzed. We utilize the Vader sentiment intensity analyzer, a rule-based model which was built on top of lexical features extracted from social media text, which are combined with respect to conventional rules to model sentiment intensity (e.g., analysis of capitalization, punctuation, slang) [3]. As such, the raw unprocessed tweet text is fed to the Vader model, which generates three polarity scores: positive, neutral and negative.

Empath categories: Empath is a tool that analyzes text across around 200 built-in, pre-validated categories generated from common topics and emotions [4]. Each category is a set of tokens the model has validated to be very closely related. We pass the raw tweet text to the Empath analyzer in order to retrieve the normalized category scores as features.

3) Feature Selection: After the feature extraction process we obtain around 1200 feature values for each tweet. In order to reduce the computational complexity incurred from the high dimensionality of the generated dataset, we perform a feature cleaning process. First, the feature values were scaled to the interval $[0, 1]$ using min-max normalization. Then, a Random Forest classifier is fitted to the now normalized training data and the computed information gain values from the model are used to represent feature importances.

Figure 1 displays the cumulative distribution function of the distribution of the feature importances. We observe that if we only preserve 40 of the best features, as was done in the implementation, we already cover about 45% of the total importance.

4) Model validation and training: As the baseline model, five different standard classifiers were considered: Naive Bayes, Logistic Regression, linear SVMs, a three-layer Neural Network and Random Forest. Hyperparameters were fitted using grid search and 5-fold cross-validation with stratified random sampling. The classifiers were scored on 4 evaluation metrics: accuracy, precision, recall and F-1. The best model and hyperparameters were selected based on the mean F-1 validation score.

5) Implementation: For assistance in performing the natural language processing tasks in the preprocessing and feature extraction pipelines the NLTK package [5] was utilized, while all of the necessary standard machine learning procedures for the baseline model were implemented using the ScikitLearn library [6].

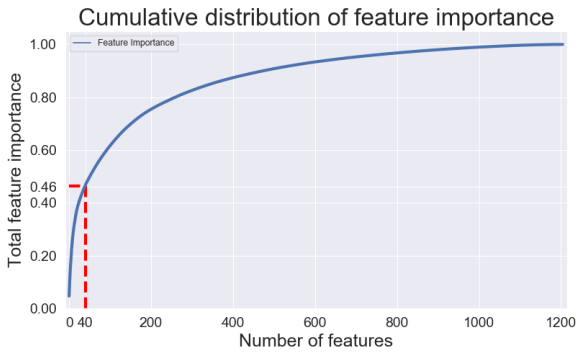

Figure 1. Cumulative distribution of feature importances computed by the feature selection model

B. Deep Learning: GloVe embeddings and GRUs

Diving in the world of Deep Learning, the idea of the second model is to transform a tweet into a sequence of vectors and then feed them sequentially to a RNN before classifying the output of the RNN. The model is thus composed of 3 distinctive parts: Embeddings, GRU and classifier as is shown in Figure 2 and written using PyTorch [7].

1) Embeddings: We use Stanford NLP twitter GloVe of dimension 200. The preprocessed tweets are first transformed into a list of compatible word indexes and then passed through the Embeddings layer. The weights of the Embeddings are frozen.

2) GRU: We use Bidirectional Gated Recurrent Units [8] with a hidden state of dimension 256. There's a 0.25 Dropout [9] between the connection of the hidden states. The final model has 3 layers of GRU and the initial hidden state is learnable. We will show that this improves the quality of the model.

3) Classifier: The classifier is a 2-hidden-layer feed forward Network of the shape $(256 \to 512 \to 256 \to 2)$ that takes the sum of the last hidden state of each extremity of the Bidirectional GRUs as input. The activation function is ReLU and there's a 0.5 Dropout between the layers. A softmax function is applied at the last layer.

4) Validation and training: All seeds are set to 1 and a 90%/10% train validation split is performed. We train the model for 8 epochs using Adam [10] with $\eta = 1e{-}3$, saving it and computing the validation accuracy at the end of each epoch. We choose the model of the epoch with the highest validation accuracy.

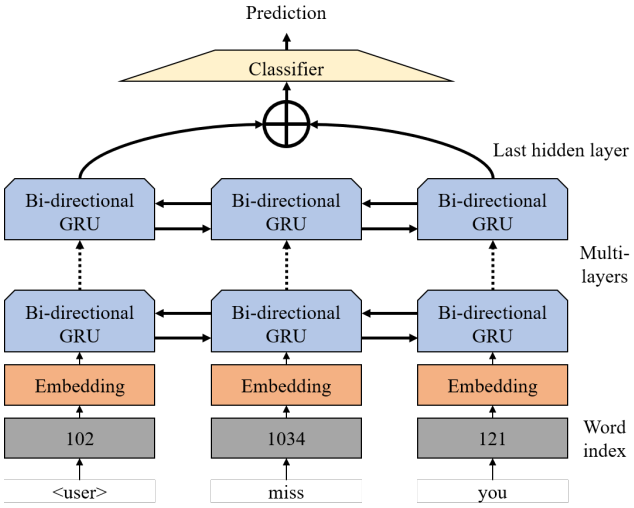

Figure 2. Schema of the Deep Learning model

C. Universal Language Model Fine-tuning for Text Classification (ULMFit)

Traditional RNN models used for NLP tasks, as used in our second method, present some drawbacks in that having to train them from scratch takes a consequent amount of data and time to converge. To address this issue we use inductive transfer learning for NLP as used in the ULMFit paper [11]. This approach consists in using a language model [12] that has already been trained on a large dataset of English literature, which we then fine-tune on our tweet dataset. After fine-tuning, the encoder (embeddings+RNN) of the language model is then used in conjunction with a classifier in order to classify the encoded tweets. Having an encoder that already knows how to encode meaningful information from the task specific language has not only been shown to greatly improve convergence rate and data required, but also to improve overall accuracy/loss.

1) Models: The language model consists of an encoder composed of a unidirectional 3-stacked layers RNN with LSTM units [13], and a simple linear layer decoder which maps outputs of the encoder to word distribution probability using softmax. Our classifier uses the same encoder with a pooling linear classifier on top which, from the output of the last layer of the encoder, concatenates the last output, the average and the max of all the inputs and then classify it.

2) Training: Usual activation dropout techniques [14] used to regularize neural networks tend not to work well with RNN architectures, which is why we use RNN specific dropouts for the encoder:

Embeddings dropout: Dropout applied to entire rows of the embeddings matrix.

Variational dropout: Dropout applied sequentially over the sequence length, that is, the same dropout is applied to each element of a sequence. This dropout is applied to each input and output sequences of the LSTM layers.

Weight tying: Dropout applied directly to the weights of the LSTM units.

We also use temporal activation regularization and activation regularization [15] which are modified L_2 regularization techniques for RNNs. As for the decoders we use basic dropout and batchnorm [16] on activations. For better convergence in the training loop, we use the Adam optimizer [10] with some techniques used in [11] and [15]:

Discriminative fine-tuning: Applying different learning rates to different layers of the model as they do not capture the same type of information.

Gradual unfreezing: With the model completely frozen, gradually fine-tune the model with each time unfreezing one layer of the encoder starting from the last layer. This prevents catastrophic forgetting [17].

Gradient clipping: Prevents gradient from getting too big which enables training with higher learning rate.

Cyclical learning rate and momentum: Modify the learning rate and Adam momentum during the training [15]

For the full implementation using PyTorch with the different chosen hyper-parameters refer to the code's ULMFit section.

D. Bidirectional Encoder Representations from Transformers (BERT)

BERT is a recent approach to learning language representation, including the context which isn't the case in GloVe embeddings. It "is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers [18]". Thus, in our case, we use a pre-trained model, add a classifier on top and then fine-tune the model for our task. We use the python library `Transformers` [19] for the implementation.

1) Pre-trained model: `Transformers` proposes a wide variety of pre-trained models. We started with DistilBERT [20], a distilled version of BERT with less layers, as they train faster using `distilbert-base-uncased`. After initial tests where conducted, we used the bigger model of `bert-base-uncased` as our base pre-trained model. Thus the according Tokenizer is used to encode the preprocessed data before feeding it to the model.

2) Classifier: The classifier is a 1-hidden-layer feed forward Network of the shape $(768 \rightarrow 768 \rightarrow 2)$, with Tanh activation and a 0.2 Dropout. It is added on top of the output of the last layer hidden-state of the first token of the sequence. This is specified for classification in [18] as the first token is a special `[CLS]` token.

3) Fine-tuning and Validation: We used the same splits as for the GRU model, but trained over it for 3 epochs, as each epoch took more than 3 hours to complete on a Nvidia P100. The optimizer was AdamW with $\eta = 1e-5$ and the gradient norms were clipped at 1. For the evaluation, we selected the model of the epoch with the highest validation.

IV. Results and Discussion

In table I we present the cross-validation results from the training of the baseline classifiers. As elaborated previously, we selected the baseline model based on the F-1 validation score and the neural network showed the best performance. It can be observed that in general the performance of the different classifiers does not vary by large amounts, however all models showed a much higher recall than precision value, indicating that the amount of false negatives had been much lesser than of the false positives. As the class distribution of the data is balanced, this would indicate that with our extracted features true negative tweets were much easily distinguished.

Classifier	Parameters	Accuracy	Precision	Recall	F-1
Naive Bayes	`var_smoothing = 1e-18`	0.7226	0.6766	0.8481	0.7527
Logistic Regression	`C = 1e4` `solver = lbfgs`	0.7536	0.7238	0.8163	0.7673
Linear SVMs	`C = 1e2`	0.7524	0.7186	0.8257	0.7685

Neural Network	`hidden_layer_sizes = (100,)`
	`activation = relu`
	`alpha = 1e-4`
		0.7617	0.7308	0.8255	0.7752

Random Forest	`n_estimators = 1000`	0.7628	0.7347	0.8192	0.7747

Table I
Cross-validation results of baseline classifiers

Although with the baseline approach we achieved much lower scores compared to the other methods, as will be presented further below, we considered this approach in order to improve the interpretability of our results and to attempt to discover language patterns in the tweet data using the crafted features. This was also another motivating reason on the choice to perform the feature selection process. From the plot displayed on figure 3, depicting the class distribution of some of the best descriptors, we can make many interesting observations about the nature of the tweet data.

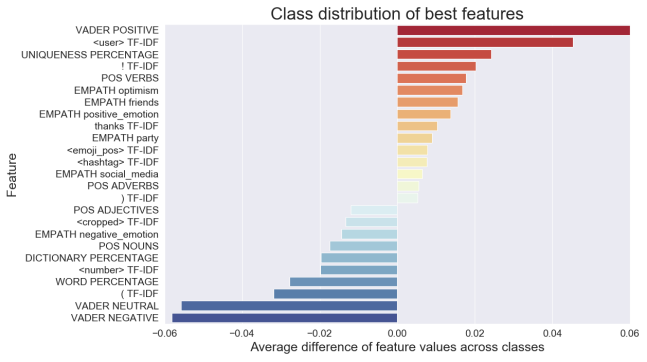

Figure 3. Plot of average difference of feature values across the classes

We observe that positive tweets more frequently: have a higher number of unique words; a higher positive Vader polarity score; contain tokens such as "`<user>`", "`!`", "`thanks`", "`<emoji_pos>`", "`)`"; verbs and adverbs are more common than other morphology and contain tokens from the Empath categories "optimism", "friends" and "positive_emotion".

On the other hand, for the negative tweets: higher language quality is observed (higher word and dictionary percentage); neutral and negative polarization was more frequent; nouns and adjectives have higher frequency; tokens such as "`<cropped>`", "`<number>`", "`miss`", "`wish`" and "`(`" and the prevalent Empath topics were more often "negative_emotion" and even "death".

As it seems, the feature selection process on the baseline features automatically discovers very general and commonsense ways to distinguish the tweets, but nevertheless it leads to a very efficient characterization. However, a surprising result is the discovery of the very high predictive power of the TF-IDF values of the bracket tokens "`)`" and "`(`". Upon further investigation, we conjecture that these tokens are remnants of the initial annotation and processing performed on the tweet dataset i.e. incomplete removals of the emojis ":)" and ":(", as the appropriate bracket is a good predictor of each class.

In table II we present the results of each of the four methods on the test set. We observe that with the latter three methods we are able to achieve a performance increase of more than 10% in both accuracy and F-1 score.

Method	Submission ID	Accuracy	F-1
Classic ML	26044	0.770	0.783
GRUs w/o preprocessing	24552	0.873	0.876
GRUs w/ preprocessing	24756	0.877	0.879
GRUs multi-layer	24841	0.881	0.883
ULMFit	27520	0.885	0.886
DistilBERT (base-uncased)	28613	0.895	0.895
BERT (base-uncased)	30617	**0.904**	**0.904**

Table II
Final submission results

V. Conclusion

We proposed a baseline based on traditional machine learning techniques that showed that we can extract meaningful features from the tweets for our classification task. At the expense of longer training times and bigger infrastructures, we devised three Deep Learning architectures, with our best performing one, a BERT model, reaching 0.904 accuracy and F-1 score on the test set.

Provided more time and computing power, it would be interesting to explore ensemble methods, such as bagging and different ways to combine the trained models, to further improve our final score as it is commonly done in competitions. As papers on this field are published at frantic pace, one could keep an eye on them like for example XLNet [21].

References

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

[2] M. Honnibal and I. Montani, "spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing," 2017, to appear.

[3] C. J. Hutto and E. Gilbert, "Vader: A parsimonious rule-based model for sentiment analysis of social media text," in *Eighth international AAAI conference on weblogs and social media*, 2014.

[4] E. Fast, B. Chen, and M. S. Bernstein, "Empath: Understanding topic signals in large-scale text," in *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, 2016, pp. 4647–4657.

[5] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. " O'Reilly Media, Inc.", 2009.

[6] 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.

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems*, 2019, pp. 8024–8035.

[8] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[9] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, 2016, pp. 1019–1027.

[10] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[11] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[12] E. Arisoy, T. N. Sainath, B. Kingsbury, and B. Ramabhadran, "Deep neural network language models," in *Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT*, ser. WLM '12. Stroudsburg, PA, USA: Association for Computational Linguistics, 2012, pp. 20–28. [Online]. Available: http://dl.acm.org/citation.cfm?id=2390940.2390943

[13] R. C. Staudemeyer and E. R. Morris, "Understanding lstm – a tutorial into long short-term memory recurrent neural networks," 2019.

[14] 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: http://jmlr.org/papers/v15/srivastava14a.html

[15] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay," 2018.

[16] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *CoRR*, vol. abs/1502.03167, 2015. [Online]. Available: http://arxiv.org/abs/1502.03167

[17] R. Kemker, M. McClure, A. Abitino, T. Hayes, and C. Kanan, "Measuring catastrophic forgetting in neural networks," 2017.

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

[19] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, and J. Brew, "Huggingface's transformers: State-of-the-art natural language processing," *ArXiv*, vol. abs/1910.03771, 2019.

[20] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[21] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R. Salakhutdinov, and Q. V. Le, "Xlnet: Generalized autoregressive pretraining for language understanding," *arXiv preprint arXiv:1906.08237*, 2019.