Project 2 - Text Sentiment Classification

Clémence Barsi (CS), Pauline Conti (DS) and Younes Moussaif (GM)
clemence.barsi@epfl.ch, pauline.conti@epfl.ch, younes.moussaif@epfl.ch

***Abstract*—Text sentiment classification is a very common application of natural language processing (NLP). We used different embeddings, classifiers and models in order to compare their performances on a simple sentiment analysis task. The best results were obtained using BERT for sequence classification, scoring 90.1% accuracy on AI-crowd.**

I. Introduction

The objective of the project is to do sentiment analysis on a dataset of Tweets. After analyzing 2'500'000 positive and negative tweets, the aim is to be able to tell if a tweet contained a positive or negative smiley solely based on the text inside it.

II. Exploratory data analysis & preprocessing

The exploratory data analysis (EDA) is the first step when choosing what models to use given a machine learning objective on a dataset. In the case of text data, a lot of the EDA is done concurrently to some preprocessing steps in order to be able to better understand the data. We have four datasets which are pairs of datasets containing only positive or negative tweets: two large ones summing up to 2.5 Million tweets and two small ones summing up to 200'000 tweets.

In order to help us direct our preprocessing regarding stop-words or punctuation, we decided to start the EDA by obtaining the ten most frequently occurring strings when considering words, stop-words and punctuation. To obtain those metrics, we first lower-cased each tweet, then removed the contractions and slang using dedicated dictionaries [1] [2] [3]. The rest of the pre-processing steps used different functions and dictionaries from the NLTK library [4].

Finally, we stemmed the resulting words and separated the punctuation and stop-words (both are removed for the final preprocessing). Fig. 1 shows the resulting ten mostbot occurring tokens (stemmed words without stop-words), stop-words and punctuation marks in the full positive and negative datasets. The most occurring word is *love* for positive tweets and *frame* for negative tweets. We can see that $< url >$ (a link is in the tweet) is more frequent for negative tweets. Finally, while the punctuation marks are mainly the same in different order, ")" is frequently used for positive tweets, while "(" is for negative ones, which coincides with the use of happy or sad smileys.

After these steps, we decided to remove any token that appeared less than five times in the large datasets to remove some of the noise from the vocabulary. Indeed, any word appearing less than five times would not appear enough times to really help categorize any tweet as positive or negative, while augmenting the size of the vocabulary.

III. Models Used and Fitting

A. General Presentation and Overview

When choosing which model to use for sentiment classification, one is faced with the choice of embedding as well as the choice of classifier. In order to assess the performance of each, we chose and implemented different combinations of them for a more accurate comparison.

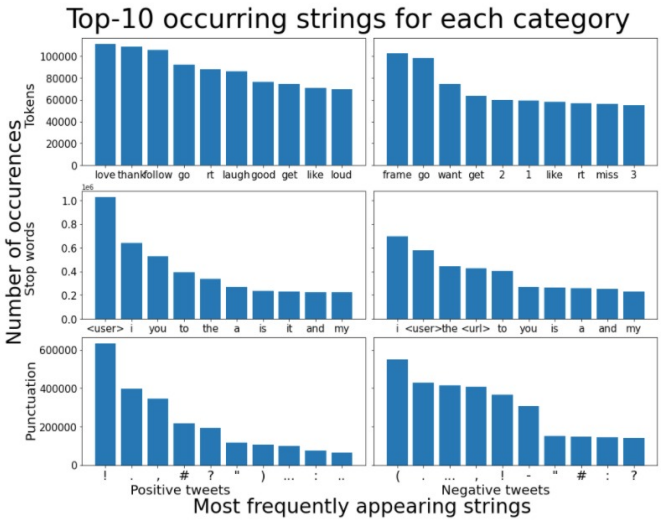

Figure 1. Histogram plots the occurrence frequency of tokens, stop-words and punctuation in the positive and negative tweets. Note: the y-axis for the stop-words is expressed in millions.

For the embeddings, we chose to use TF-IDF [5] and GloVe as they are two widely used embedding systems which are supposed to capture two different aspects for language. Indeed, TF-IDF focuses on a token's (word or characters) frequency in its "document" (tweet in this case) weighted by its frequency in the corpus, encapsulated in a sparse representation. GloVe, on the other end, is a pre-trained model that uses a dense co-occurrence matrix to extract semantic meaning.

For classifiers, we chose to compare three models, varying the level of complexity. The most simple model chosen was Naive Bayes, to be used as a baseline. We then chose to use Logistic Regression with L2 Regularization as it is used in many machine learning classification problems and remains relatively simple to implement and train. Finally, we decided to use a Neural Network for a more "advanced" classifier as they are able to properly handle complex tasks such as natural language processing.

In addition to those models, we chose to also incorporate some more recent and complex pre-trained models that already included their own embeddings. Ever since its paper was published in 2018, BERT [6] has been praised for its ability to tackle diverse and complex NLP problems. We thus decided to use it and build upon it. Indeed, we used BERT for Sequence Classification as well as BERT with our own Neural Network Classifier. This allowed us to evaluate the impact of BERT's embedding and its transformer-based structure, as well as use tools made available by modern ML research in order to get the best results possible.

B. Embedding Methods

1) TF-IDF: TF-IDF [5] stands for *Term Frequency – Inverse Document Frequency*. This type of embedding gives

more importance to words in a document that are overall less frequent in the corpus (collection of documents), but frequent with respect to the document, as they would be more representative of the document. Since the matrix can be quite large we used sparse representation in the *sklearn* library, and used the top 10000 most common words and n-grams (with n in $\{1, 2, 3\}$).

2) GloVe: GloVe stands for Global Vectors for word representation. It is an unsupervised learning algorithm that focuses on word co-occurence. It is based on the idea that a word often co-occurs with the same words rather than other ones. It maps the words into a meaningful space where the distance between each word is representative of the semantic similarity. Training is performed on aggregated global word-word co-occurences statistics. In our case we used GloVe embeddings of dimension 25 that were pretrained on 27 billion tweets.[7][8][9], and we used the average of a tweet's word embeddings as its embedding (number vector).

C. Classifiers

1) Naive Bayes: As baseline, we used a simple model: Naive Bayes [10], implemented with the *scikit-learn* library [11]. It is very fast to train compared to other classification algorithms and is known to outperform some more sophisticated algorithms. Moreover, it doesn't have hyper-parameters to tune. That's why we chose it for baseline. It uses the Bayes theorem of probability to determine the probability that an unknown point belongs to each class assuming independence between the features. Then it assigns this point to the class with the highest probability.

We used both GloVe embeddings and TF-IDF embeddings with Naive Bayes. With 5-fold cross-validation, we obtained an average accuracy of 0.6273 ($std = 0.0052$) with GloVe embeddings, and an average accuracy of 0.7505 ($std = 0.0016$) with TF-IDF embeddings, achieving a higher accuracy and a 5 times smaller standard deviation.

2) Logistic Regression with L2 Regularization: Then, we implemented logistic regression using the *scikit-learn* library [11]. We chose logistic regression because it trains more quickly than more complex models [12] but is more sophisticated than Naive Bayes and thus could refine our predictions. It is also one of the most used machine learning algorithm for binary classification tasks. Logistic regression determines a relationship between the probability of a particular outcome and the features.

To optimize our model, we did a hyperparameter grid search on the penalizing term C and on the solvers. The term C should be a positive float, with default value set to 1.0. We decided to try five values between 100.0 and 0.01 to see the performance of different orders of magnitude [11][13] (smaller *C* = stronger regularization). For the solver, we chose to compare only two options: *"lbgfs"* (default parameter) and *"saga"*. We chose *"lbgfs"* as it is based on Newton's method but is more efficient (approximates the Hessian matrix). In the case of *"saga"*, it performs well and fast on large datasets according to the *scikit* documentation [11]. We performed 5-fold cross-validation on the full dataset in order to also measure the standard deviation of each combination of parameters and choose accordingly.

Table I
AVERAGE TRAINING ACCURACY AND STD WHEN USING CROSS-VALIDATION ON **LOGISTIC REGRESSION** WITH DIFFERENT HYPER-PARAMETERS

Model		GloVe		TF-IDF	
C	Solver	Training accuracy	Std	Training accuracy	Std
100.0	lbfgs	0.661084	0.000564	0.780855	0.000695
	saga	0.661088	0.000562	0.781389	0.000650
10.0	lbfgs	0.661084	0.000564	0.780964	0.000659
	saga	0.661086	0.000564	**0.781393**	**0.000643**
1.0	lbfgs	0.661084	0.000564	0.781213	0.000636
	saga	**0.661089**	**0.000564**	0.781385	0.000636
0.1	lbfgs	0.661082	0.000564	0.779859	0.000612
	saga	0.661087	0.000564	0.779850	0.000614
0.01	lbfgs	0.661081	0.000564	0.76972	0.000543
	saga	0.661080	0.000565	0.762970	0.000549

	learning rate				
Activation	0.1	0.01	0.001	model	
ReLU	0.7016 (0.00256)	0.7335 (0.0027)	0.7325 (0.0017)	GloVe	accuracy
Tanh	0.6989 (0.0033)	0.7302 (0.0021)	**0.7357 (0.0023)**		
Logistic	0.7179 (0.0032)	0.7328 (0.0022)	0.7311 (0.0047)		
ReLU	0.7209 (0.0021)	0.7604 (0.0033)	0.7650 (0.0017)	TF-IDF	
Tanh	0.7197 (0.0033)	0.7635 (0.0032)	**0.7663 (0.0022)**		
Logistic	0.7175 (0.0028)	0.7266 (0.0039)	0.7348 (0.0038)		

Table II
5-FOLD CROSS-VALIDATION ACCURACIES (STD) OF THE **MLP** MODELS, WITH DIFFERENT LEARNING RATES AND ACTIVATION FUNCTIONS.

In the table of results I, we can see that TF-IDF embeddings allow to achieve a clearly better accuracy than GloVe embeddings. Indeed, whereas all the training accuracies are relatively similar when using a specific embedding, using TF-IDF over GloVe allows to gain at least 10 points of accuracy while keeping a fairly low standard deviation across runs of the cross-validation. The combination that turned out to perform the best was the solver *"saga"* combined with a penalizing term of $C = 10.0$ when using TF-IDF embeddings, with the average training accuracy of 0.7814. When we used GloVe embeddings, the best combination was a penalizing term of $C = 1.0$ and a *"saga"* solver, with an average training accuracy of 0.661089. However for GloVe, it is worth noting that the changes in accuracy when using different combinations of hyperparameters are very slight, in the order of $10^{-4}\%$. This suggests that the impact of the parameters on the model's performance is very minimal compared to the embedding's. This leads us to think that TF-IDF's embedding is more adapted to the problem of Text Sentiment Classification than the pretrained GloVe embeddings we used. Overall, the results let us think that another classifier or model could perform better, with the best average training accuracy being shy of 80%.

3) Multi-Layer Perceptron: Following logistic regression, in an attempt to better our results we implemented a multi-layer perceptron [14] using the *scikit-learn* library [11]. This model takes longer to train than previous ones but has the benefit over logistic regression that it also works well on non-linearly separable data [12]. Furthermore, logistic regression works better than neural networks when there isn't a lot of data nor features, which is not our case [12]. Multi-layer perceptron has a good ability to learn any mapping function, they learn the representation of the training data and how to relate it to the outcome.

In order to fine tune the model, we performed a grid search with 5-fold cross-validation, but only on the small dataset

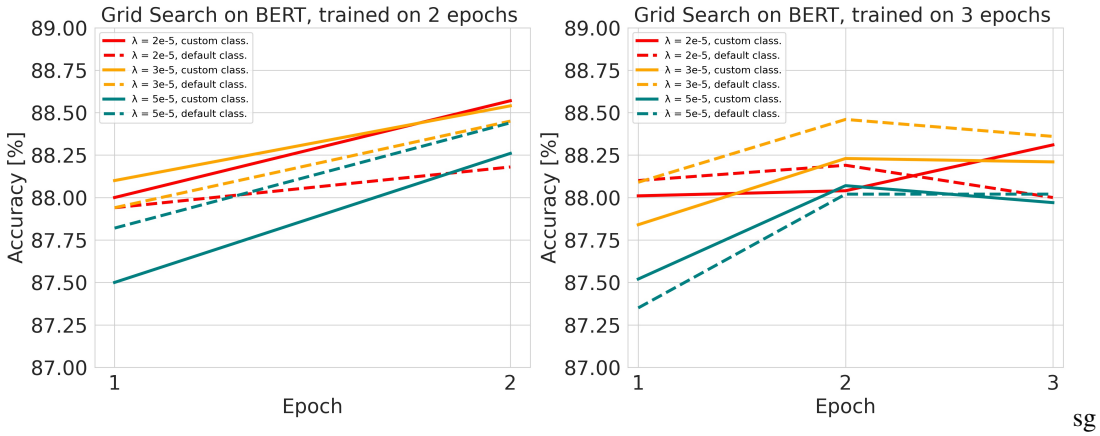

Figure 2. Validation accuracies of the two BERT models. Dotted lines for *BERT for Sentence Classification*, full lines for *BERT with our Custom Classifier*. Note: Due to the very close results, the y axis goes from 87 to 89% (for increased visibility of the superimposed areas)

due to the long training times with the full dataset. The hyperparameters included in the grid search were the activation function – for which we used ReLU, Tanh and Sigmoid – and the learning rate for Adam. The maximum number of epochs were 50 or 5, and a regularization parameter of value 0.0001 for GloVe and TF-IDF embeddings respectively. A higher regularization parameter was used with TF-IDF due to the higher number of features. Both were trained while activating $early\_stopping = True$ which stopped training if the validation accuracy didn't increase for 2 iterations. As we can see in Table II, the best parameters were the Tanh activation and a learning rate $lr = 0.001$ for both TF-IDF and Glove embeddings giving accuracies of respectively 0.7663 and 0.7357.

D. BERT Models

The BERT model [6], which stands for *Bidirectional Encoder Representations from Transformers*, relies on previous findings such as Transformers [15]. Its use of bi-directional contexts when training on very large amounts of data yielded very convincing results.

We decided to use the existing libraries like *HuggingFace* [16], providing us with pre-trained versions of BERT Tokenizers and BERT models to improve upon the results more traditional techniques would yield. For classification in a sentiment analysis task, the model *BertForSequenceClassification* comes with pre-trained embedding layers, and a randomly initialized classification layer [17] [18]. However, we wanted to try and improve our results by using our own Neural Network classifier on top of BERT to measure the impact of having a more complex classifier after BERT.

1) BERT pre-processing: In this case, the default pre-processing from *BERT Tokenizers* was used.[19][20] Since additional steps like stop words removal weren't required as BERT is a contextual model and relies on sentences. The BERT Tokenizer lowercases, removes accents, and encodes each word and subwords (parts of words) to a number token from its pre-trained vocabulary.

2) BERT for Sequence Classification: As previously stated, we started by using *BertForSequenceClassification*. It simply consists in the original BERT model with a binary classification layer added on top. As the model is pre-trained, our training consisted on fine-tuning the model to our data after selecting which hyper-parameters to use.

3) BERT with Custom Neural Network Classifier: Since the *BertforSequenceClassification* model had pre-trained embedding layers and a classification head composed of a fully connected layer with dropout it seemed promising to implement a similar neural network with an extra layer and evaluate the model's performance. Using a classifier with more capacity could maybe increase the model's generalisation ability. [21] [22] The classifier trained in our experiments takes as input the 786 outputs of BERT's pooling layers, feeds it to a linear layer with 500 outputs, passes throught a Tanh activation and finally a linear layer with 2 output logits. The number of hidden units was kept at 500 as taking a lower value decreased accuracy while increasing it slowed down training.

4) BERT hyperparameter selection: As most of the hyperparameters were the same in both BERT models, we decided to perform grid search on the same values on both models. The first parameter is the maximum sentence length to pad or truncate to when encoding our sentences to tokens using the *BertTokenizer*. This value, *max_len* corresponds to the constant number of IDs that the tokenizer will assign for each word (token). Tweets have a maximum of 280 characters (including), so we first chose the value of 140. However, this resulted in very long training times, reaching more than 6 hours for one epoch on the full dataset. As a result, we decided to decrease to $max\_len = 40$, as a vast majority of tweets contain less than 40 words [23][24]. This helped considerably speed-up the training time (∼3h/epoch with a Nvidia P100 GPU).

As training parameters, there was the batch size, the number of epochs, the learning rate for the AdamW optimizer and its scheduler. We chose to keep the batch size fixed to 32 as it was the value advised in the original paper [6]. We also decided to keep the same scheduler: for a total $num\_training\_steps = batch\_size \times n\_epochs$, we chose to have $num\_warmup\_steps = 0.1 \times num\_training\_steps$, as recommended in the paper. The scheduler allows for the learning rate to change during the training. During warm-up, the scheduler linearly increases the learning rate from 0 to the chosen learning rate, after it linearly decreases the learning rate to 0 at the last training step [25]. For an optimizer like AdamW, they allow to compute correct statistics of the gradients and help the network slowly adapt to the data at the start of the training

Table III
VALIDATION & TEST ACCURACIES OBTAINED FOR THE MODELS USED.

Model	**GloVe** Accuracies		**TF-IDF** Accuracies	
	Validation	Test (AI-crowd)	Validation	Test (AI-crowd)
Naive Bayes	0.627	0.578	0.751	0.742
Logistic Regression	0.661	0.649	0.781	0.770
Multi-Layer perceptron	0.729	0.713	0.782	0.771
	BERT Accuracies			
BERT Sentence Classification	0.9159	0.901		
BERT Custom Classifier	0.9167	0.899		

Model	BERT Sentence Classification		BERT Custom Classifier	
`max_len`	140	40	140	40
Accuracy (ai-crowd)	0.901	0.899	0.899	0.898

Table IV
VALIDATION SCORE ON AI-CROWD FOR THE BERT MODELS WITH DIFFERENT MAXIMUM LENGTH VALUES.

[26] [27]. Also AdamW, which decouples L_2 regularization and weight decay, improves Adam's generalization [28]. The number of epochs was either 2 or 3, and the learning rate was grid-searched between $5e-5$, $3e-5$ and $2e-5$.

Given that the training of one epoch took 30 minutes on the small dataset and around 3 hours on the full dataset, we chose not to perform cross validation, but simply a grid search over 80% of the small dataset, validating on the remaining 20%. This also motivated our choices of restricting the grid-search to a few options per hyper-parameter and to fix some of them to constant values.

As we can see from Fig. 2, the validation accuracy (on the small dataset) of both BERT models was better when training over 2 epochs rather than 3. The overall highest validation accuracies obtained for *BERT for Sentence Classification* was 0.8845, using $lr = 2e-5$, and training over 2 epochs. For *BERT with Custom Classifier*, it was 0.8857, using the same hyper-parameters. Additionally from Figure 2, it seems that our custom classifier did improve slightly BERT's performance. However, keeping in mind that all the validation accuracies are very close to each other, it is clear that the classifier is not the motor of the model's good performance. Indeed, all those results are in the range of 87-89% validation accuracy which are close to the numbers announced by the authors of BERT [6]. We can imagine that the embedding as well as the stacked transformers architecture are responsible for those performances. In particular, those models systematically achieve accuracies more than 10 points above the other models we used, even after the first epoch of training. This hints to the fact that the bidirectional context that BERT implements, and the large dataset on which it was pre-trained are key in the model's success on NLP tasks.

IV. RESULTS COMPARISON & INTERPRETATION

We can see that Naive Bayes performs better with TF-IDF word embeddings, likely due to large number of features it has. This makes sense as having more dimensions can lead to problems due to the *curse of dimentionality* but naive bayes assumes the features are independent, with GloVe's small amount of features it would seem more likely that features will interact with each other versus when there is a large number of features, which TF-IDF embeddings offer.

Logistic regression performs better than Naive Bayes and when using TF-IDF embeddings we gained 10 points of accuracy in both cases. While it would seem like a semantic mapping like GloVe should perform better, TF-IDF seems to represent our dataset better, since the GloVe embeddings we used weren't trained on our dataset, and TF-IDF uses words and n-grams from the training tweets. In addition to that, the semantics of the words might reveal less information on the positive or negative aspect of a sentence, whereas some frequently occurring words in the positive tweets might be understood as positive by the models, and thus, TF-IDF's embedding scheme seems more adapted to thisa classification problem than GloVe.

The MLP model has the best score amongst the baseline models but it is very close to the logistic regression scores. Although it performed better overall with TF-IDF embeddings it also performed better than the other models with the GloVe embeddings. Since, even with TF-IDF, the number of features is still lower than the number of documents, the features are likely not independent since some words co-occur in text and have complex non-linear relation-ships, that goes against naive Bayes's assumption and could maybe be modeled by an MLP.[29]

From the results in Table III, we can see that both Bert models perform a lot better than the other models. They are more sophisticated and capture more meaning in the tweets The test data tokenization could be done after truncating it to either 40 or 140 tokens, the latter yielded better testing results on ai-crowd, and was found by mistake. It seems that the extra information brought by all of the tokens is very important in the testing phase, whereas in the training phase, the loss of information from the truncation to a length of 40 could be compensated by the large quantity of data as tweets containing more than 40 words are a clear minority. It is a trade-off, we chose to keep the length to 40 as it would otherwise take $8h \cdot 2\ epochs$ versus $3h \cdot 2\ epochs$ to complete training. Finally, we see that the standard BERT classifier does marginally better than the one with a custom classifier (90.1% versus 89.9% test accuracy respectively), this could be due to variance in the model's training (since we couldn't cross-validate on the large data-set in a reasonable amount of time) or simply because a hyper parameter optimization wasn't performed on the entire data-set. Otherwise, this insists more on the importance of the embedding layers versus the classification layers since the extra classification layer didn't significatly improve the results.

V. CONCLUSION

To conclude, as expected both BERT models are the best performing models. With test accuracies reaching 90.1%, they have 10 points more in accuracy than the other models. In addition to that, we saw that the embeddings chosen do have a very big impact on a model's performance if it is not adapted to the task. Future improvements could include using a deeper MLP classifier, a CNN, using an ensemble of BERT models [30], or random-forests. We could pretrain our own GLoVe embedding, or improving training performance of BERT by using a smaller version of BERT like packed BERT[31] or distil-BERT[32].

REFERENCES

[1] B. Han and T. Baldwin, "Lexical normalisation of short text messages: Makn sens a #twitter," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Portland, Oregon, USA: Association for Computational Linguistics, Jun. 2011, pp. 368–378. [Online]. Available: https://aclanthology.org/P11-1038

[2] Wikipedia. List of english contractions1. [Online]. Available: https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions

[3] ——. List of english contractions2. [Online]. Available: https://gist.githubusercontent.com/Sirsirious/c70400176a4532899a483e06d72cf99e/raw/e46fa7620c4f378f5bf39608b45cddad7ff447a4/english_contractions.json

[4] S. Bird, E. Klein, , and E. Loper, *Natural Language Processing with Python.* O'Reilly Media Inc., 2009. [Online]. Available: https://www.nltk.org/book

[5] C. V. den Rul. Understanding word embeddings with tf-idf and glove. [Online]. Available: https://towardsdatascience.com/understanding-word-embeddings-with-tf-idf-and-glove-8acb63892032

[6] J. Devlin, Ming-Wei, C. K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv*, no. 1810.04805, 2018. [Online]. Available: https://arxiv.org/pdf/1810.04805.pdf

[7] J. Pennington, "Glove: Global vectors for word representation." [Online]. Available: https://nlp.stanford.edu/projects/glove/

[8] J. Pennington, R. Socher, and C. D. 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.

[9] "Gensim/glove-twitter-25 · hugging face." [Online]. Available: https://huggingface.co/Gensim/glove-twitter-25

[10] (2021) Naive bayes classifier. [Online]. Available: https://en.wikipedia.org/wiki/Naive_Bayes_classifier

[11] 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.

[12] D. Varghese. (2018) Comparative study on classic machine learning algorithms. [Online]. Available: https://towardsdatascience.com/comparative-study-on-classic-machine-learning-algorithms-24f9ff6ab222

[13] J. Brownlee. (2019) Tune hyperparameters for classification machine learning algorithms. [Online]. Available: https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/

[14] ——. (2016) Crash course on multi-layer perceptron neural networks. [Online]. Available: https://machinelearningmastery.com/neural-networks-crash-course/

[15] J. Uszkoreit. (2017) Transformer: A novel neural network architecture for language understanding. [Online]. Available: https://ai.googleblog.com/2017/08/transformer-novel-neural-network.html

[16] Hugging face. [Online]. Available: https://huggingface.co/

[17] [Online]. Available: https://huggingface.co/docs/transformers/training

[18] Huggingface, "modeling_bert.py at v4.14.1 · huggingface/transformers - line 1506 to 1510," Nov 2021. [Online]. Available: https://github.com/huggingface/transformers/blob/v4.14.1/src/transformers/models/bert/modeling_bert.py#L1496

[19] Tokenizer. [Online]. Available: https://huggingface.co/docs/transformers/main_classes/tokenizer

[20] V. Lendave. A guide to text preprocessing using bert. [Online]. Available: https://analyticsindiamag.com/a-guide-to-text-preprocessing-using-bert/

[21] M. Belkin, D. Hsu, S. Ma, and S. Mandal, "Reconciling modern machine learning practice and the bias-variance trade-off," *arXiv preprint arXiv:1812.11118*, 2018.

[22] M. Telgarsky, "Benefits of depth in neural networks," in *Conference on learning theory*. PMLR, 2016, pp. 1517–1539.

[23] A. Boot, E. Tjong Kim Sang, and K. Dijkstra, "How character limit affects language usage in tweets," *Palgrave Commun*, vol. 5, no. 76, 2019. [Online]. Available: https://doi.org/10.1057/s41599-019-0280-3

[24] ——, "How character limit affects language usage in tweets figure 6," *Palgrave Commun*, vol. 5, no. 76, 2019. [Online]. Available: https://www.nature.com/articles/s41599-019-0280-3/figures/6

[25] Hugging face – transformers, get linear schedule with warmup. [Online]. Available: https://huggingface.co/docs/transformers/main_classes/optimizer_schedules#transformers.get_linear_schedule_with_warmup

[26] L. N. Smith, "A disciplined approach to neural network hyper-parameters: Part 1–learning rate, batch size, momentum, and weight decay," *arXiv preprint arXiv:1803.09820*, 2018.

[27] L. N. Smith and N. Topin, "Super-convergence: Very fast training of neural networks using large learning rates," in *Artificial Intelligence and Machine Learning for Multi-Domain Operations Applications*, vol. 11006. International Society for Optics and Photonics, 2019, p. 1100612.

[28] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[29] H. n. Shimodaira, "Notes on naiv bayes," December 2021. [Online]. Available: https://www.inf.ed.ac.uk/teaching/courses/inf2b/learnnotes/inf2b-learn06-notes-nup.pdf

[30] H. Dang, K. Lee, S. Henry, and O. Uzuner, "Ensemble bert for classifying medication-mentioning tweets," in *Proceedings of the Fifth Social Media Mining for Health Applications Workshop & Shared Task*, 2020, pp. 37–41.

[31] M. Kosec, S. Fu, and M. M. Krell, "Packing: Towards 2x nlp bert acceleration," *arXiv preprint arXiv:2107.02027*, 2021.

[32] 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.