A Comparison of Techniques for Sentiment Classification

Jonas Hübotter, Franz Nowak, Fiona Muntwyler, and **Saahiti Prayaga**
ETH Zürich - Swiss Federal Institute of Technology
{jhuebotter, fnowak, fionamu, prayagas}@ethz.ch

Abstract

We evaluate multiple models for sentiment classification of tweets, achieving a maximum accuracy of 90%, and showing that transformer-based architectures (BERT) outperform standard classification algorithms (Naive Bayes, XGBoost, Logistic Regression) that use averaged word embeddings. Finally, we evaluate a novel clustering-based approach aimed at improving training efficiency, which does not lead to any significant performance gains on the considered corpus.

1 Introduction

Every second, around 6000 messages of up to 140 characters are posted on the social network Twitter[1], making it a figurative gold mine for companies and organizations who want to gauge the opinions of its hundreds of millions of users. One part of that process is to analyse the sentiment of these so-called tweets, i.e. recognizing whether they are expressing positive or negative sentiment.

Methods for sentiment classification range from standard classification methods such as Logistic Regression and Bayesian classifiers (Vikramkumar et al., 2014) to more sophisticated transformer-based generative language models such as BERT (Devlin et al., 2018).

In this work, we compare a range of different machine learning models on a corpus of 2.5M tweets on their ability to learn the task of binary sentiment classification. The main contributions of this work can be summarized as follows:

- We train a number of simple classifiers to attain a baseline for the task, using various types of word and subword embeddings.
- We train multiple pre-trained BERT models on the task, comparing the effects of warm start vs fine-tuning.
- We investigate a clustering-based approach that could potentially make training of large language models more efficient, but which on this corpus did not lead to a significant efficiency gain.
- We achieve an accuracy of 0.90 on the task with our best model.

We first introduce the different baseline models in section 2. We also implement and assess the effects on performance of different preprocessing techniques (see appendix A). We then train and optimize BERT in section 3, and explore an idea for a more efficient training regime. Our results are presented in section 4 and their implications discussed in section 5.

The code for our implementations of the presented models can be found on our GitHub repository[2].

2 Baseline Models

To make a meaningful comparison of our model to standard approaches, we implement a number of baseline models for classification tasks, including two state-of-the-art language models. A brief overview of these approaches is given here, and more details are provided in appendix B.

First, we implement a Naive Bayes Classifier (Jurafsky and Martin, 2009) using a Bag of Words approach, which is known to produce reasonable results. This model is simple to implement and therefore an ideal first baseline. For more details, refer to appendix B.1.

Second, we implement the XGBoost algorithm (Chen and Guestrin, 2016) using GloVe embeddings and utilizing the implementation provided by the XGBoost library.[3] For more details, refer to appendix B.2.

Third, we implement a Logistic Regression model using fastText embeddings using the "fastText for Text Classification" library (Joulin et al., 2017). It offers a decent tradeoff between accuracy and training efficiency. For an overview of logistic regression and more details on the implementation, refer to appendix B.3.

[1] https://www.internetlivestats.com/twitter-statistics/

[2] https://github.com/franznowak/twitter-sentiment-classification

[3] https://xgboost.readthedocs.io/en/stable/python/python_api.html

Fourth, we implement BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018), a transformer-based language representational model. For an overview of the BERT architecture, see appendix B.4. Since BERT models take enormous computational resources to train, training a model for our task from scratch is not a suitable approach, and we therefore make use of a pre-trained model such as DistilBERT. We used PyTorch[4] and Hugging Face[5] to implement our BERT models.

3 Improving BERT

Instead of simply using a pre-trained model that classifies language data into two classes based on sentiment on our test data without training, we applied the following approaches to improve the model.

3.1 Fine tuning

As opposed to retraining all parameters of a pre-trained model, a common approach is to fix the majority of a model's parameters and only train a subset of selected parameters with the new data. For pre-trained BERT models, these parameters are typically the ones on the output classification layer and the ones of the intermediate layer norms. It was shown in (Lu et al., 2021) that transformer models are even capable of generalizing to different input modalities when pre-trained on language data and only fine-tuning the above-mentioned parameters, which is why fine-tuning these parameters for our task is a reasonable approach.

To fine-tune our model, we have added a single fully-connected layer with two output neurons, representing the two classes. We have trained three different models with different sets of parameters set to trainable:

1. All parameters (1 epoch)
2. Output layer and layer norm parameters (LN) (5 epochs)
3. Output layer parameters (5 epochs)

The number of epochs for which we train each model differs, since the number of trainable parameters strongly impacts the training time and how fast a model tends to overfit.

3.2 Better embeddings

When fine-tuning a pre-trained model, it may be the case that performance is improved when the pre-trained model was trained on data and for a task that is similar to the data and task at hand. For this reason, we used the Twitter roBERTa Base for Sentiment Analysis model[6] as our base for fine-tuning. It is a roBERTa-based model which was trained on approximately 124M English tweets and then fine-tuned for sentiment analysis with the TweetEval benchmark. It classifies tweets into the three categories Positive, Negative, and Neutral. The difference between BERT and roBERTa is that the latter uses an improved masking strategy.

3.3 Specialized Sub-networks

The fact that for our project, we only have access to a single GPU for training prompted us to try and find more efficient ways to train the computationally heavy BERT models. We came up with the following idea:

Instead of training a single model on all the data until convergence, we first divide the training and test data into k disjoint clusters. We then measure the performance of a base BERT model we trained for one epoch on all of the training data for each individual cluster. Finally, we retrain separate BERT models on just those clusters where performance was low, so that these models are trained only to optimize classification performance for the specific data of that cluster type. The idea is that this will avoid wasting time training BERT on parts of the data distributions in which data are easy to classify (where performance is already good enough), and instead have more specialized models that focus on disjoint sets of "harder" examples. Our hypothesis is that these harder examples can be attributed to more complex data sub-distributions, which will require more training.

3.3.1 Emotion Clusters

The first attempt at dividing the data was to classify tweets into finer grained emotion and use the emotion label as clusters. Bostan and Klinger (2018) provide a thorough analysis of a number of available emotion classification datasets. Out of these, the one best suited for our purpose was the CROWDFLOWER dataset[7], which contains 40,000 tweets labelled with specific emotions. Using the mapping from Bostan and Klinger (2018) with a small adjustment[8], these were mapped to the categories identified by Plutchik (2001) (see fig. 1). We then trained a base Bert model to classify all training and test tweets. We obtained $k = 7$ classes, which were those out of the above categories that were encountered in the training data, namely: *joy, fear, sadness, love, surprise, anger, and additionally noemo for none of the above*.

[4] https://www.pytorch.org
[5] https://www.huggingface.co
[6] https://huggingface.co/cardiffnlp/twitter-roberta-base-sentiment-latest
[7] data.world/crowdflower/sentiment-analysis-in-text
[8] "hate" is mapped only to "anger" and not "disgust"

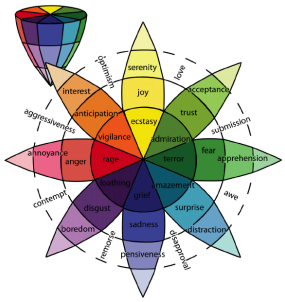


Figure 1: Wheel of emotions by Plutchik (2001)

3.3.2 Unsupervised Clusters

In order to compare the grouping based on emotion categories to a different method of partitioning the data, we also clustered the fastText (Bojanowski et al., 2016) embeddings of the training and test set using the (unsupervised) K-means clustering algorithm (Jin and Han, 2010) with the same number of $k = 7$ clusters. For an overview of text clustering with K-means, see section C.1 in the appendix.

3.3.3 Resulting partitions

Our hypothesis was that both methods might find useful partitions of the data. However, as is illustrated by fig. 2, the emotion classes do not seem to have any specific correlation to their position in the space of fastText embeddings (projected to two dimensions). This is emphasized by the fact that all the clusters found in this way have exactly the same number of positive and negative examples, indicating that there is no connection between emotion class and polarity (see table 5).

On the other hand, the unsupervised clustering did cluster the data in a recognizable (if apparently arbitrary) way, as can be seen in table 4. In fact, there are clear differences in polarity distribution, as seen in table 4.

4 Results

Our main results are shown in Figure 3, which compares our best performing BERT models with the three baseline models: Naive Bayes, XGBoost with GloVe, and logistic regression with fastText. As expected, BERT has better performance when trained on the whole training dataset (shown in orange), as opposed to a small subset (shown in blue). The highest accuracy on the test set was achieved by BERT models trained for a single epoch on all 2.5M labelled samples of the dataset. The training and test accuracies of the baselines and the best BERT models are summarized in table 1.

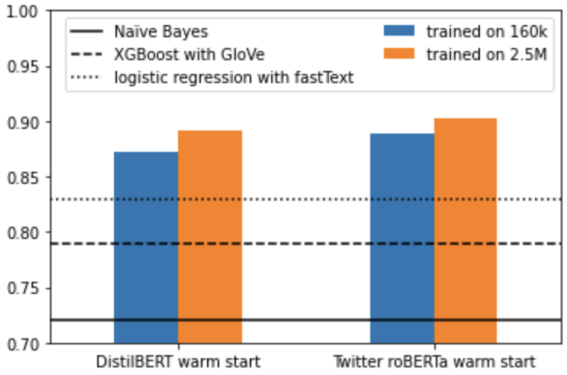


Figure 3: Test accuracy of BERT models trained on a subset and all data compared against the baselines.

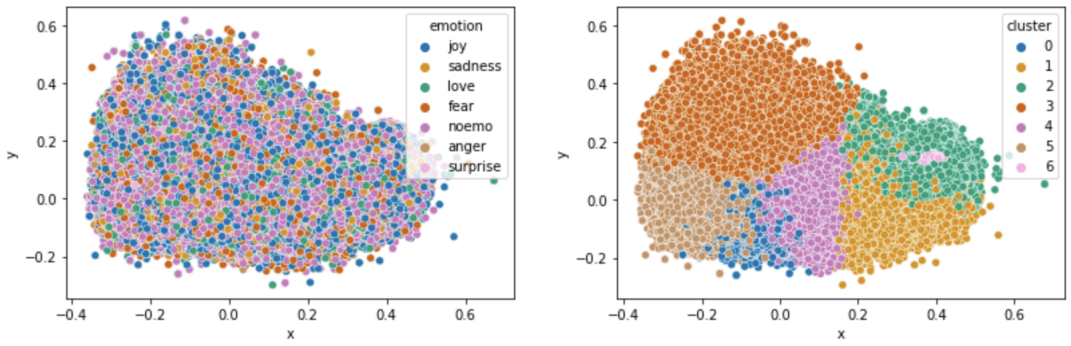
Model	Training Acc.	Test Acc.
Naive Bayes	0.753	0.729
XGBoost	0.805	0.786
Logistic Regression	0.833	0.827
DistilBERT	–	0.891
Twitter roBERTa	–	**0.903**

Table 1: Performance of the baseline models vs our best BERT model (with warm start)

Table 3 shows in more detail the performance of BERT models trained on 160k training samples on 40k validation samples. The training accuracy of the Twitter roBERTa Warm Start model for each of the unsupervised and emotion clusters is shown in table 6 and table 7, respectively. Between the emotion clusters, the performance is roughly identical. In contrast, when looking at the clusters obtained by unsupervised clustering, clusters 1, 2, and 6 have (near-) perfect accuracy, whereas samples of cluster 5 are classified correctly with only an accuracy of 0.88. In an attempt to further increase overall performance while using little computational resources, we trained new BERT models separately on clusters 0, 3, 4, and 5. However, the performance on the test set did not improve significantly. The results are shown in table 8. For the worst-performing cluster (5), we also considered an XGBoost model with BERT embeddings and fastText embeddings, but the validation accuracy on roughly 300k samples when trained on the remaining 300k samples of cluster 5 was only 0.77 and 0.78, respectively.

5 Discussion

We generally observed that, on this corpus of tweets, models such as BERT, which take into account the



(a) Emotion Clusters

(b) Unsupervised Clusters

Figure 2: Visualization of clusters in the space spanned by the first two principal components of fastText embeddings

context of words, are far superior to methods that are based on averaging word embeddings.

5.1 Warm Start vs Fine-Tuning

In our experiments, we found that adjusting all model parameters of BERT (warm start) has superior performance when compared to only adjusting the final classification layer (cf. table 3 and table 8). Yet, potentially, fine-tuning may still lead to better performance when training for a much larger number of epochs. This hypothesis we could neither confirm nor reject.

To further improve the model, training all parameters for some more epochs and then switching to training the classification layer parameters only, or simply training the classification layer (but for more epochs), could both be a viable approach.

5.2 Clustering Approach

We found that clustering the tweets and then learning separate BERT models did not improve performance. This is likely because the BERT model is already very expressive. Therefore, making some latent variables (such as the clusters) explicit does not increase the expressiveness of the model. If useful, BERT would likely capture this clustering information implicitly. In fact, training on clusters separately for multiple epochs may increase the risk of overfitting.

Besides improving the accuracy, our main objective of the clustering-based approach was to reduce the required training time for a good model. The fundamental idea is to spend more time learning data distributions of the clusters – which the model did not learn well initially – and to not spend much time learning data distributions of clusters, which the model had already learnt "well-enough". We expect this approach to work under the following two necessary conditions:

1. when there is a significant difference between the data distributions of the used clusters; and
2. when the model has not yet adjusted to the data distributions of all clusters as well as the training data permits.

In particular, if (1) is not satisfied, then for learning the model of a cluster, samples from other clusters will be helpful. In this case, continuing to train on just the data belonging to one cluster is effectively the same as just reducing the number of training samples, which then leads to worse model performance.

On this corpus, it appears that clustering did not produce a sufficiently diverse set of clusters for this method to work (cf. fig. 2). Moreover, our results indicate that training the BERT models for multiple epochs does not yield a substantial boost in performance, and hence, (2) does also not appear to be satisfied. The presented approach may, however, still be useful when the corpus is too large for a BERT model to be trained on all samples with reasonable efficiency.

6 Conclusion

We trained multiple pre-trained BERT models on the task of sentiment classification and compared them against a set of baseline models (Naive Bayes, XGBoost, Logistic Regression), showing that transformer-based architectures significantly outperform models that are based on averaging word embeddings.

To improve the performance of our models, we experimented with fine-tuning specific parameters, using different pre-trained models, and – to improve training efficiency – dividing the data into clusters and retraining on those with the worst performance.

The best classification accuracy of 0.90 was achieved by Twitter roBERTa Warm Start, which improved upon the DistilBERT implementation by 1.3 percentage points, thereby confirming our hypothesis that pre-training on Twitter-specific data would be helpful.

References

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2016. Enriching word vectors with subword information.

Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.

Laura-Ana-Maria Bostan and Roman Klinger. 2018. An analysis of annotated corpora for emotion classification in text. In *Proceedings of the 27th International Conference on Computational Linguistics*, pages 2104–2119, Santa Fe, New Mexico, USA. Association for Computational Linguistics.

Tianqi Chen and Carlos Guestrin. 2016. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, page 785–794, New York, NY, USA. Association for Computing Machinery.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jianyu Huang. (beta) dynamic quantization on bert. https://pytorch.org/tutorials/intermediate/dynamic_quantization_bert_tutorial.html. Accessed: 2022-07-27.

Xin Jin and Jiawei Han. 2010. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2016. Bag of tricks for efficient text classification. *arXiv preprint arXiv:1607.01759*.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. 2017. 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*, pages 427–431. Association for Computational Linguistics.

D. Jurafsky and J.H. Martin. 2009. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Pearson international edition. Prentice Hall.

Kevin Lu, Aditya Grover, Pieter Abbeel, and Igor Mordatch. 2021. Pretrained transformers as universal computation engines.

Robert Plutchik. 2001. The nature of emotions: Human emotions have deep evolutionary roots, a fact that may explain their complexity and provide tools for clinical practice. *American Scientist*, 89(4):344–350.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems*, 30.

Vikramkumar, Vijaykumar B, and Trilochan. 2014. Bayes and naive bayes classifier.

A Preprocessing

In the following, we discuss several preprocessing techniques we implemented in order to improve the quality of the usually noisy twitter data, which often contain spelling mistakes, grammar errors, and neologisms. The tweets in our training and test data were already provided somewhat preprocessed: They are all lowercase, tokenised[9], and URLs and user tags were replaced by "<url>" and "<user>", respectively. In addition, we implemented and tested the following common preprocessing methods:

- Lemmatization and stopword removal using the nltk[10] library
- Spelling correction using SymSpell[11]
- Single symbol & tag removal.

While these methods improved performance on some of the baseline models - especially lemmatization and stopword removal helped counteract bias and unknown words in Naive Bayes - we found that they did not improve or actively worsened the performance of our best models, which is why they were not used for most of our later work. The only type of preprocessing with a small measurable positive effect was the removal of <user> and <url> tags (see table 2).

Preprocessing	Accuracy
None	0.869
Tag Removal	**0.875**
Stopword Removal	0.843
Lemmatization	0.843
Spellchecking	0.868

Table 2: Validation accuracy on 40k validation samples of DistilBERT Warm Start models (1 epoch, batch size 32) with preprocessing when trained on 160k samples

It was especially surprising to us that running spelling correction on the whole training and test data deteriorated performance since we would have expected it to improve the word embeddings significantly. However, it turned out that many "corrections" actually changed correctly spelled names into words, thereby altering the semantics of the sentence drastically, which might explain the effect.

[9]Meaning here that each word and each non character symbol is separated from the others through whitespaces.

[10]https://www.nltk.org

[11]https://github.com/wolfgarbe/symspell, specific implementation: https://www.kaggle.com/yk1598/symspell-spell-corrector

B Baseline Models

B.1 Naive Bayes

A commonly used baseline for sentiment classification is the Naive Bayes Classifier[12]. It uses a Bag of Words (BOW) approach meaning words are taken as independent from each other, which is a very strong assumption, but it works reasonably well and was deemed a good baseline due to its simplicity. It determines the class of a document (tweet) by calculating the following:

$$\hat{c} = \underset{c \in \{pos, neg\}}{\arg\max} \log P_c + \sum_{w \in \mathcal{D} \cup V} \log \hat{P}(w_i|c) \quad (1)$$

Where P_c is the prior (fraction of tweets in class c), and $\hat{P}(w_i|c)$ is the estimated likelihood of a word occurring in a tweet of class c with additive smoothing:

$$\hat{P}(w_i|c) = \frac{count(w_i, c) + 1}{\sum_{w \in W}(count(w, c) + 1)} \quad (2)$$

The tweets were preprocessed using lemmatization and removing tags and stopwords.

This approach gave us a baseline accuracy of 0.75 for the training set and 0.72 on the test set. Note that this score may be improved by using a sentiment lexicon and/or preprocessing words following a negation to include the negation (see Jurafsky and Martin (2009)), which we omitted to pursue more promising approaches.

B.2 XGBoost

Another approach for text classification is the XGBoost algorithm (Chen and Guestrin, 2016). It works by creating increasingly deep regression trees with continuous leaf scores as weak regressors/ classifiers to classify individual examples. The resulting output is a weighted average of all trees. The algorithm then uses gradient descent to derive the optimal leaf values and tree weights. XGBoost utilises data very efficiently and includes regularisation parameters, making it appropriate for another baseline. We used the Scikit-Learn wrapper XGBClassifier[13] based on the highly optimised sklearn implementation. As the embeddings for the model input, we used the provided GloVe[14] embeddings from the task with a dimension of 100. We additively combined the individual word embeddings to get an overall tweet embedding. However, one should note that due to the commutativity of addition, like Naive Bayes, this approach does not take sentence structure into account and thus is expected

[12]For a detailed discussion, see Jurafsky and Martin (2009)

[13]https://xgboost.readthedocs.io

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

to perform worse than Recurrent Nerual Net architectures or Transformers such as BERT (discussed in appendix B.4). The model was run with default parameters, a maximum tree depth of 6 and 1000 boosting rounds. This method gave us training and test accuracies of 0.81 and 0.79, respectively.

B.3 Logistic Regression

Predicting the (binary) sentiment can also be framed as a regression problem. Instead of learning a classifier for the sentiment directly, one can learn a probability distribution over sentiments. A simple example of this approach is to learn a linear model with weights $\mathbf{w}$ using the embedding $\mathbf{x}$ as a predictor and then transform the predictions $\mathbf{w}^\top \mathbf{x}$ to the probability of the corresponding tweet having positive sentiment using the logistic function,[15]

$$\sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + \exp(-\mathbf{w}^\top \mathbf{x})} \in (0, 1), \quad (3)$$

$$y \mid \mathbf{x}, \mathbf{w} \sim \text{Bernoulli}(\sigma(\mathbf{w}^\top \mathbf{x})). \quad (4)$$

Here, $y \in \{-1, 1\}$ corresponds to the tweet being classified as negative or positive, respectively. Thus, using the symmetry of the logistic function around 0, the probability of a correct classification of the embedding-sentiment pair $(\mathbf{x}, y)$ is given as $\sigma(y\mathbf{w}^\top \mathbf{x})$.

To obtain a vector representation of a tweet, we average the fastText embeddings of each word, which represent each word as an n-gram of characters and use subword embeddings as additional features to capture some partial information about the local word order (Bojanowski et al., 2017).

With these embeddings, we obtain the maximum likelihood estimate of the weights,

$$\begin{aligned} \hat{\mathbf{w}} &= \arg\max_{\mathbf{w}} \prod_{i=1}^{n} \mathbb{P}(y_i \mid \mathbf{x}_i, \mathbf{w}) \\ &= \arg\max_{\mathbf{w}} \sum_{i=1}^{n} \log \sigma(y_i \mathbf{w}^\top \mathbf{x}_i) \\ &= \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \log(1 + \exp(-y_i \mathbf{w}^\top \mathbf{x}_i)), \end{aligned} \quad (5)$$

where we assume that the samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ are independent.

Our implementation is using the Logistic Regression classifier provided by the fastText library (Joulin et al., 2016) and achieved an accuracy of 0.833 on the training set and 0.827 on the test set.

[15] Instead of the logistic function, we use the hierarchical softmax as an approximation.

B.4 BERT

BERT (Bidirectional Encoder Representations from Transformers) (Devlin et al., 2018) is a transformer-based language representational model that pre-trains representations from unlabeled text by jointly conditioning on left and right context in all layers. These representations are not task-specific, such that the model can be pre-trained and then fine-tuned on many different tasks, including sentiment classification. The core part of BERT makes use of the Transformer model (Vaswani et al., 2017), an attention based model that learns contextual relations between words in a text. A Transformer consists of an encoder that produces representations from the input and a decoder that outputs a prediction for a specific task. BERT only uses the encoder, because its goal is to produce a language model. BERT is bidirectional in the sense that it doesn't read an input sequence from one direction word by word, but reads the entire sequence at once, which allows it to learn contextual information of a word from all the words to both of its sides.

Figure 4 provides an overview of the BERT architecture. For more detailed information, refer to the original paper (Devlin et al., 2018).

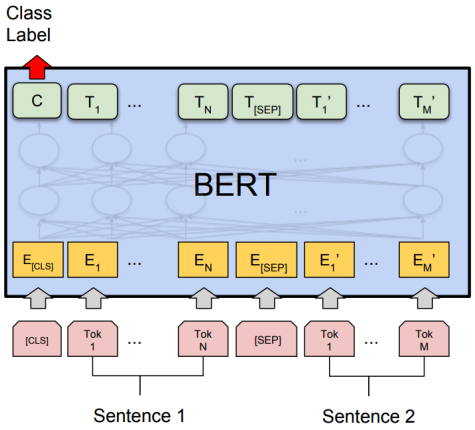


Figure 4: BERT architecture (Huang)

Often, pre-trained BERT models are used, since they take a lot of computational resources to train but also generalize well. Our warm start model trained for one epoch with batch size 16 starting from the DistilBERT pre-trained model achieves a validation accuracy of 0.87.

C Clustering

C.1 K-means

K-means (Jin and Han, 2010) is one of the simplest unsupervised learning techniques that solve the problem of clustering. It works by defining k centroids, one for each cluster, which are initially randomly placed.

Then, during the training phase of the algorithm, iterative calculations are performed to optimize the positions of the centroids:

1. By reducing the in/cluster sum of squares, each point in the dataset is taken and associated to the centroid closest to it.
2. The values of the centroids are recalculated as the mean of all the points belonging to that centroid – hence the name k-Means.

These steps are repeated until the centroids have been stabilized or training has surpassed a certain number of iteration steps.

Model	Accuracy	Precision	Recall	F1
DistilBERT Warm Start 1 epoch, batch size 16	0.871	0.863	0.884	0.873
DistilBERT Warm Start 1 epoch, batch size 32	0.869	0.879	0.856	0.867
DistilBERT Warm Start 1 epoch, batch size 64	0.874	0.862	0.890	0.876
DistilBERT Warm Start 2 epochs, batch size 32	0.873	0.885	0.858	0.871
DistilBERT Warm Start 3 epochs, batch size 32	0.872	0.870	0.874	0.872
Twitter roBERTa Warm Start 1 epoch, batch size 32	**0.896**	**0.891**	**0.903**	**0.897**
Twitter roBERTa Fine-tuned 5 epochs, batch size 32	0.850	0.846	0.857	0.851
Twitter roBERTa Fine-tuned (LN) 5 epochs, batch size 32	0.879	0.874	0.886	0.880

Table 3: Performance of BERT models trained on 160k samples on 40k validation samples

	0	1	2	3	4	5	6
% of samples	22%	9%	7%	9%	26%	26%	1%
% of positives	47%	9%	2%	82%	65%	54%	0%

Table 4: Analysis of unsupervised clusters

	noemo	joy	fear	sadness	anger	love	surprise
% of samples	38%	29%	16%	7%	7%	0%	3%
% of positives	50%	50%	50%	50%	50%	50%	51%

Table 5: Analysis of emotion clusters

	0	1	2	3	4	5	6
Accuracy	0.909	0.987	0.995	0.927	0.911	0.876	1
Precision	0.910	0.913	0.880	0.943	0.923	0.888	0.727
Recall	0.894	0.953	0.926	0.970	0.940	0.887	1
F1	0.902	0.933	0.903	0.956	0.932	0.888	0.842

Table 6: Training performance of Twitter roBERTa Warm Start after training on all 2.5M samples per (unsupervised) cluster

	noemo	joy	fear	sadness	anger	love	surprise
Accuracy	0.917	0.917	0.914	0.915	0.916	0.901	0.916
Precision	0.914	0.914	0.912	0.913	0.914	0.906	0.914
Recall	0.922	0.921	0.917	0.919	0.921	0.885	0.921
F1	0.918	0.917	0.914	0.916	0.917	0.895	0.917

Table 7: Training performance of Twitter roBERTa Warm Start after training on all 2.5M samples per emotion cluster

	0	3	4	5
base: Twitter roBERTa Base fitted: Warm Start; 1 epoch	0.9008	**0.9032**	**0.9020**	0.9026
base: Twitter roBERTa Warm Start fitted: Warm Start; 1 epoch	**0.9026**	0.9020	**0.9020**	**0.9032**
base: Twitter roBERTa Warm Start fitted: Fine-tuned; 1 epoch	–	–	–	0.9020

Table 8: Test accuracy (across all clusters) of BERT models that use Twitter roBERTa Warm Start (1 epoch, batch size 32) for all test samples of other clusters and *fitted* initialized with *base* for all test samples of the respective cluster