Assignment 4: Emotion Classification using LLMs

261250317 Léo Valette
261091403 Kayvan Dharsee

Abstract

In this assignment, we used the GoEmotions dataset [Demszky et al. (2020)] to perform emotion classification. The dataset is made up of a set of messages from the Reddit social network annotated with 27 different emotions (plus neutral). We experimented with different types of models to compare their performance on this task. We found that simple models such as Random Forest or XGBoost performed particularly well when coupled with good data pre-processing, outperforming even Naive Bayes. We also tried out different pre-trained language models, an encoder-only model, BERT, and a decoder-only model, GPT-2. Although theory suggests encoder-only models are superior for this task, we did not observe this. Our results show slightly better results from GPT-2 when compared to BERT, despite the fact that it uses a decoder-only architecture. We were also able to experiment with the effect of fine-tuning classifiers for large pre-trained language models, versus the effect of fine-tuning the entire large language model.

Introduction

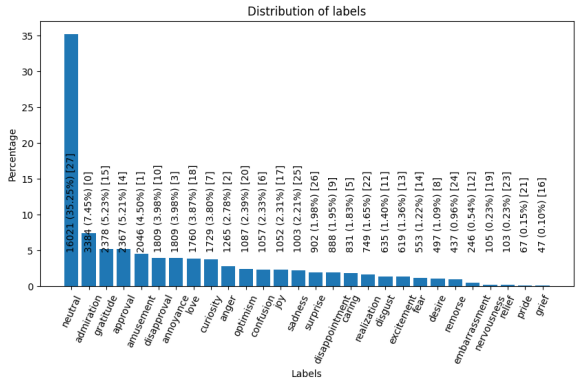
Emotion classification in text is a highly complex task requiring a thorough understanding of language. It is also very useful and has gained in popularity in recent years due to its multiple applications. These applications range from the creation of empathic agents to the analysis of customer sentiment. Research in the field has benefited from the creation of large annotated datasets like GoEmotions, which provides over 58,000 textual data points annotated by 82 human raters. This dataset, with its panel of 27 emotions plus neutral, is a very useful resource for evaluating emotion classification models.

Traditional approaches require complex feature engineering, and for a long time, methods using embedding representations such as Word2Vec [Mikolov et al. (2013)] were state of the art. In present day, transformer architecture-based models capabilities have overcome this dominance. Large pre-trained language models have been shown to be superior in many NLP tasks, and emotion classification is no exception [Gao et al. (2019)]. In this assignment, we experimented with a number of models based on different architectures in order to compare their performance. We tried different text representation methods such as bag of words or TF-IDF on different models such as Random Forest or XGBoost. We also implemented Naive Bayes and fine-tuned 2 LPLMs based on different architectures.

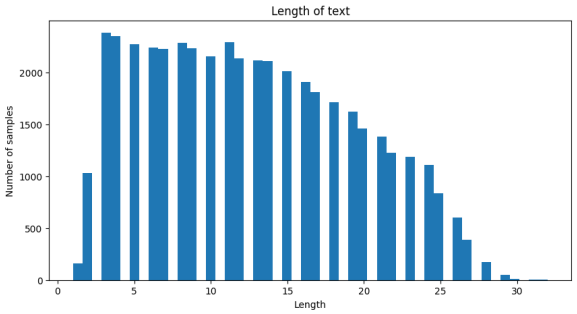
The results were surprising at first sight. Our Naive Bayes model was outperformed by both Random Forest and XGBoost, even when the latter two are in their default configuration. We also showed that training the classifier layer of an LPLM alone was not enough to achieve truly satisfactory results, and that only fine-tuning the entire model could approach and surpass the baseline models.

Methods

To preprocess the data, we removed all data points containing more than one label, and converted the labels from lists to integers. This removed roughly 16% of the data. We then mapped the integer classes to their respective emotional categories, and plotted the class distribution which showed clear class imbalance. We found that the neutral class had around 16000 data points, making it the most represented class. On the other hand, grief was the least represented class with under 50 data points. We also plotted and computed metrics for the length of the texts. The mean was 12.6 words, with a standard deviation of 6.7 words. The longest text had 32 words, while the shortest had 1. We used a bag-of-words vectorizer initially to vectorize the text data for our Naive Bayes model, and also experimented with adding a TF-IDF transformer to the vectorized data. We used the TF-IDF transformer vectors to train our baseline models, and utilized the LLM's respective tokenizers to transform the text data for the finetuning.



(a) Class distribution



(b) Length of text distribution

Figure 1

We implemented Multinomial Naive Bayes as our vectorizer gave us word frequency data. We implemented the class using basic PyTorch matrix operation functions to allow us the option to run our model on a GPU. We included an alpha hyperparameter for additive smoothing, which proved to be useful as many of the words appeared an infrequent amount of times compared to the total amount of words, causing all of our theta parameters to underflow to 0. We used the log-sum-exp trick and worked in the log domain to improve numerical stability. We also included a method to compute the accuracy, precision, recall, and f1-score of our model. We included the latter three as we had large class imbalance, and thus accuracy was not the best indicator of the model's overall performance. We used the unweighted average of the precision, recall, and f1-score across every class to get our model scores. Lastly we included a method to plot the confusion matrix, allowing for visualization of which classes our model predicted well, and which ones it struggled to predict accurately.

Initially we chose BERT as our LLM of choice as it was an encoder-only transformer. encoder-only transformers tend to perform better on tasks involving a deep understanding of the input text, such as sentiment analysis [Nielsen et al. (2024)]. BERT's bidirectional processing allows the model to collect more contextual information of the input sequence than a model with unidirectional processing such as GPT2. For the finetuning process, we froze all of BERT's parameters except the classifier layer parameters. We used data loaders to train the classifier with mini-batch SGD, and used BERT's tokenizer to convert the text data into tokens. We used the Adam optimizer, and Cross-Entropy Loss, and kept track of the model with the lowest validation loss, which we loaded as our trained model to avoid overfitting. We also fine tuned GPT2 using a similar process.

For our best Naive Bayes model, our Random Forest model, and our best XGBoost model, we used the TF-IDF transformed and downsampled data. For the Naive Bayes model we also used additive smoothing. For the Random Forest model, we used the default sci-kit learn hyper-parameters. For the XGBoost model, we used various hyper-parameters such as lambda and alpha for L2 and L1 regularization respectively, and the maximum depth of the tree. For our LLMs, the best models came from tuning the entire model for the sentiment analysis task.

Results

Starting with our baseline models, we implemented both Random Forests and XGBoost. In addition, we also performed hyper-parameter tuning for XGBoost to achieve optimal performance. As seen in Table 1, in terms of f1-score, the tuned XGBoost model had the best performance and is thus what we will refer to as our baseline.

For the Naive Bayes model, we first trained our model using the bag-of-words vectorized data. We then performed hyper-parameter tuning to maximize the f1-score of our model. Using Optuna, we found that the optimal bag-of-words vectorizer ignored words that appeared in more than 86% of the documents or in less than 9 of the documents, and considered both anagrams and bigrams. We recorded the metrics of this model, and produced its confusion matrix to quantify its performance. We observed that underrepresented classes such as grief are not predicted accurately, while more represented classes such as gratitude are.

We utilized Optuna for the tuning, as it uses a Bayesian Optimization algorithm which is both more accurate and more efficient than traditional grid search. In non-rigorous terms, "Optuna uses a history record of trials to determine which hyperparameter values to try next. Using this data, it estimates a promising area and tries values

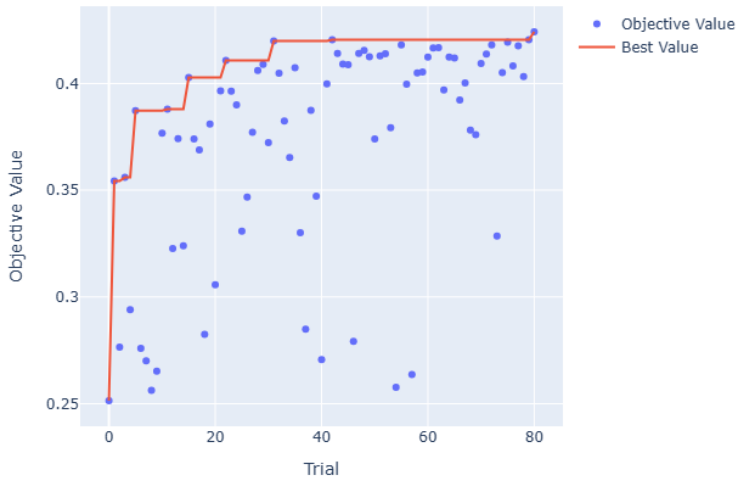


Figure 2: Optimization plot with Optuna - F1 objective

Category	Model Name	Test Acc.	Test F1-Score	Model Settings
Naive Bayes	Count Vectorizer	41.0%	34.0%	Downsample Target: 5457, Alpha: 2.83
	TF-IDF Vectorizer	**43.2%**	**35.5%**	Downsample Target: 10837, Alpha: 1.73
BERT	BERT No Fine-tuning	2.0%	0.0%	
	BERT With Fine-tuning	44.0%	16.0%	Mini-Batch SGD, Adam Optimizer, CE Loss
	BERT Downsampled	43.0%	13.0%	Downsample Target: 5000
	BERT Larger Classifier	43.0%	13.0%	Layers: (768, 512) → (512, 256) → (256, 28)
	BERT All Fine-tuned	**52.4%**	**45.3%**	No freezing of the BERT model
GPT2	GPT2 No Fine-tuning	2.0%	1.0%	
	GPT2 With Fine-tuning	45.0%	21.0%	Mini-Batch SGD, Adam Optimizer, CE Loss
	GPT2 Downsampled	43.0%	22.0%	Downsample Target: 5000
	GPT2 Larger Classifier	44.0%	24.0%	Layers: (768, 512) → (512, 256) → (256, 28)
	GPT2 All Fine-tuned	**56.5%**	**49.1%**	No freezing of the GPT-2 model
Baseline	Random Forest (Default)	54.2%	38.6%	Nb estimator : 100, Min sample split : 2
	XGBoost (Default)	**56.8%**	39.2%	ETA:0.3, Gamma:0, Max depth:6 Lambda:1, Alpha:0
	XGBoost (Optimized)	56.6%	**42.3%**	ETA:0.09, Gamma:0.45, Max depth:8 Lambda:0.53, Alpha:0.51

Table 1: Performance comparison of Naive Bayes, BERT, GPT2, and Baseline Models on Test set

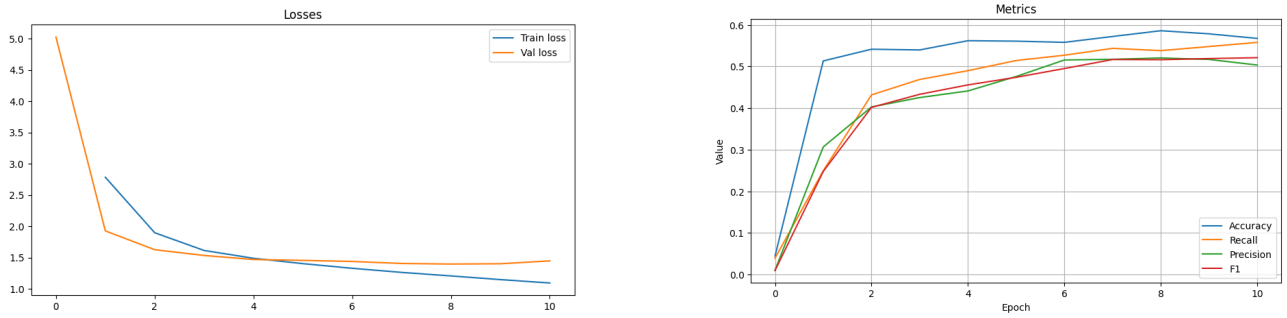
in that area" [*Optuna: An Automatic Hyperparameter Optimization Framework — Open Data Science Conference* (n.d.)].

We then applied a TF-IDF transformer to the vectorized data. All of the optimal parameters found can be viewed in our Naive Bayes notebook. Regarding the hyper-parameters exclusive to the transformer, we had IDF weighting, document frequency smoothing, and log-scaling of term frequencies all applied. We also recorded the metrics of this model and produced its confusion matrix, showing the TF-IDF vectorized data allowed for slightly better performance than the bag-of-words vectorized data.

We also calculated and plotted the hyper-parameter importance for these two models. A key observation we

noticed was that when optimizing f1-score, both models had the minimum document frequency cut-off as the most important hyper-parameter by a large margin. However, when optimizing accuracy, the downsampling target and alpha proved to be the most important hyper-parameters. We utilized this information to observe that when maximizing the f1-score, we focused more on creating a smaller vocabulary of approximately 3000 words compared to 11000 when maximizing accuracy. When maximizing accuracy, we focused more on having less imbalance, with a downsample target of around 5000, versus the approximate 11000 target when maximizing f1-score.

When comparing our best Naive Bayes model to the baseline model, we see that the Naive Bayes model scores lower in both accuracy and f1-score. Our assumption as to why Naive Bayes may be underperforming is that we do not fully have conditionally independent features given the label. Although the dependance may be low, it still does exist. Since this is a key assumption of the Naive Bayes algorithm, the fact that it is not entirely true likely has a negative effect on the model's performance.

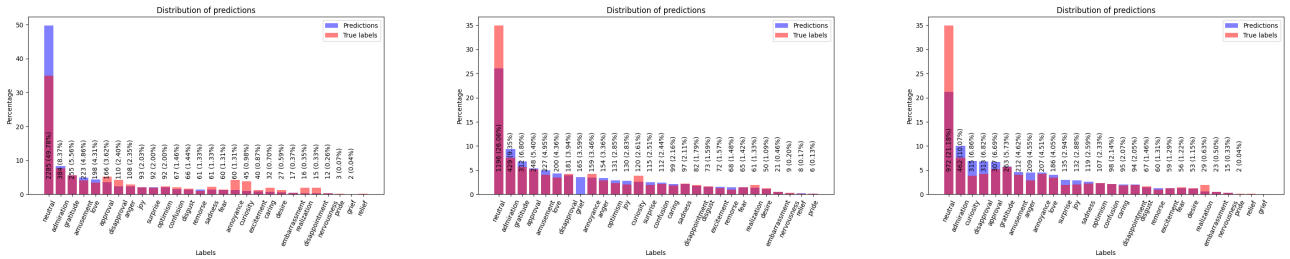


(a) Training and validation loss

(b) Metrics on the validation set during training

Figure 3: Training loss on the train and validation splits and metrics on the validation split for the GPT2 model

For the LLMs, we fine-tuned both BERT and GPT2. Originally we began with freezing all parameters except those in the classification layer. Without finetuning, both models performed slightly worse than chance level. With fine-tuning the classifier, we saw GPT2 have a slight edge over BERT in both accuracy and f1-score. However, both of these models still were well below the baseline. We finetuned the model with a downsampled dataset and a larger classifier, however the performance did not vary much with these changes. We then unfroze the frozen parameters and fine-tuned both models entirely, resulting in a large performance increase in both models. We also observed that the distribution of the predictions for GPT2 was closer to the class distribution of the dataset than the baseline model, and the confusion matrix for GPT2 maximized the diagonal values to a greater extent when compared with the baseline model and the best Naive Bayes model. The distribution of predictions is a good indicator of if your model is biased to a particular class, however it is not a good indicator of model performance, as the model can still be making these predictions incorrectly. The fact that the Naive Bayes model fits the class distribution better than the baseline model is a clear example of this, as the Naive Bayes model's performance is much worse than the baseline model. GPT2 still maintained the edge over BERT, however both models had passed the baseline model in terms of f1-score, and were close to the baseline model's accuracy score. GPT2 performing better than BERT in sentiment analysis was a shock to us, as we expected the encoder-only model to outperform the decoder-only model as the task involved a deep understanding of the context of the input text.



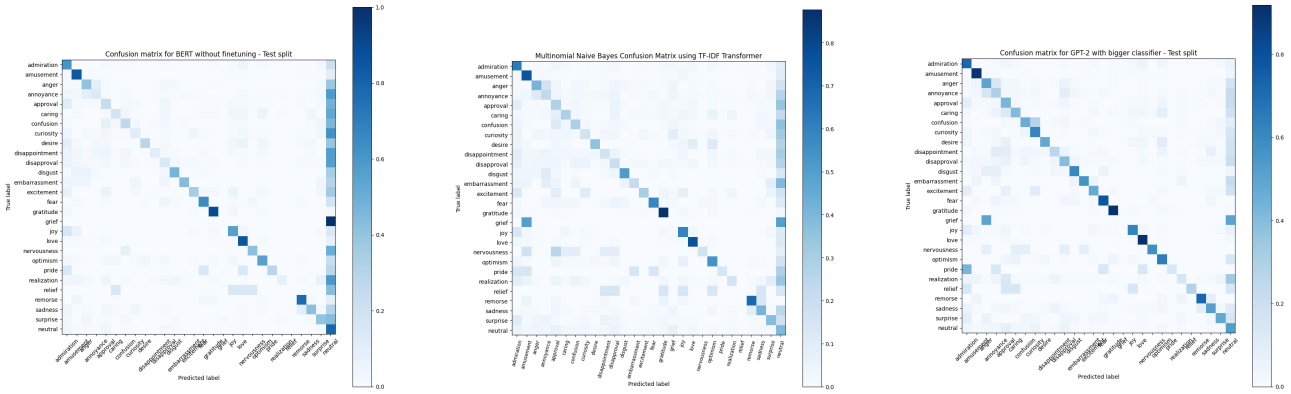
(a) XGBoost optimized

(b) Naive Bayes model optimized

(c) GPT-2 fine-tuned

Figure 4: Distribution of the predictions for the best base model, best Naive Bayes model and best LLM.

When examining the attention matrix between words and the class tokens for some of the correctly and incorrectly predicted documents, we can analyse how the attention heads of our model affect our prediction. We analyse the heads of the final layer before the classifier, as the class token here directly affects what class we predict. We can observe the improvement from BERT to the fully fine-tuned BERT model through the different attention heads.



(a) XGBoost optimized (b) Naive Bayes model optimized (c) GPT-2 fine-tuned

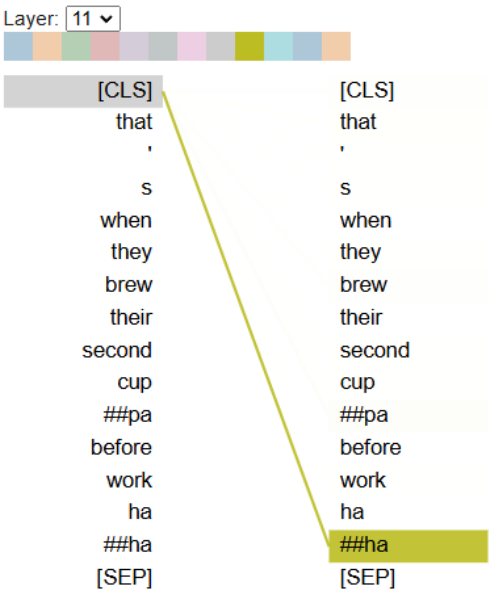
Figure 5: Confusion matrices for the best base model, best Naive Bayes model and best LLM.

In the pre-trained BERT model, we see that all the attention heads do not pay attention to any words specifically. However with the fully fine-tuned model, we see the different heads tend to pay attention to the same words ("when", "their", "before", "haha"). This could also be a sign of slight overfitting though, as typically different heads should observe different patterns. We also see on the fully-fine tuned model that one head pays attention to the token "##ha", which is the second token of the word "haha". With all other heads paying attention to the first token of this word, it is clear how our model correctly classifies the text "that's when they brew their second cuppa before work haha" in the "amusement" class.

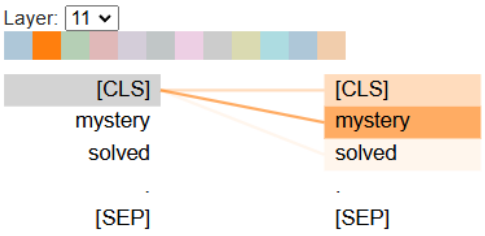
(a) Pre-trained BERT (b) Fine-tuned BERT

Figure 6: Visualization of the attention of all the heads of the last layer of the base BERT model and the fine-tuned BERT

Looking at the matrix of an incorrectly classified text, we can also observe why our model made the incorrect classification. We see in Figure 7(b) that one of the heads pays attention to the word "mystery" heavily, leading it to classify the text under the class "curious". However this text was classified as part of the "joy" class. We believe that the lack of exclamation mark to end the sentence makes the text "mystery solved." a difficult one to classify correctly, as even us as humans do not find it obvious that this text should belong to the class "joy".



(a) Correctly classified example



(b) Incorrectly classified example

Figure 7: One head of the final layer of the fully fine-tuned BERT model for 2 different examples.

Discussion and Conclusion

Although the GoEmotions dataset is very useful, it can be used unethically. For example, as the text data all comes from Reddit, one may be able to use this dataset to train hard to detect Reddit bots that can spread misinformation on the platform. In addition, it must also be understood that this dataset is not a good representation of the general public as all the data comes from the demographic of Reddit users. Using a model trained on this dataset generally can lead to bad performance, even if the model is extremely accurate on the test data.

Pretrained LLMs help with this task, as the main goal of LLMs is to understand human language. This goal is very general, and is useful for our task as understanding human language is necessary for accurate text sentiment analysis. Fine-tuning the LLM allows for quicker and less computationally intensive training as the model already has an understanding of human language, while if we tried to train the model from scratch we would have to develop this understanding through the training.

Overall, the project provided a great insight on how various models perform on a sentiment analysis task, and highlighted the upsides and downsides of these models. Starting with the two baseline models, we observed the great power that Random Forest and XGBoost had out of the box. With Naive Bayes, we observed how the vectorizer used and additive smoothing affected our model's performance. We also concluded that the fact that we do not fully have conditionally independent features is likely part of the reason our model cannot reach the baseline model's performance. Lastly with the LLMs, we observed the large differences in model performance between fine-tuning only the classifier and fine-tuning the entire model. These were evidently shown when we examined the attention heads for correctly and incorrectly classified texts.

If compute power was not an issue, we would have also experimented with other open-source models such as Mistral and Llama. As we experimented with an encoder-only model (BERT), and a decoder-only model (GPT2), it would also have been interesting to experiment with an encoder-decoder transformer such as T5 or BART. It also would have been interesting to implement our own LLM and see how it stacks up against the baseline models.

Statement of Contributions

Léo and Kayvan combined their code and reports to create a final report and code.

References

Demszky, D., Movshovitz-Attias, D., Ko, J., Cowen, A., Nemade, G., & Ravi, S. (2020). *Goemotions: A dataset of fine-grained emotions.* Retrieved from https://arxiv.org/abs/2005.00547

Gao, Z., Feng, A., Song, X., & Wu, X. (2019). Target-dependent sentiment classification with bert. *IEEE Access*, *7*, 154290-154299. doi: 10.1109/ACCESS.2019.2946594

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). *Efficient estimation of word representations in vector space.* Retrieved from https://arxiv.org/abs/1301.3781

Nielsen, D. S., Enevoldsen, K., & Schneider-Kamp, P. (2024). *Encoder vs decoder: Comparative analysis of encoder and decoder language models on multilingual nlu tasks.* Retrieved from https://arxiv.org/abs/2406.13469

Optuna: An automatic hyperparameter optimization framework — open data science conference. (n.d.). https://odsc.com/blog/optuna-an-automatic-hyperparameter-optimization-framework/#:~:text=Optuna%20uses%20a%20history%20record,based%20on%20the%20new%20result. ((Accessed on 11/18/2024))

Appendix

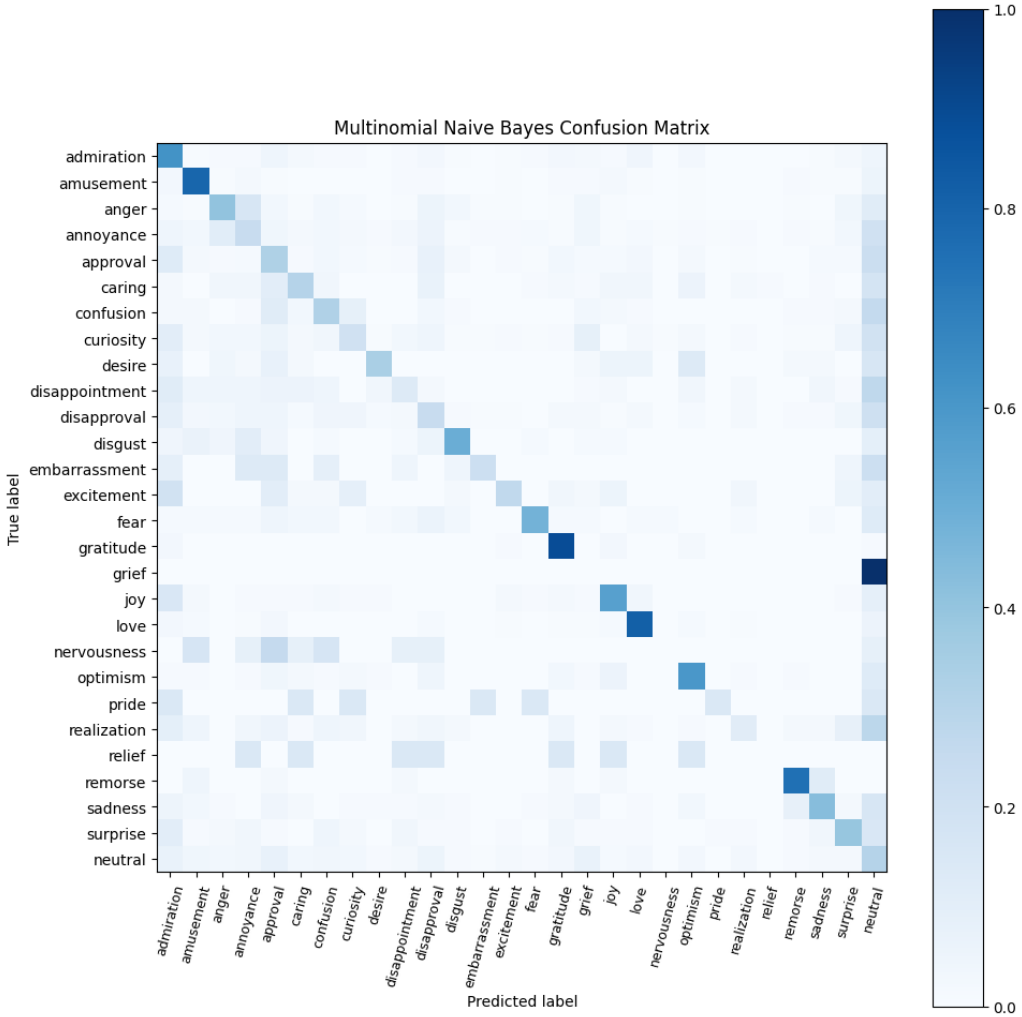
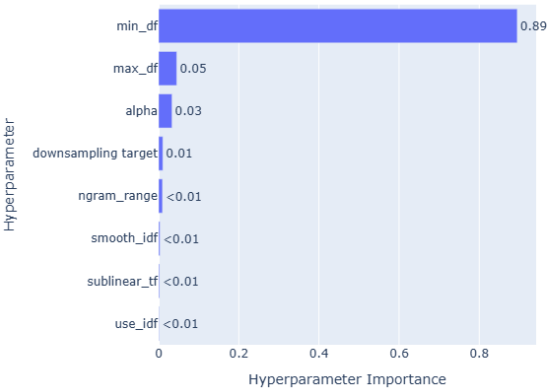
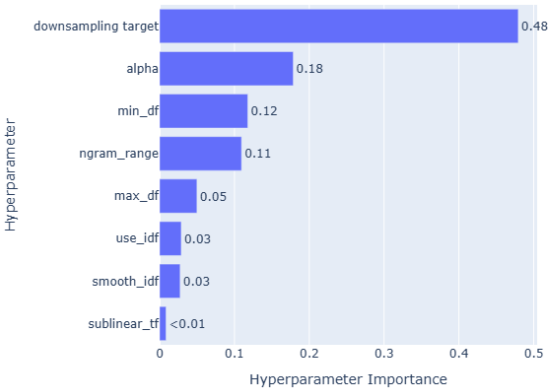


Figure 8: Confusion matrix of the Naive Bayes model with Bag-Of-Words representation



(a) Hyper-parameter importance when optimizing F1 score



(b) Hyper-parameter importance when optimizing accuracy

Figure 9: Comparison of hyper-parameter importance when optimizing Naive Bayes for TF-IDF transformer