Text2Emoji Classifier

Yauheni Paturyla, Daniil Belopolskikh

Oktober 2024

Abstract

This project presents a text-to-emoji classifier designed for the Russian language, aimed at mapping textual input to the most contextually appropriate emojis. The model leverages the BERT architecture, fine-tuned for emoji prediction with a custom classification head. A focus on handling class imbalance and optimizing model performance was achieved through the application of techniques such as LoRA for fine-tuning and focal loss adjustments. The classifier can be employed in social media platforms, messaging apps, or other user interaction environments where emoji-based sentiment expression is common. Link to the project: `https://github.com/megnar/Text2EmojiRus/tree/main`.

1 Introduction

Emojis play an important role in enhancing digital communication. But existing text emoji classifiers are mainly focused on English. This leaves a gap for Russian-speaking users. This project addresses the need for classifying Russian text emoji using a fine-tuned BERT model designed specifically for the Russian language. In contrast to simpler approaches, Our method uses deeper context and prevents class imbalance with techniques such as LoRA tuning and focus loss.

1.1 Team

Yauheni Paturyla collected dataset, cleaned dataset and bringed it to the desired form, experimented with training the model.
Daniil Belopolskikh experimented with training the model, prepared final illustrations and wrote a report.

2 Related Work

Several approaches have been proposed to solve the problem of text-to-emoji classification. Early methods often relied on simple rule-based systems or keyword matching. For instance, predictive text algorithms were designed to sug-

gest emojis based on specific keywords, which had limitations in capturing context and sentiment nuances [pre, 2018]. More advanced approaches utilized word embeddings such as Word2Vec and GloVe to map words to vector spaces and compute similarity scores for emoji suggestions [glo, 2023], but these methods struggled with complex sentence structures and lacked deep contextual understanding.

Another way was to use such methods like CNN [Kuldeep Vayadande, 2023] or LSTM-like models [Sounava Pal, 2023] which performs better than usual methods like GloVe. With the advent of transformers-like models it has become stadart to use transformers to make embeddings for sentences. Models like BERT or more advanced alternatives of bert. Another part of our research is creating classifier for a large number of classes. And it obviously cause different problems. To resolve problems like zeroing classes and unbalanced data in multi-class classification can be used such approaches like Focal loss [Foc, 2018] and weighted CrossEntropy [Cro, 2023]. Another problem was out of domain data for BERT and trained BERT for another task, so we trained model using different approaches like LoRa [Lor, 2023].

3 Model Description

Our approach to solve the russian text-to-emoji classification problem is based on a fine-tuned BERT model with a classification head. The base model is `DeepPavlov/rubert-base-cased-sentence`, a pre-trained Russian-language BERT. This model was chosen for its ability to capture semantic relationships in Russian text, which is necessary for understanding the context in which emojis are used.

3.1 Architecture

The architecture is following:

- **BERT Encoder:** The input text is tokenized and passed through the BERT encoder, which outputs word embeddings. The output [CLS] token representation is fed into a several fully connected layers. These layers predicts the probability distribution over the 62 emoji classes.

The model architecture can be described as follows. Let $\mathbf{x} = [x_1, x_2, \ldots, x_n]$ tokenized input text sequence. The BERT model encodes this sequence to produce embeddings $\mathbf{H} = \text{BERT}(\mathbf{x})$ with size 768, where $\mathbf{H} = [h_1, h_2, \ldots, h_n]$ and h_i is the embedding of token x_i. The embedding of the [CLS] token, h_{CLS}, is passed through a fully connected network:

$$\mathbf{z} = \text{ReLU}(W_1 h_{\text{CLS}}), \tag{1}$$

Finally, a softmax layer is applied to obtain the probability distribution over the emoji classes:

$$\hat{y} = \text{Softmax}(W_2\mathbf{z}), \tag{2}$$

where $\hat{y}$ is the predicted probability of the emoji labels.

3.2 Training Strategy

To resolve class imbalance(for some emoji in final dataset we get more than 40k items and for some we get less than 2k) we use focal loss. Focal loss formula:

$$\text{FL}(p_t) = -\alpha_t(1 - p_t)^\gamma \log(p_t), \tag{3}$$

γ is a tunable parameter that adjusts the focus on hard examples.

Also we applied LoRA (Low-Rank Adaptation) fine-tuning to optimize the training process. LoRA makes low-rank updates to the weight matrices in the transformation layers, reducing the number of trainable parameters. This approach allows for increased training efficiency and adaptation of the model to the dataset specific to the task and our domain.

3.3 Evaluation Metrics

By resason of huge number of classes and simillaruty between some emojis(for example you can use two simillar crying emojis to the same sentence, because they meaning the same emotion and user can use them randomly) we decided to reduce total number of emoji classes to 62 most frequently used classes. But the same problem remains. Although we reduced num of classes and cleaned out dataset, number of classes still remained large and simillar to each other. Thats why we used two accuracy metrics.

- Top1 accuracy metric for counting percent of correctly predicted for all classes.
- And Top10 accuracy metric that measures as ratio of emojis which made it into the top 10 predicted emojis and total number of emojis.
- Due to disballance of data, there was another metric for the largest class in dataset(in some experements all emojis were predicted as the largest class and it was another metric for understanding how model copes with this problem). It counts correctly predicted, number of elements of this class in dataset and number of times were this class was predicted.

4 Dataset

Search of free available dataset ended in failure and it was decided to collect the dataset yourself. First of all we decided to use tools for pooling data from telegram groups using telegram lib in python, using telegram api tokens. But

collecting dataset in this way was problematic because it was available only a small amount of data after that your token freezed. Another way was to simply download comments to telegram channels manually. Lack of this approach is that we collect data from small limited number of channels with its context and "special" words and it's viewers with peculiar comments. It can make our dataset slightly different from regular telegram messages.

You can see the statistics for the mentioned dataset . 4. First of all we collected raw data. After that we deleted all comments without emojis. After that we preprocessed data. We take raw comment, take all words before emoji, and truncated if it was too long($>$ than 15 words), and deleted if it was too shoort($<=$ 2 words). In some cases there were several emojis in a raw, and we created 2 different elements in a dataset with same texts and different emojis. After that we cleaned from duplicates, because there are a lot of spam in comments and it was important to clear dataset from it. After big number of experements we decided to reduce number of classes and after that we get final dataset.

	Train	Valid	Test
Raw data with emojis(sentences)	784274	0	0
Cleaned from duplicate 1 text - 1 emoji(sentences)	582956	0	0
Final dataset(sentences)	401850	44650	20864
Num of emojis we trained		62	
Total number of emojis in raw dataset		>1100	

Table 1: Statistics of the our dataset. We decided to filter all emojis that appears in dataset less than 1.5k times.

All data you can find here: `https://github.com/megnar/Text2EmojiRus`.

5 Experiments

All about evaluate metrics we described in detail in section 3.3. We started our experiments from rubert-base-cased model `https://huggingface.co/DeepPavlov/rubert-base-cased` with crossentropy loss and 2 linear layers. Main problem was zeroing of some classes and small total accuracy of model.

5.1 Lora adapter and wormup

The reason for the low accuracy could be the incorrect setting of the bert for our task. And we decided to train bert for our task. We didn't have access to powerful GPU servers, so we used LoRA adapter for our purposes. Adapter strongly increases accuracy metrics for our model. Adding wormup showed no gain or loss in the model.

	model without LoRA	model with LoRA
TOP1, %	13.84	16.26
TOP10, %	54.55	57.10

Table 2: Training model with LoRA, measurements after 1 epoch. Model with crossentropy loss

5.2 Focal loss

To resolve problem of zeroing of classes we decided to use focall loss. There were a lot of experiments, what parameters to use. Almost immediatly we understood that we don't need to change γ parameter and we focused on finding optimal α. We called metric of the ratio between correctly predicted elements to all elements in validation set for the biggest class in dataset as **Recall**. And metric of the ratio between correctly predicted elements to all predicted to this class elements as **Precision**.

	$\alpha = \frac{1}{\sqrt{fr(c_i)}}$	$\alpha = 1$	$\alpha = \frac{1}{fr(c_i)^{3/4}}$
TOP1, %	18.18	20.35	14.86
TOP10, %	67.71	69.25	62.61
TOP class Recall, %	40.99	60.17	25.37
TOP class Precision, %	26.45	24.34	28.71

Table 3: Focal loss with different parameters. $fc(c_i)$ is frequency of i class after normalization.

After this measurements we decided to use $\alpha = \frac{1}{\sqrt{fr(c_i)}}$.

5.3 Other embedding models

We decided to use other embedding extractors to find the best model for embeddings.

- Another bert-like model $ru - en - RoSBERTa$ `https://huggingface.co/ai-forever/ru-en-RoSBERTa`
- Another bert-like model $sbert - large - nlu - ru$ `https://huggingface.co/ai-forever/sbert_large_nlu_ru`
- Extraction embeddings from word2vec and averaging it for all sentence.

	$ru-en-RoSBERTa$	$sbert-large-nlu-ru$	$rubert$	$word2vec$
TOP1, %	20.35	20.27	17.12	16.34
TOP10, %	69.25	68.26	62.13	59.55
TOP class Recall, %	60.17	68.50	84.35	83.39
TOP class Precision, %	24.34	22.10	17.56	16.97

Table 4: Cross entropy loss with different embedding strategies

5.4 Lr, lora-alpha, lora-rank tune-up

We've tried different learning rates, lora parameters, scheduling, wormup. So, for final training we started training our model(ru-en-RoSBERTa) with lr = 2e-4, lora-alpha = 32, lora-rank = 32. After that 2-3 epochs we trained with lr = 4e-5.

6 Results

Final metrics for our task could be found in Tab. 5.

	$final$
TOP1, %	19.67
TOP10, %	69.61
TOP class Recall, %	51.82
TOP class Precision, %	26.70

Table 5: Final result with focal loss, parameters that we took above with ru-en-RoSBERTa model

After training it was interesting to understand how emojis distributed and how are they simillar to each other. So we take final layer(shape [512 x 62]) and used it like embeddings for all emojis. For each class we get vector size 512. After that we decided to compress (using algorithms like PCA) this vector to dimension 2 and place on a plane. Result on image 1

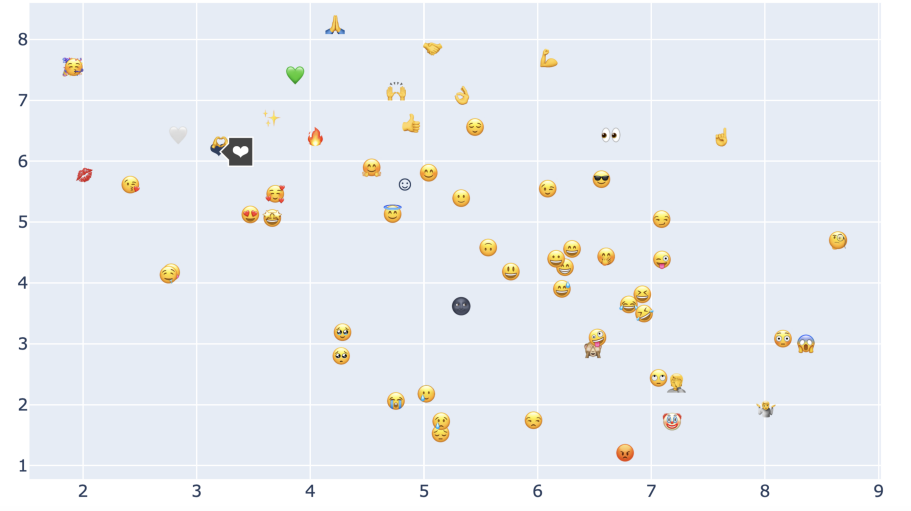

Figure 1: Distribution of emojis. More similar emojis are closer than others

As we see, many emojis created clusters and split into groups. For example, when we will try to clasterize this data, we will get following results on image 2

['😒', '😔', '😢']
['🙌', '👍', '😌', '👌', '💪']
['😋', '🤤']
['😳', '🤦', '🙄', '🤡', '😱', '😡']
['❤', '🙏', '\U0001faf6', '🤍', '💋']
['🥺', '🥲', '\U0001f979', '😭']
['🤭', '🧐']
['😇', '🙃', '👀', '🕳']
['🥳', '😽']
['😂', '😁', '😆', '🤣', '🤪']

Figure 2: Emoji clusterizing

Some of emojis doesn't display due to font. As we see crying emojis are in group with crying. And laughing also in correct group. That allows us observe, that model correctly understands mood and context of classes.
And finally we present our model on some random phrases. Image 6

```
Text: Здравствуйте, как ваши дела? #### Predicted class: 😊
Text: Послушай песню которую я сам сочинил #### Predicted class: 😎
Text: Знания Сила #### Predicted class: 💪
Text: по выходным сплю дома целый день #### Predicted class: 😢
Text: Сегодня сдал экзамен #### Predicted class: 😎
Text: Сегодня завалил зачет #### Predicted class: 😭
Text: Я забыл сегодня поздравить Никиту #### Predicted class: 😭
Text: У меня очень хорошее настроение #### Predicted class: 🥰
Text: Давай выпьем пива вечером #### Predicted class: 😉
Text: Огненное настроение #### Predicted class: 🔥
Text: Ты крайне мила #### Predicted class: ❤
Text: Сегодня планирую работать всю ночь #### Predicted class: 😢
```

You can try your own phrases. Just follow readme `https://github.com/megnar/Text2EmojiRus` Here you can find data and model weights and pipeline of training.

7 Conclusion

We have done good results for our model. This task is well solved in English, but there are no models for the Russian language. This set us the task of finding and preparing a dataset. After that, understand what to train with and solve problems associated with specific data. We coped with the task and the final result more than satisfied us. There are some possibilities for how the project could be developed.

- Instead of using LoRA adapters we can train all BERT using our data. It can take up more time and resources but result can be better.
- As we see many emojis are simillar to each other and some emojis can be used instead of others. For example 2 different crying emojis are used in the same texts, but with different probabilities. It's profitable for model to use more frequent emoji to get less loss. That's why some classes get zero predictions. One way to solve this problem is clustering and union some emojis before training. And after training just return emoji with probability of this class in dataset.
- As we see in 5.3 section word2vec embeddings worse than bert-like embeddings, but they also can bring some information to our model. So we can concatenate word2vec with bert embeddings and train model on this embeddings.

References

[pre, 2018] (2018). Find text to trigger a particular emoji in predictive keyboard. https://apple.stackexchange.com/questions/316237/find-text-to-trigger-a-particular-emoji-in-predictive-keyboard.

[Foc, 2018] (2018). Focal loss for dense object detection. https://paperswithcode.com/method/focal-loss.

[glo, 2023] (2023). Emoji text classification. https://github.com/Kiana-Jahanshid/Emoji-Text-Classification.

[Lor, 2023] (2023). Low-rank adapter (lora). https://medium.com/@shelikohan/low-rank-adapter-lora-explained-0d3677395639.

[Cro, 2023] (2023). Use weighted loss function to solve imbalanced data classification problems. https://medium.com/@zergtant/use-weighted-loss-function-to-solve-imbalanced-data-classification-problems-749237f38b75.

[Kuldeep Vayadande, 2023] Kuldeep Vayadande, Ubed Shaikh, R. N. (2023). Mood detection and emoji classification using tokenization and convolutional neural network. *ICICCS*.

[Sounava Pal, 2023] Sounava Pal, S. M. (2023). Classification of texts with emojis to classify sentiments, moods, and emotions. *ICMC 2023*, pages 275–291.