Food Retail NPS Classification With Small Amount of Training Data and Many Classes

Sergey Zaitsev, Senior Data Scientist, BurgerKing Rus

May 2022

Abstract

This article provides a real-life example of solution to business task of user's reviews classification, given small amount of training data, with comparison of best practice approaches and use of LLM. Code can be found on github:

`https://github.com/RvsL/mipt_food_retail_review_classification`.

1 Introduction

Extracting pains and gains from Net Promoter Score (NPS) is crucial to any business, its brand, and decision it makes. NPS frequently comes in unstructured text format, so one should be able to quickly label small amount of NPS and using it, get a bigger picture from the whole data, in order to quickly adopt to customer's needs.

This research aims to take best-practice approaches of text classification given our constraints of small training sample, Russian language and partly mislabelled data, apply this approaches (combining with business domain knowledge) in order to find most suitable one, and compare to BERT approach with different architectures and data preprocessing techniques.

This is a practical research, which will lead us to the most appropriate approach, which will be further applied to other NPS problems in same domain.

2 Related Work

First of all, let's list the constraints of current task:

1 The NPS comments are in Russian, which has plenty of forms per word and non strict word declension rules

2 The business goal is to manually label couple of dozens comments per problem, and then get the bigger picture. Task setup different from training on a big enough dataset

3 Data is partly mislabelled, as user aims to list couple of problems in one comment, and data is labelled as multiclass (1 comment - 1 class)

Good baseline solution is classification using TF-IDF transformation of comments, described in [Chen et al., 2016]. Code example given in: [MOHAN, 2022]

This work deals with classification of English comments, with clear text, and therefore, no preprocessing, wich will lead to poor quality in our case. Other constraint is that this work is about binary classification, while our dataset has 15 class labels.

So what we will do is use this approach as a model-zero, as it is SOTA solution, and combine it with another work [Alper Kursat Uysal, 2014], which deals with classification efficiency boost due to data preprocessing (lemmatization, sinonyms groupping, etc.), and each new data transformation we'll analyze on subject of impact that it brings to model quality.

Data augmentation techniques reviewed in [Jason Wei, 2019], we need to analyze its increment to model quality.

Model finetuning for text classification is reviewed in [Jeremy Howard, 2018], so we'll take best practice from it, and rate their impact on model quality.

Solution of problem of Russian text classification with BERT taken from [Shitkov, 2021]

Example of applying BERT to multilabel classification reviewed in: [Valkov, 2021]

Example of adding custom layers on top of BERT reviewed in: [Sangani, 2022]

3 Model Description

Our assumption is that given our main constraint, small training sample, we can make use of transfer learning - take pretrained model and retrain it on our sample, to map pretrained model's knowledge and rules in our training data. Utilization of this approach will be ruBERT model with dropout and linear classification layer on top of it, as shown in the figure 1.

Cross entropy Loss function seem to be most suitable for this task:

$$L_{CE} = -\sum_{i=1}^{n} t_i log(p_i),$$

for n classes, where t_i is the truth label, and p_i is the Softmax probability for i_{th} class.

The path to our goal is divived to the following steps:

Step1: SOTA solution will be built using Tf-Idf text vectorization and Gradient Boosting on Decision Trees (GBDT)

Step2: Our further workaround would be data preparation with use of lemmatization, cleansing, synonimization and business domain knowledge text preprocessing.

Step3: When it's done, we'll figure out the best way of data preprocessing by feeding it to GBDT, looking for best performance.

Step4: Next step is combine input text for BERT-based approach from initial text and preprocessed text after SEP token, in order to add business knowledge to model

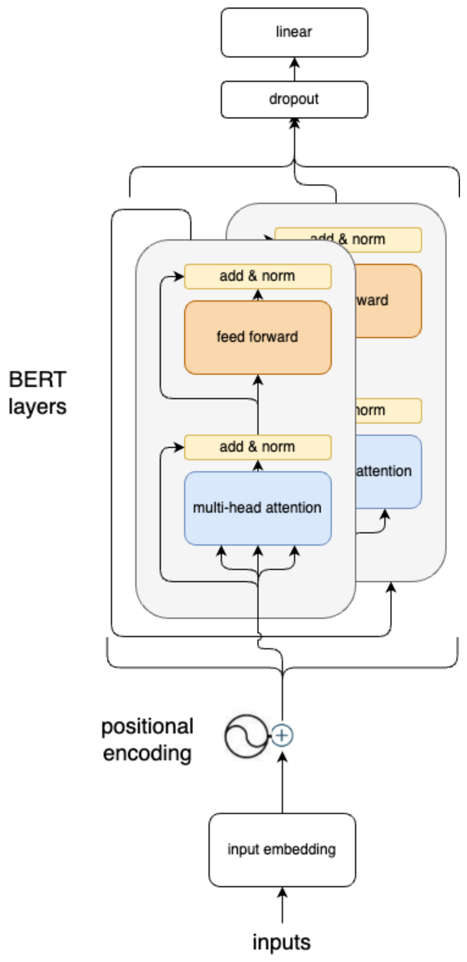

Figure 1: Torch model used for classification

Step5: Last step is BERT architecture fine-tuning, in order to get best from all approaches.

Step6: The above described steps seem to be enough for picking the most suitable approach. But we also need to act under constraint of partly mislabelled data, and the plan to cope with it is to train SOTA solution, add random insert to each comment, predict probabilities, and take probability of above empirical threshold as a proposal of second class presence.

Step7: We'll try to modify dataset to multi-label as described above, and check if such transformation of task statement boosts our final performance.

4 Dataset

Dataset includes manually labelled in multiclass style, negative comments of users, which installed mobile application. Those users have experienced issues with usage of application, or with other aspects of communication with the brand, and they hope the company will react and fix problems.

Dataset contains user comment and class name, which was obtained manually by sorting the comments for given timeframe. Some labels may describe about the same problem, and hardly distinguished even on human level performance.

Other issue is that user usually tries to cover several issues in one comment - if user has got two problems, he won't write two comments, he'll put it to one comment, so that it contains lots of text, and user hopes this will attract company's attention. But the target variable is multiclass, which means no matter how many problems really covered in comment, only one goes to target, which may mislead any classifier.

Dataset is included in github repo, mentioned in the abstract section of current document. This dataset consists from small portion of comments in AppStore and GooglePlay of BurgerKing app for small timeframe, and anyone can see them, so this dataset is avaliable for research purposes.

If we artificially divide dataset into 80/20 train/test, we get the parameters shown in table 1.

Comment count per class label given in table 2.

	Train	Test
Comments	2,536	635
Tokens	55,321	14,592
Unique tokens	8,863	1,302
Vocabulary size	10,165	
Out of Vocab rate	14.7%	

Table 1: Statistics of the dataset. The out of vocabulary (OoV) rate notes what percentage of tokens have been replaced by an $\langle unk \rangle$ token. The token count includes newlines which add to the structure of the dataset.

5 Experiments

5.1 Metrics

Knowing from table 2 that dataset is imballanced, we need to use F1 metric, as it accounts for imballanced multi-class targets.

$$F1 = 2 * Precision * Recall/(Precision + Recall)$$

$$Precision = TP/(TP + FP), Recall = TP/(TP + FN)$$

Table 2: Comment count per class in the dataset.

Class name	Comments count
глюки баги тормоза	300
доставка общее	300
лояльность	300
другое	293
оплата	250
uxui	246
создание заказа	223
обслуживание	215
купоны	183
аккаунт	168
регистрация/коды	157
не возвращаются деньги отмененного заказа	156
обновление	151
долгое ожидание доставки	120
цена	116
Total	3,171

$$F1_{macro} = 1/n * \sum_{i=1}^{n} F1_i,$$

n is number of classes

5.2 Step1: SOTA solution using Tf-Idf text vectorization and GBDT

We start with building the best SOTA solution to refer to its quality in future. We'll do it by building catboost implementation of GBDT with hyperparameters grid search on data, class weight balancing, preprocessed according to flow, described in steps 2-3.

According to this approach , the best quality we can get using all the pipeline of data preparation is 66% F1 score, see classification report in table 3. This will be our baseline, we'll try to beat with LLMs.

5.3 Step2: data preparation

Following [Alper Kursat Uysal, 2014] we'll clean data in several steps.

1. Obscene words cleaning. Obscene lexicon frequently appear in comments and obviously don't carry any information, because our classification don't touch emotions - we already know these comments concern dissatisfaction. So we can clear obscene words. This is done via regular expressions, but one issue with regex and obscene lexicon, is that users know that particular comment

Table 3: GBDT classification report on preprocessed data

	precision	recall	f1-score	support
uxui	0.56	0.34	0.42	44
лояльность	0.78	0.87	0.82	46
не возвращаются деньги отм заказа	0.71	0.86	0.78	29
обновление	0.62	0.80	0.70	25
регистрация/коды	0.85	0.71	0.77	24
другое	0.53	0.68	0.60	47
доставка общее	0.74	0.58	0.65	45
цена	0.75	0.75	0.75	20
купоны	0.71	0.88	0.79	17
аккаунт	0.87	0.83	0.85	24
долгое ожидание доставки	0.60	0.62	0.61	24
обслуживание	0.56	0.50	0.53	28
глюки баги тормоза	0.56	0.50	0.53	48
оплата	0.66	0.68	0.67	28
создание заказа	0.39	0.43	0.41	28
accuracy			0.65	477
macro avg	0.66	0.67	0.66	477
weighted avg	0.65	0.65	0.64	477

may be just deleted only because it contains obscene word. So users slightly modify obscene words in a random manner, so regex can no longer recognize it. Our approach to clean obscene lexicon is to use inexact search via fuzzywuzzy implementation, to clear every word if it is like one of obscene.

2. Emoji also carry emotional component, so we can easily get rid of them. There is one issue with emoji - sometimes a meaningful word is replaced with emoji. But this word almost always has great document frequency, so, once again, can easily be removed.

3. Business domain replacements. E.G. when user mentions loyality program issue, he faces an issue with particular personal offer (coupon), which have various naming, and can not be strictly associated with coupons, or loyality, or any other particular word. Approach used is to build a dictionary of coupons, use inexact search to find word clusters that are similar to name of any coupon, and replace this cluster with a particular word "coupon".

4. Lemmatization is very promising when it comes to Russian language, we use Yandex implementation of lemmatizer. One issue with lemmatization, it uses most common word in language (not in our corpus) to replace word in multivariate case. E.G. word "плачу" which in our case stands for "платить"("pay"), yandex lemmatizer refers as "плакать"("cry"), so knowing this, we need another translation dicrionary to replace incorrectly recognized words with words

we think they might mean.

5. Synonyms. Even after we lemmatized words, we still will have synonyms which there is no need to distinguish as they carry same information. In our code we again use inexact search to group them (replace synonyms with most common one)

5.4 Step3: impact of data preparation

Impact of data preparation steps given in 4. We see that all the steps give boost to model performance. To our surprise, emoji cleansing don't give effect. It can be explained in the way that emoji most likely have great document frequency, thus don't influence on tf-idf based classifier.

We keep in mind that results mentioned above were obtained using Random Forest Classifier without hyperparameter tuning on all text manipulations except random insert. Comparisson of table 4 and 3, gives us that hyperparameter tunning and use of GBDT instead of RandomForest gives us 1.7 percent points raise of F1 score.

Adding random insert to GBDT don't contribute to model quality, but contributes to Random Forest solution. We make conclusion that random insert works like class balancing mechanism, because we constructed it this way, and GBDT used class balancing from scikit learn package, and both solutions have equal quality.

Table 4: Increment of data preprocessing steps, using Random Forest Classifier

Description	f1-score
(1) pure content	0.626 ± 0.01
(2) obscene cleaned	0.63 ± 0.01
(3) 2 + emoji cleaned	0.63 ± 0.01
(4) 3 + lemmatization	0.640 ± 0.01
(5) 4 + business domain replacements	0.64 ± 0.01
(6) 5 + synonims groupping	0.643 ± 0.008
(7) 6 + random insert	0.650 ± 0.005

5.5 Step4: build BERT-based approach

This experiment contains three stages with out-of-the-box BERT classifier:

1 classifier built on raw comments: F1 = 66%

2 classifier built on preprocessed text: F1 = 66%

3 classifier built on raw comments + <SEP> token + preprocessed text, in order to add business knowledge to model: F1 = 66%

Batch size varying gives effect: best quality gained with batch size = 8. E.G. for classifier built on raw data, F1 = 64% | batch size = 2, F1 = 66% | batch size = 8

Training on preprocessed data (compared to raw) gives boost from 57% to 62%. If we analyse F1 scores per class in order to find out which class had the best boost, we'll find out that all classes gain raise, except for UxUi, which falls from 47% to 36%, but at the same time class "service" ("обслуживание") gain raise from 21% to 40%.

As far as we see significant difference in model performance due to data we feed, let's make comparison of different texts and different checkpoints. Results presented in table 5

Learning history of best base BERT model given in figure 2

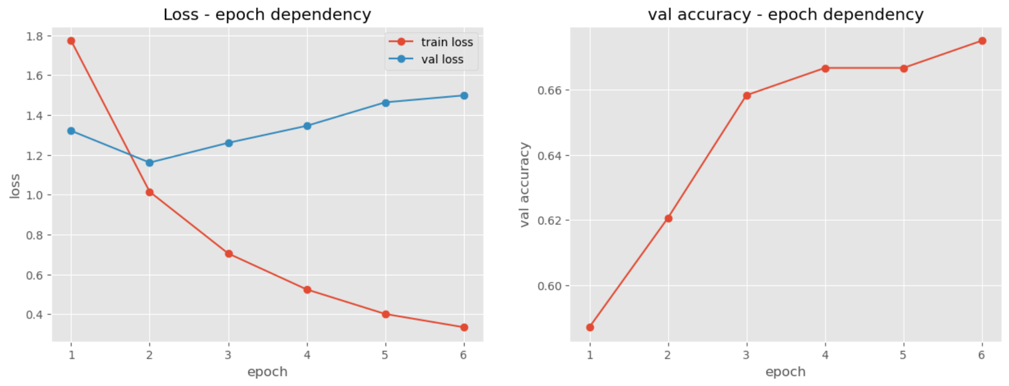

Figure 2: Learning history of best base BERT model

Table 5: BERT classification F1 macro, % on different combinations of checkpoint and text type

Checkpoint	raw text	preprocessed text
rubert tiny	64	64
rubert tiny 2	66	66
distilrubert conversational	66	66

5.6 Step5: BERT architecture fine-tunning

We assume that fine-tuning of architecture and hyperparameters will affect model quality in a positive way, so in this subsection we'll implement architecture described in figure 1, adding dropout layer on top of BERT, and searching for best batch size.

It really gives significant boost with batch size = 8, classification report given in table 7, F1 score = 68%, which is 2 percent points better than SOTA.

Interesting that for this architecture, the trick with building classifier on preprocessed data doesn't work: F1 score falls from 68% to 66%, which is still best of all listed results, but significantly worse than training on raw data.

Cross table of text type and checkpoint, analogically to table from step 5 given in table 6

Learning history of best custom BERT model given in figure 3

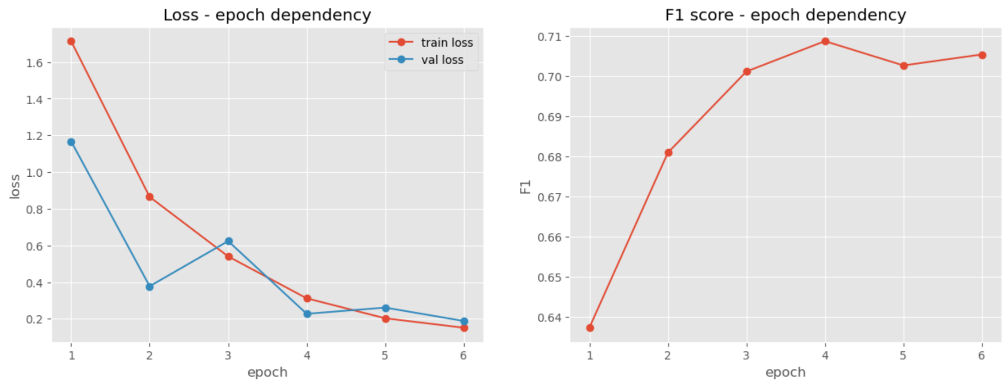

Figure 3: Learning history of best custom BERT model

Table 6: BERT classification F1 macro, % on different combinations of checkpoint and text type

Checkpoint	raw text	preprocessed text	raw + random insert
rubert tiny	60	64	
rubert tiny 2	65	66	
distilrubert conversational	68	66	67

5.7 Step6: convert dataset to multi-label

The above described steps seem to be enough for picking the most suitable approach. But we also need to act under constraint of partly mislabelled data, and the plan to cope with it is to train SOTA solution, add random insert to each comment, predict probabilities, and take probability of above empirical threshold as a proposal of second class presence.

5.8 Step7: use multi-labelled dataset to verify if it gives performance boost

We'll take the approach from previous steps with the best F1 metric and try to use it on dataset with target converted to multi-label in step 6, in order to find out if it boosts models' quality.

Table 7: Customized BERT classification report on initial data

	precision	recall	f1-score	support
uxui	0.57	0.71	0.63	34
лояльность	0.85	0.83	0.84	47
не возвращаются деньги отм заказа	0.37	0.50	0.42	14
обновление	0.70	0.64	0.67	25
регистрация/коды	0.84	0.81	0.82	26
другое	0.68	0.70	0.69	37
доставка общее	0.76	0.79	0.77	39
цена	0.60	0.75	0.67	12
купоны	0.89	0.75	0.81	32
аккаунт	0.67	0.80	0.73	20
долгое ожидание доставки	0.50	0.48	0.49	21
обслуживание	0.69	0.56	0.62	45
глюки баги тормоза	0.64	0.60	0.62	47
оплата	0.81	0.78	0.79	37
создание заказа	0.66	0.66	0.66	41
accuracy			0.70	477
macro avg	0.68	0.69	0.68	477
weighted avg	0.70	0.70	0.70	477

The way we're doing it is the following:

1 balance data with random inserts so that each class has equal count of examples

2 build Random Forest Classifier

3 add another random insert to all data, and predict probabilities using classifier from previous step.

4 for each row mark class label with max probability, EXCEPT TRUE LABEL, as a second class label in this example.

5 build target variable as a multi-label, with maximum two labels per example

6 feed it into customized BERT

7 construct custom metric, positional recall, denoted as:

$$Recall_{positionalMacro} = (1/N) * \sum_{i=1}^{N} Recall_{positional}$$

$$Recall_{positional} = (1/n) * \sum_{j=1}^{n} \begin{cases} +1 & \text{if } pred_j = true_j \\ -1 & \text{otherwise} \end{cases},$$

where n is number of classes (15), and N is number of test examples

The result of this experiment is negative because algorithm described above makex class balancing non straightforward. Our approach was to balance classes using random inserts, but the real experiment showed that after steps 4 and 5, dataset becomes more disbalanced than it was before. If before transformation to multi-label format, class counts varied from 116 to 300, then after step 5, it varied (in total occurences) from 121 to 520.

Macro F1 obtained by best model is 54%

Positional recall of best model is 91%

Positional recall of best model on multi-class data is 96%

Conclusion is that this approach increase false positives, and uncontrollably disbalance our metric, so this decreases our metric. But, as it was empirically verified, approach in step 1-5 is good for finding second problem in text.

6 Results

Main result is that customized architecture of BERT with tuned hyperparameters and correctly picked checkpoint show better F1 score than SOTA with all possible preprocessing and augmentation. Also this customized BERT is better than basic BERT for sequence classification with the same hyperparameters and checkpoint.

Brief results summarized in table 8.

Approach	F1 macro, %	F1 weighted, %
Tuned GBDT Tf-Idf with preprocessing	66	64
Basic BERT with best hyperparams	66	66
Custom BERT with best hyperparams	68	70

Table 8: Results

7 Conclusion

1 all preprocessing steps work fine for SOTA solution, raising F1 from 62% to 65%

2 random inserts prepare augmentation, which imply class balancing, and raise accuracy to the best SOTA F1 metric (66%)

3 GBDT with hyperparameter tuning on give same F1 as from previous point

4 BERT with basic configuration don't show increment compared to SOTA, not on raw data, nor on preprocessed data

5 BERT with basic configuration shows overfitting after two epoches, which is not convenient and not robust

6 attempt to convert task to multi-label fails, because generates uncontrollable imbalanced dataset, which confuses any model and brings too many false

positives. Though, this approach can be used to help identify hidden problems like a recommender system

7 custom configuration presented in figure 1 show robust fitting and best F1 scores. It will be used on more complex tasks in this domain.

References

[Alper Kursat Uysal, 2014] Alper Kursat Uysal, S. G. (2014). The impact of preprocessing on text classification. https://dokumen.tips/documents/the-impact-of-preprocessing-on-text-classification.html?page=1.

[Chen et al., 2016] Chen, K., Zhang, Z., Long, J., and Zhang, H. (2016). Turning from tf-idf to tf-igm for term weighting in text classification. https://www.sciencedirect.com/science/article/pii/S0957417416304870.

[Jason Wei, 2019] Jason Wei, K. Z. (2019). EDA: Easy Data Augmentation Techniques for Boosting Performance on Text Classification Tasks. https://arxiv.org/abs/1901.11196.

[Jeremy Howard, 2018] Jeremy Howard, S. R. (2018). Universal Language Model Fine-tuning for Text Classification. https://arxiv.org/abs/1801.06146.

[MOHAN, 2022] MOHAN, N. (2022). NLP - Text Classification using TF-IDF Features. https://www.kaggle.com/code/neerajmohan/nlp-text-classification-using-tf-idf-features.

[Sangani, 2022] Sangani, R. (2022). Adding Custom Layers on Top of a Hugging Face Model. https://towardsdatascience.com/adding-custom-layers-on-top-of-a-hugging-face-model-f1ccdfc257bd.

[Shitkov, 2021] Shitkov, K. (2021). BERT for Russian Text Classification. https://habr.com/ru/articles/567028/.

[Valkov, 2021] Valkov, V. (2021). Multi-label Text Classification with BERT and PyTorch Lightning. https://curiousily.com/posts/multi-label-text-classification-with-bert-and-pytorch-lightning/.