Sber interview task

Initial task formulation:

Основная задача новостного мониторинга - обрабатывать входящий поток новостей, находя в них интересные пользователям события. В задании предлагается построить модель для выявления в новости события, соответствующего задержке ввода некоторого объекта в эксплуатацию

Дано:

- Обучающая выборка (train_data.csv)

- Новостной поток за несколько дней (test_data.csv)

Задача:

Найти в большом потоке новости, в которых есть информация о событии.

Формат предоставления результатов:

- Файл с кодом и описанием алгоритма поиска релевантных новостей. Желательно сделать код полностью воспроизводимым.

- Файл test_data.csv с добавленным полем, содержащим вероятность принадлежности новости к положительному классу

Business task:

Find news that contains information about events that corresponds to a delay in putting some construction object into operation

Applied task:
Build a model that predicts a possibility of object operation delay by sentiment analysis. Basically a classification task.

Data objects:
Some text from csv. More TBD after EDA

Metrics:

Assuming - Is it ok to get lots of FP? Seems like recall is more important than prescicion. Baseline F_b with b <= 1. Maybe use model probability score to map confidence and tune threshold

Inference:
Nothing said

Idea:
* first look, EDA
look for entities:
    relevant news tag words
    class keywords
build a key-word map

* take pretrained RU sentiment analysis model
    BERT

* Transfer learning on dataset

* test model PoC

Let's go!

EDA Summary

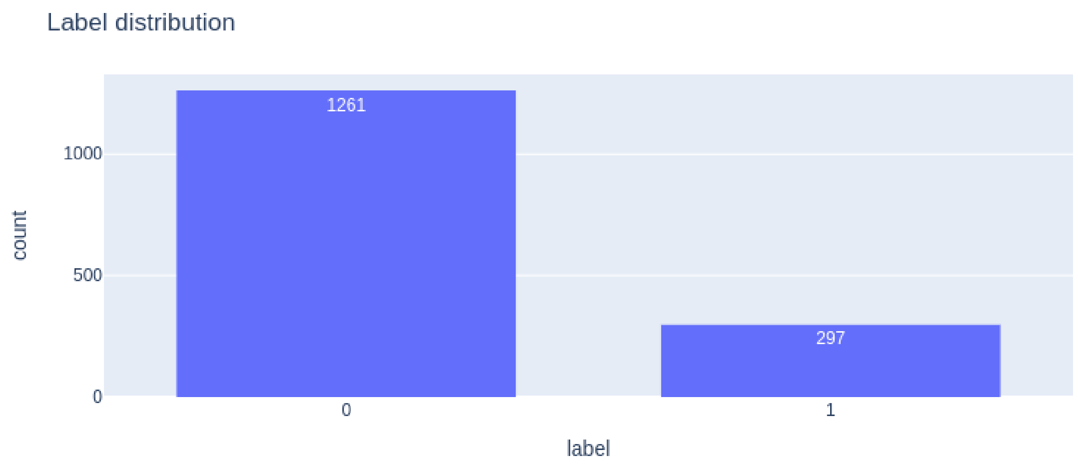
train.csv

Objects:
    'sentence' - input text, str
    'label' - class, int, 0 or 1

Size - ~1700 rows
Data fulfillness is good, no NAN
Duplicates - removed (duplicated around 100)

After preprocess ~ 1600 rows

labels disbalance is significant



test.csv

Objects:
* title
* text

baseline:
1 title + text = sentence

not marked up
size: ~ 10k rows
no duplicates
nan are present, but fixed easily with fillna, no totally empty rows

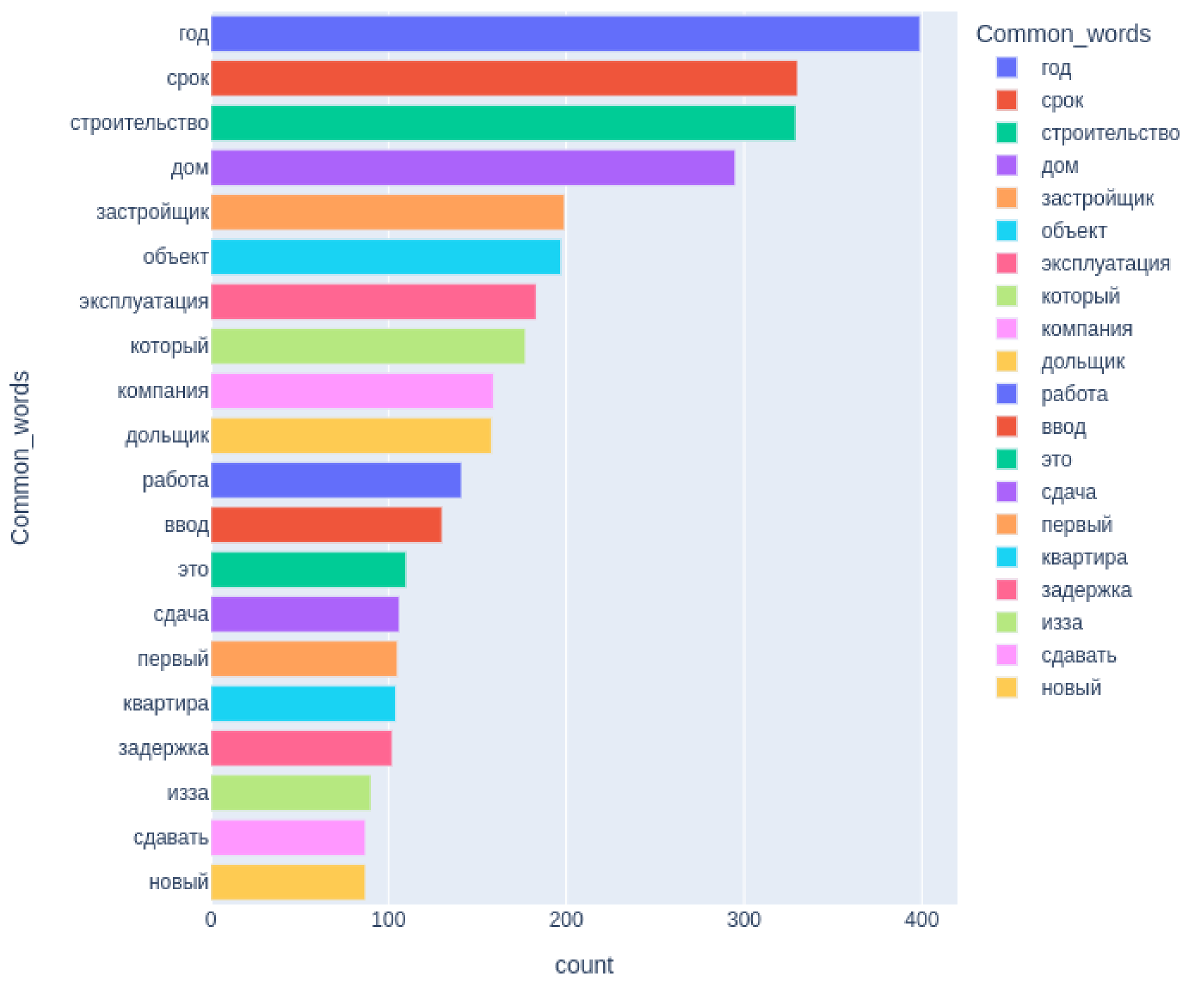
Words Analysis

Clean data with nltk stopwords and pymystem3.Mystem for lemmafication

Most common words in train set

0	год	399
1	срок	330
2	строительство	329
3	дом	295
4	застройщик	199

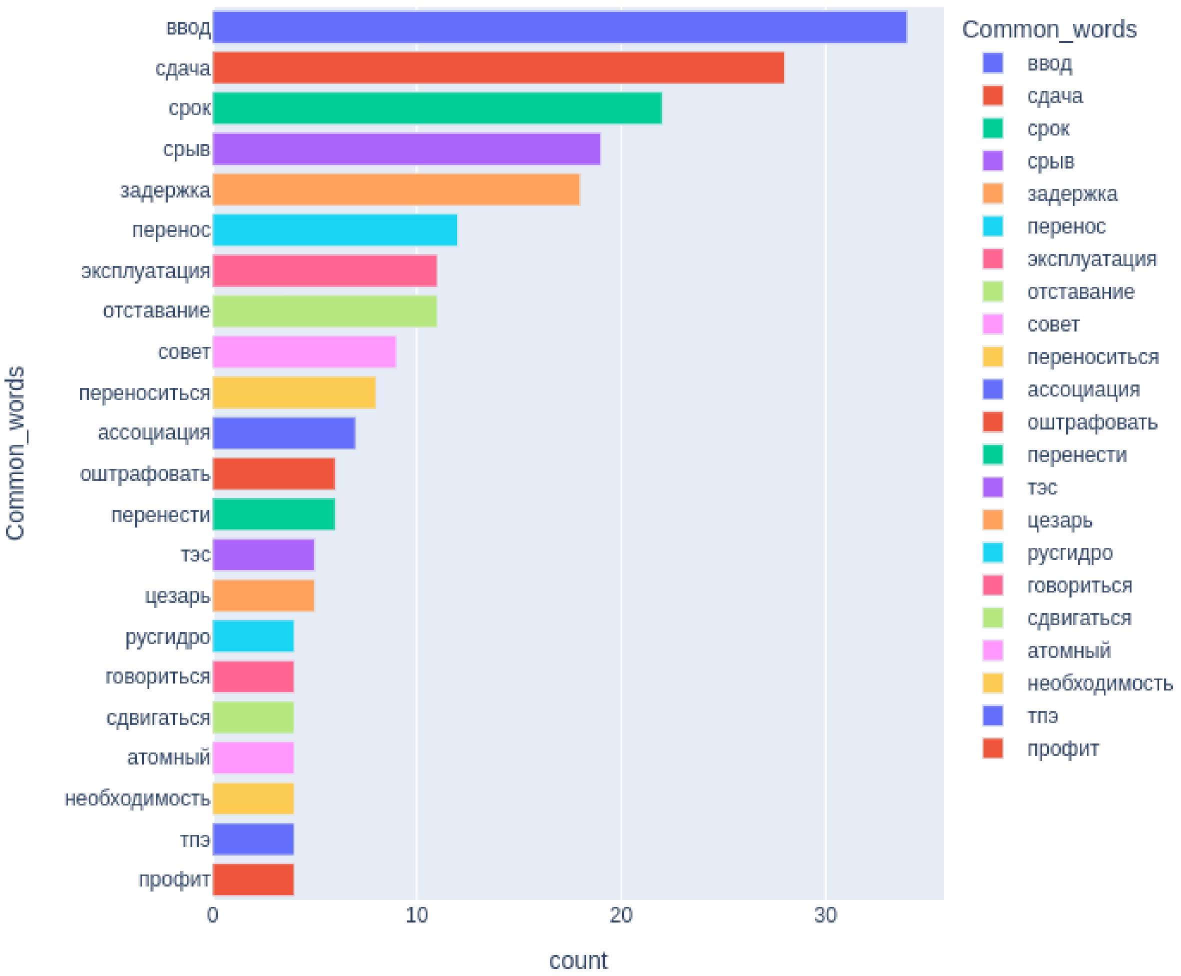
Commmon words in all sentences



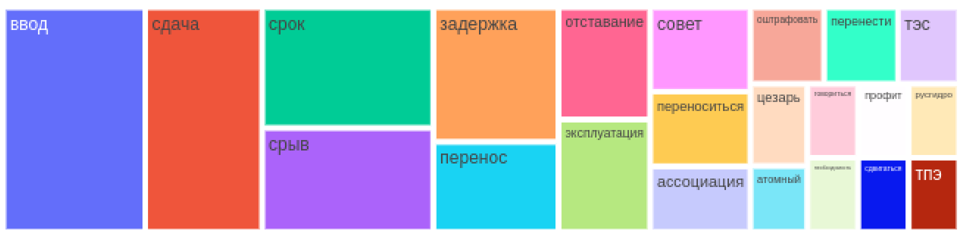
Not informative
Filter rows by label==1

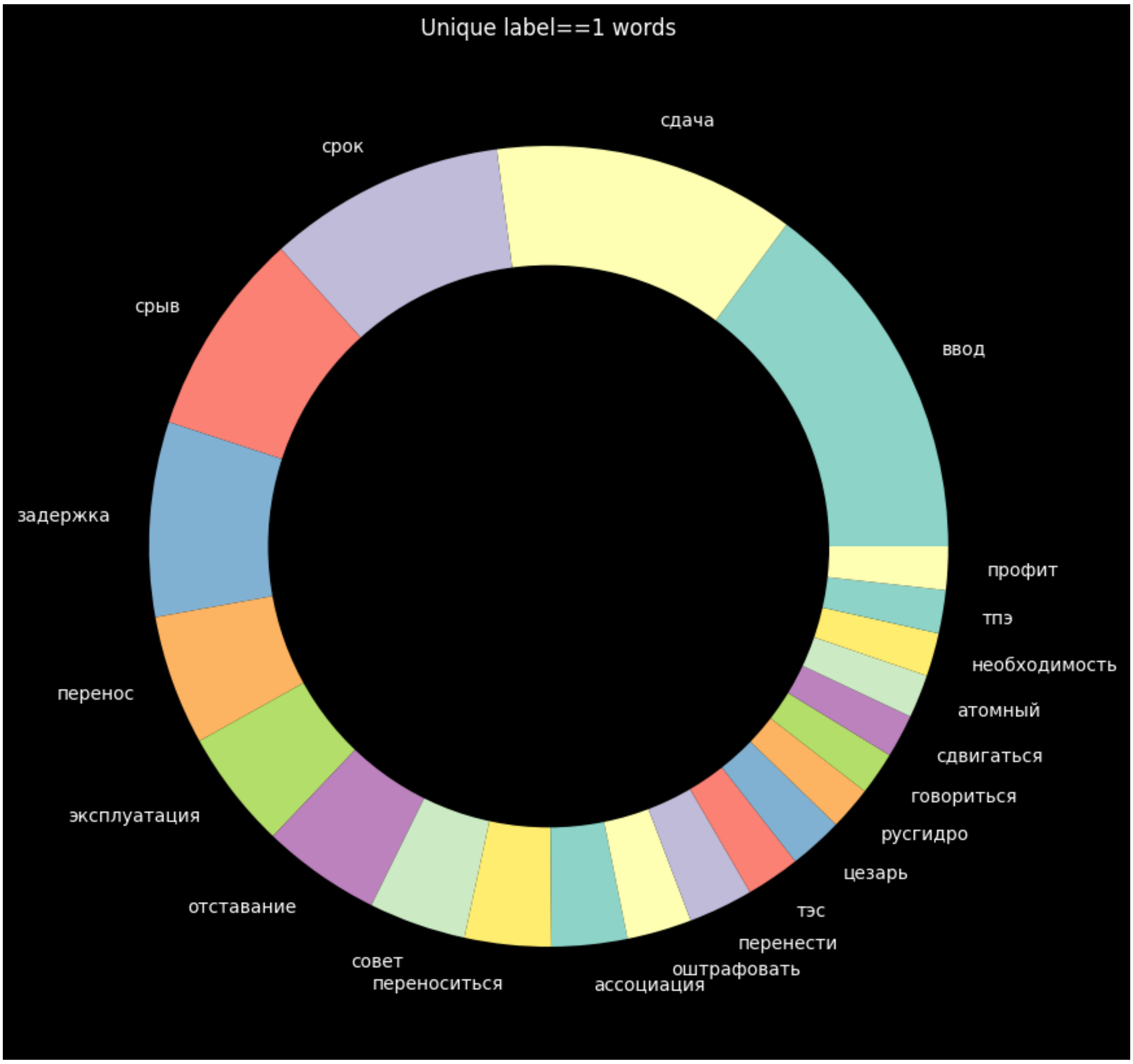
0	ввод	34
1	сдача	28
2	срок	22
3	срыв	19
4	задержка	18

Commmon unique words in label==1 sentences



Tree of commmon unique words in label==1 sentences





That's more informative

Seems like we have a unique set of words, which are used only in label==1 sentences

<u>Hypotesis</u>:
Naive classifier could use a list of tag words, which are the most common words in label==1

Class disbalance ideas

We have a class disbalance in the data
I have 2 ideas how to deal with it
* make val set without class disbalance (50/50 labels)
* use sklearn calculate class weights, but thats only applied for the model loss function

For the naive classifier I made val set with equal number of labels,

Naive classifier logic

0

- val set - 100 rows, labels (50/50)
- train set - ~1500 rows (all but val)
- no test set

1 Create a list of most common words for label==1 from **train**. This is baseline classifier tag words list

2 If a sentence contains any word from tag words - assume its label is "1"

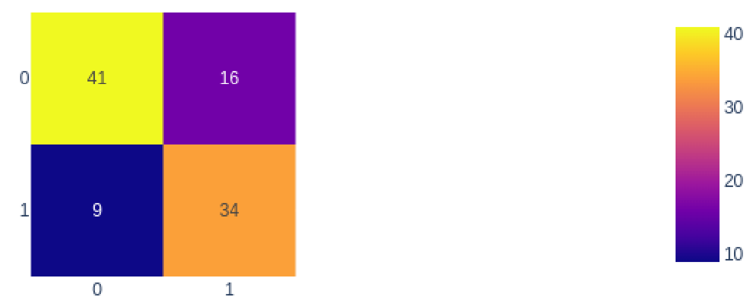
3 Remove duplicates from output

Unique most common for label==1 from train

0	ввод	23
1	срыв	19
2	сдача	18
3	задержка	11
4	перенос	10
5	отставание	10

Lets make confusion matrix

Baseline confusion matrix



	tag_words_baseline
val_size	100.00
TP	34.00
TN	41.00
FP	16.00
FN	9.00
precision	0.68
recall	0.79
accuracy	0.75
F1.0	0.73

Baseline summary:
tag words classifier has somewhat acceptable metrics of recall and accuracy. Precision is ok

Not bad for such simple logic

Pros:
* Understandable clear logic

Cons:
* poor metrics scores
* not business applicable

Possible problems:
* Not additive subset
* Low val size

Possible application:
1 tune tag words semantically
2 auto markup test.csv of 10k rows with this logic

Model

Research into
bert with ru applicability

If it is possible - use pretrained model and transfer train on data

Something lightweight is a priority

https://habr.com/ru/post/567028/
https://habr.com/ru/post/562064/

lightweight models:
1 cointegrated/rubert-tiny
2 cointegrated/rubert-tiny2

seems like SOTA for ru cases is
https://huggingface.co/DeepPavlov/rubert-base-cased-sentence/tree/main

Try lightweight first

Data preparation

1 Using val set balanced labels
val size 100

2 Using standart split
train test split: 0.8/0.2
train val split: 0.8/0.2

Class disbalance fix:
Cross Entropy loss function
Using sklearn class weight in loss
tune class weight

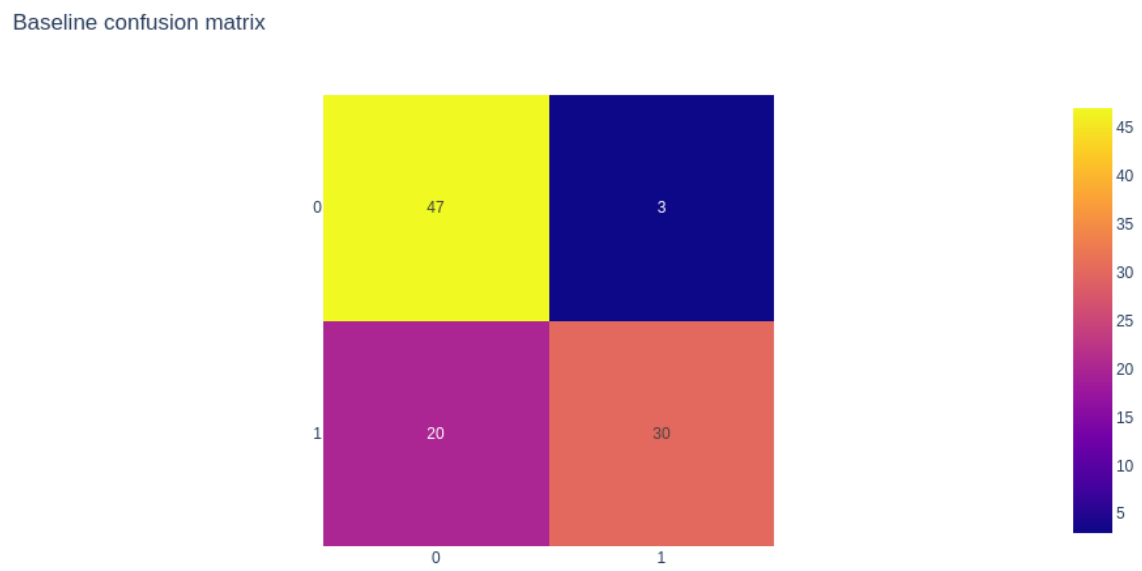
Data class weight: [0.6, 2.9]

rubert-tiny

v1

epochs: 10
val metrics
{'val_size': 100,

 'TP': 30,
 'TN': 47,
 'FP': 3,
 'FN': 20,
 'precision': 0.9090909090909091,
 'recall': 0.6,
 'accuracy': 0.77}

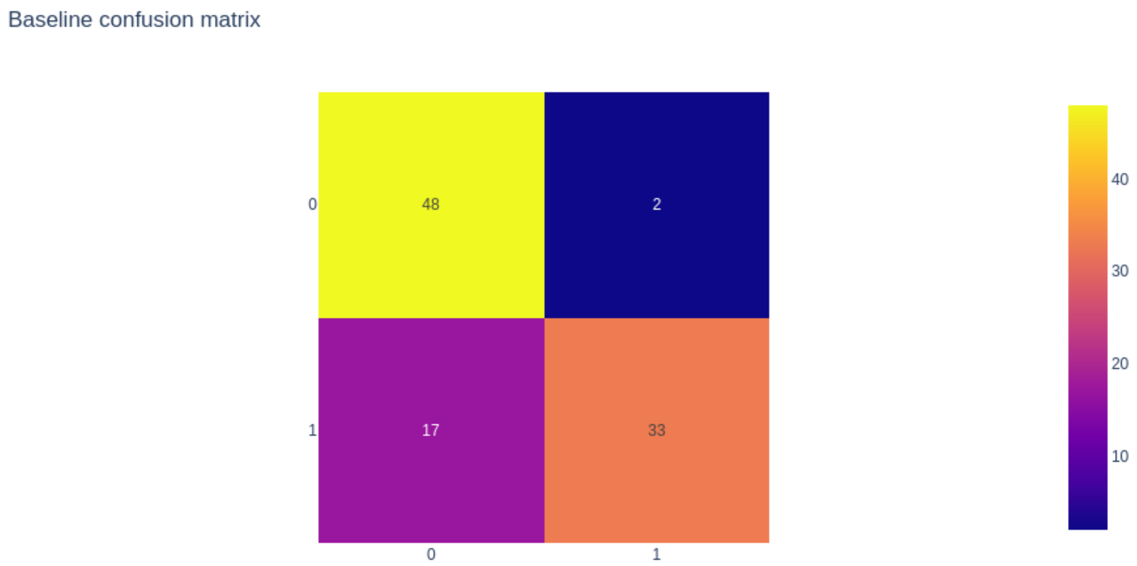


Lots of FN, weak recall. Not usable

v2

Added class_weight modifier

epochs:10
val metrics:
Train loss 0.07243817533471635 accuracy 0.9821673525377228
Val loss 1.1398921276954934 accuracy 0.81
{'val_size': 100,
 'TP': 33,
 'TN': 48,
 'FP': 2,
 'FN': 17,
 'precision': 0.9428571428571428,
 'recall': 0.66,
 'accuracy': 0.81}



Little improvement, but not usable

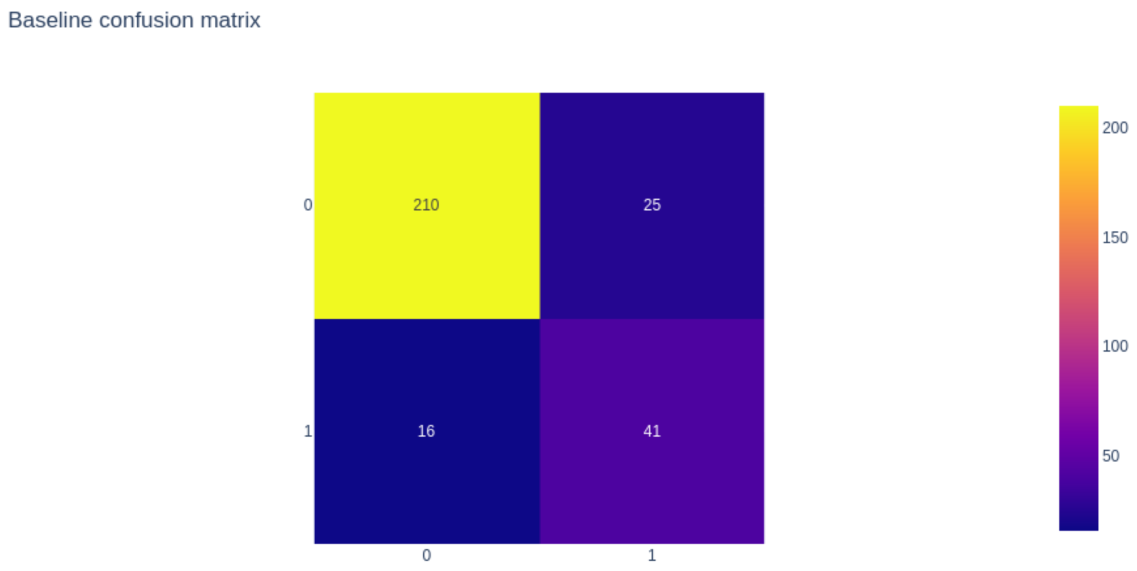
rubert-tiny2

- no significant improvement on 50/50 val set. Adding

v1

rubert-tiny2
train test split: 0.2
class_weight: [0.0001, 1]
epochs: 10

{'test_size': 292,
 'TP': 41,
 'TN': 210,
 'FP': 25,
 'FN': 16,
 'precision': 0.6212121212121212,
 'recall': 0.7192982456140351,
 'accuracy': 0.8595890410958904,
 'F1': 0.6666666666666667}



v2

rubert-tiny2
train test split: 0.2,
class_weight: 'balanced' [0.6, 2.9]
epochs: 10

```
{'test_size': 292,
 'TP': 40,
 'TN': 217,
 'FP': 26,
 'FN': 9,
 'precision': 0.6060606060606061,
 'recall': 0.8163265306122449,
 'accuracy': 0.8801369863013698,
 'F1': 0.6956521739130436}
```

Thats much better, recall has grown

v3

Using val 50/50 set

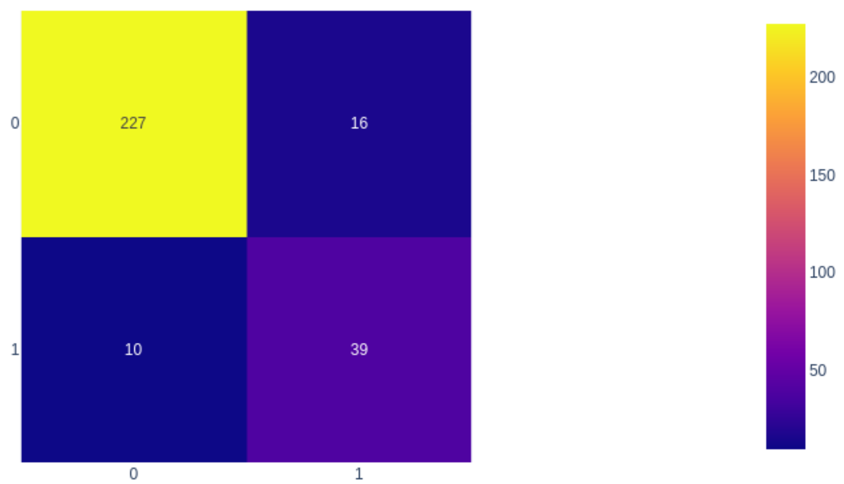
tag: ./bert_time_1668632322.pt
rubert-tiny2
train test split: 0.2
class_weight: 'balanced' [0.6, 2.9]
epochs: 10

val data classes 0.5

Train loss 0.12190150908360418 accuracy 0.9802744425385934
Val loss 1.8717926587964757 accuracy 0.76

{'test_size': 292,
 'TP': 39,
 'TN': 227,
 'FP': 16,
 'FN': 10,
 'precision': 0.7090909090909091,
 'recall': 0.7959183673469388,
 'accuracy': 0.910958904109589,
 'F1': 0.75}

Baseline confusion matrix



Acceptable for test run

v4

{'test_size': 312,
 'TP': 53,
 'TN': 240,
 'FP': 9,
 'FN': 10,
 'precision': 0.8548387096774194,
 'recall': 0.8412698412698413,
 'accuracy': 0.9391025641025641,
 'F1': 0.848}

DeepPavlov

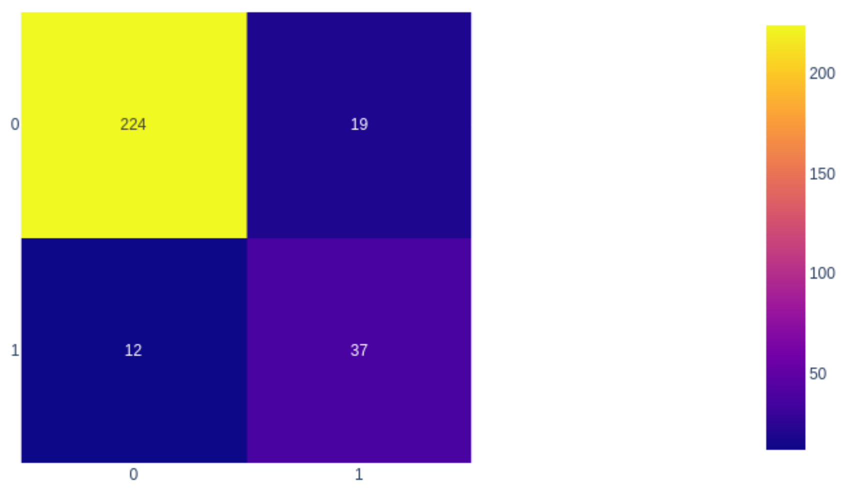
v1

train test split: 0.2,
class_weight: 'balanced' [0.6, 2.9]
epochs: 10
val set 50/50

Train loss 0.09935613534079109 accuracy 0.9845626072041166
Val loss 1.8192038302950095 accuracy 0.76

```
{'test_size': 292,
 'TP': 37,
 'TN': 224,
 'FP': 19,
 'FN': 12,
 'precision': 0.6607142857142857,
 'recall': 0.7551020408163265,
 'accuracy': 0.8938356164383562,
 'F1': 0.7047619047619047}
```

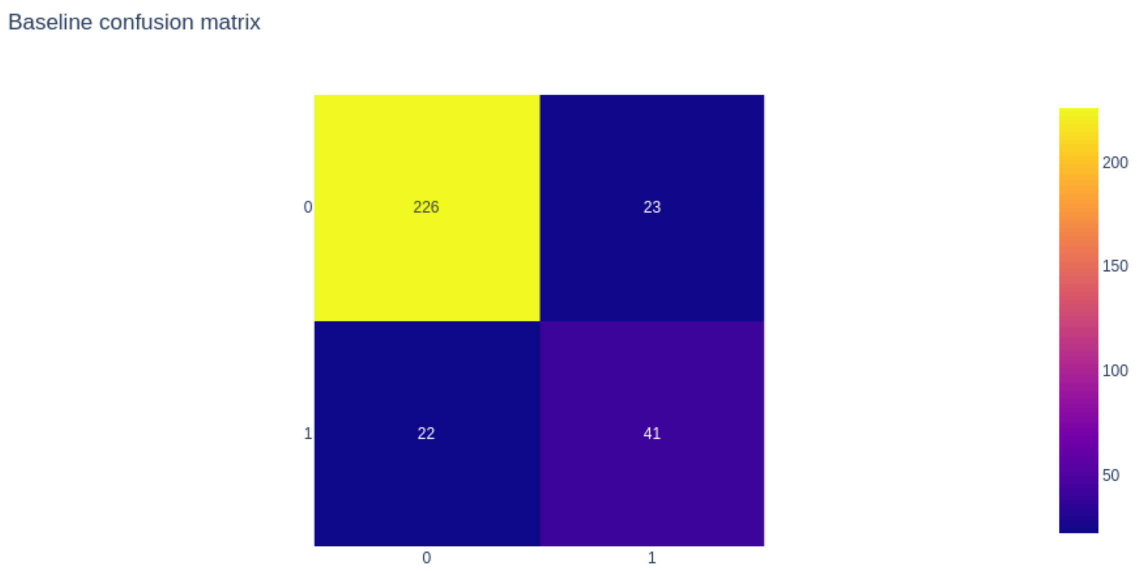
Baseline confusion matrix



Ok, but we had better results with rubert-tiny2_v3 v2

Standart train test val split 0.8/0.2
class_weight: ‘balanced’ [0.6, 2.9]

{'test_size': 312,
 'TP': 41,
 'TN': 226,
 'FP': 23,
 'FN': 22,
 'precision': 0.640625,
 'recall': 0.6507936507936508,
 'accuracy': 0.8557692307692307,
 'F1': 0.6456692913385828}



Weird that results are worse

Summary

Use rubert-tiny2_v3 for inference

Next steps:
- Cross val training (I dont have much GPU time)
- more epochs
- tune class weight

- Tune model optimizer
- get more data
- maybe research for another model