

Данные

Текст был очищен от знаков препинания (кроме точек), большие буквы были переведены в маленькие. Label из float сконвертирован в int (т.к. pytorch выдавал ошибку)

```
[ ] 1 patterns = "[A-Za-z0-9!#$%&'()*+,-/;<=>?@[\\]^_`{|}~\"'\"-]+"
```

```
[ ] 1 def lemmatize(doc):
2     doc = re.sub(patterns, ' ', doc)
3     tokens = []
4     doc = doc.lower()
5     for token in doc.split():
6         if token:
7             tokens.append(token)
8     return tokens
```

```
[ ] 1 df_test['comment'] = df_test['comment'].apply(lambda x: ' '.join(lemmatize(x)))
2 df_test
```

		Unnamed: 0	comment	toxic
0	14051	весь мир знает свиньи русские. ваше имя первой...	1	
1	13291	речь ютуб подсовывает самых популярных играх. ...	0	
2	10641	аллергия ещё звуковые спецэффекты будут	0	
3	8855	купить такую красоту	0	
4	1948	почему тупой объяснил конкретно похуй раз. ток...	1	
...	
1437	8659	брат моего соседа общежитию. хороший парень до...	0	
1438	9860	последние годы снимают откровенный шлак. знаю ...	0	
1439	8290	соня неплох пока дорохо. дексп смысл покупать ...	0	
1440	7196	плотва икрой которая крупная рублей выходить.	0	
1441	6769	похожая история стенкой ребёнка любитель поигр...	0	

1442 rows x 3 columns

Затем текст был токенизирован с помощью BertTokenizer и сохранён как тензор.

Модель

В качестве модели был использован классификатор Bert, в конце которого были прибавлены слои: Dropout, Linear, Relu (для получения вероятности при предсказании принадлежности к классу Relu была сменена на Сигмоиду)

```
[ ] 1 class BertClassifier(nn.Module):
2
3     def __init__(self, dropout=0.5):
4
5         super(BertClassifier, self).__init__()
6
7         self.bert = BertModel.from_pretrained('bert-base-cased')
8         self.dropout = nn.Dropout(dropout)
9         self.linear = nn.Linear(768, 2)
10        self.relu = nn.ReLU()
11
12    def forward(self, input_id, mask):
13
14        _, pooled_output = self.bert(input_ids= input_id, attention_mask=mask, return_dict=False)
15        dropout_output = self.dropout(pooled_output)
16        linear_output = self.linear(dropout_output)
17        final_layer = self.relu(linear_output)
18
19    return final_layer
```

В качестве Функции потерь взята CrossEntropyLoss (кросс-энтропия).
В качестве оптимизатора – Adam с lr=1e-6.

На тренировку было 90% датасета (10500 строк), на валидацию - 10% (1200 строк)

Модель училась 7 эпох (~2.5 часа). По причине того, что модель обучалась в GoogleColab, где есть лимит на использование GPU. Так что я посчитал 7 эпох оптимальным выбором.

```
[ ] 1 train(model,
2       df_train,
3       df_val,
4       LR,
5       EPOCHS)
```

100%|██████████| 5253/5253 [19:53<00:00, 4.40it/s]
Epochs: 1 | Train Loss: 0.313 | Train Accuracy: 0.667 | Val Loss: 0.290 | Val Accuracy: 0.705
100%|██████████| 5253/5253 [19:56<00:00, 4.39it/s]
Epochs: 2 | Train Loss: 0.259 | Train Accuracy: 0.751 | Val Loss: 0.247 | Val Accuracy: 0.771
100%|██████████| 5253/5253 [19:56<00:00, 4.39it/s]
Epochs: 3 | Train Loss: 0.222 | Train Accuracy: 0.801 | Val Loss: 0.237 | Val Accuracy: 0.789
100%|██████████| 5253/5253 [19:56<00:00, 4.39it/s]
Epochs: 4 | Train Loss: 0.199 | Train Accuracy: 0.831 | Val Loss: 0.211 | Val Accuracy: 0.800
100%|██████████| 5253/5253 [19:57<00:00, 4.39it/s]
Epochs: 5 | Train Loss: 0.175 | Train Accuracy: 0.853 | Val Loss: 0.210 | Val Accuracy: 0.811
100%|██████████| 5253/5253 [19:58<00:00, 4.38it/s]
Epochs: 6 | Train Loss: 0.162 | Train Accuracy: 0.866 | Val Loss: 0.201 | Val Accuracy: 0.833
100%|██████████| 5253/5253 [19:57<00:00, 4.39it/s]
Epochs: 7 | Train Loss: 0.147 | Train Accuracy: 0.883 | Val Loss: 0.210 | Val Accuracy: 0.815

Тестирование

В качестве метрики тестирования была выбрана метрика f1-score (roc-auc не был использован по причине того, что датасет был неравномерен – токсичных комментариев было лишь 30%). Дополнительно хотелось посмотреть на ассигасу(точность), поэтому она также включена в метрики.

```
accuracy = tensor([0.8627])
precision = tensor([0.7867])
recall = tensor([0.8095])
f1_score = tensor([0.7980])
```

В среднем модель обучилась одинаково хорошо для обоих классов. Это видно по recall и precision (которые равны в среднем 80%).

Инференс:

```
[ ] 1 def inference(df):
2     labels = list(df['class_true'])
3     texts = [
4         tokenizer(text, padding='max_length', max_length = 512, truncation=True, return_tensors="pt")
5         for text in df['text']
6     ]
7
8     dict_of_output = {'class_prediction':[],
9                       'probabilities':[]}
10
11     with torch.no_grad():
12         for text in texts:
13             mask = text['attention_mask'].to('cpu')
14             input_id = text['input_ids'].squeeze(1).to('cpu')
15             model.to('cpu')
16
17             output = model(input_id, mask)
18
19             label_of_edu = output.argmax(dim=1).item() # 1 - toxic, 0 - no toxic
20             sigmoid = 1 / (1 + math.exp(-output[0][label_of_edu]))
21
22             dict_of_output['class_prediction'].append(label_of_edu)
23             dict_of_output['probabilities'].append(round(sigmoid, 4))
24
25     return pd.concat([df, pd.DataFrame(data=dict_of_output)], axis=1)
```

```
[ ] 1 df_inference = inference(df_test.rename(columns={"comment": "text", "toxic": "class_true"}))
```

```
[ ] 1 df_inference
```

Unnamed: 0		text	class_true	class_prediction	probabilities
0	14051	весь мир знает свиньи русские. ваше имя первой...	1	1	0.9657
1	13291	речь ютуб подсовывает самых популярных играх. ...	0	0	0.9868
2	10641	аллергия ещё звуковые спецэффекты будут	0	0	0.9819
3	8855	купить такую красоту	0	0	0.9600
4	1948	почему тупой объяснил конкретно похуй раз. ток...	1	1	0.9749
...
1437	8659	брат моего соседа общежитию. хороший парень до...	0	0	0.6705
1438	9860	последние годы снимают откровенный шлак. знаю ...	0	0	0.9897
1439	8290	соня неплох пока дорохо. дексп смысл покупать ...	0	0	0.9924
1440	7196	плотва икрой которая крупная рублей выходить.	0	0	0.9907
1441	6769	похожая история стенкой ребёнка любитель поигр...	0	0	0.9906

1442 rows x 5 columns

