МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ
индивидуальному домашнему заданию
по дисциплине «Нейронные сети»
Тема: Quora Insincere Questions Classification

Студент гр. 0304 _____ Зуев М.Е.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург
2025

Цель работы.

В соревновании требуется разработать модель, которая сможет различать «искренние» вопросы, задаваемые пользователями, и «неискренние» вопросы, которые могут быть связаны с оскорблениями, провокациями или другими нежелательными типами контента.

Задача.

Модель нужно было обучить классифицировать вопросы как:

- Искренние, которые предполагают реальный интерес и желание получить ответ — «Как улучшить навыки программирования на Python?».
- Неискренние, которые скорее всего являются оскорбительными, троллинговыми, спамом или содержат элементы, нарушающие правила платформы — «Почему программисты такие тупые?».

В соревновании оценки моделей производятся с использованием F1-Score — метрики, которая гармонически балансирует точность (precision) и полноту (recall):

Выполнение работы.

В качестве исходных данных выступают 4 файла:

- embeddings.zip — содержит архив предобученных эмбеддингов GoogleNews-vectors-negative300, glove.840B.300d, paragram_300_sl999, wiki-news-300d-1M,
- sample_submission.csv — шаблон csv-файла для отправки предсказаний на соревновательную платформу
- test.csv — данные для валидации
- train.csv — данные для обучения

Сначала данные загружаются в формат pandas Dataframe. В train содержится около 13 млн вопросов пользователей (см. рис. 1), в test — 375 тысяч (см. рис. 2). Получается отношение примерно 80/20.

	qid	question_text	target
0	00002165364db923c7e6	How did Quebec nationalists see their province...	0
1	000032939017120e6e44	Do you have an adopted dog, how would you enco...	0
2	0000412ca6e4628ce2cf	Why does velocity affect time? Does velocity a...	0
3	000042bf85aa498cd78e	How did Otto von Guericke used the Magdeburg h...	0
4	0000455dfa3e01eae3af	Can I convert montra helicon D to a mountain b...	0
...	...	...	...
1306117	ffffcc4e2331aaf1e41e	What other technical skills do you need as a c...	0
1306118	ffffd431801e5a2f4861	Does MS in ECE have good job prospects in USA ...	0
1306119	ffffd48fb36b63db010c	Is foam insulation toxic?	0
1306120	ffffec519fa37cf60c78	How can one start a research project based on ...	0
1306121	ffffed09fedb5088744a	Who wins in a battle between a Wolverine and a...	0

1306122 rows × 3 columns

Рисунок 1 — Train

	qid	question_text
0	0000163e3ea7c7a74cd7	Why do so many women become so rude and arroga...
1	00002bd4fb5d505b9161	When should I apply for RV college of engineer...
2	00007756b4a147d2b0b3	What is it really like to be a nurse practitio...
3	000086e4b7e1c7146103	Who are entrepreneurs?
4	0000c4c3fbe8785a3090	Is education really making good people nowadays?
...	...	...
375801	ffff7fa746bd6d6197a9	How many countries listed in gold import in in...
375802	ffffa1be31c43046ab6b	Is there an alternative to dresses on formal p...
375803	ffffae173b6ca6bfa563	Where I can find best friendship quotes in Tel...
375804	ffffb1f7f1a008620287	What are the causes of refraction of light?
375805	fffff85473f4699474b0	Climate change is a worrying topic. How much t...

375806 rows × 2 columns

Рисунок 2 — Test

Далее нужно выяснить, как между собой соотносятся классы (см. рис. 3)

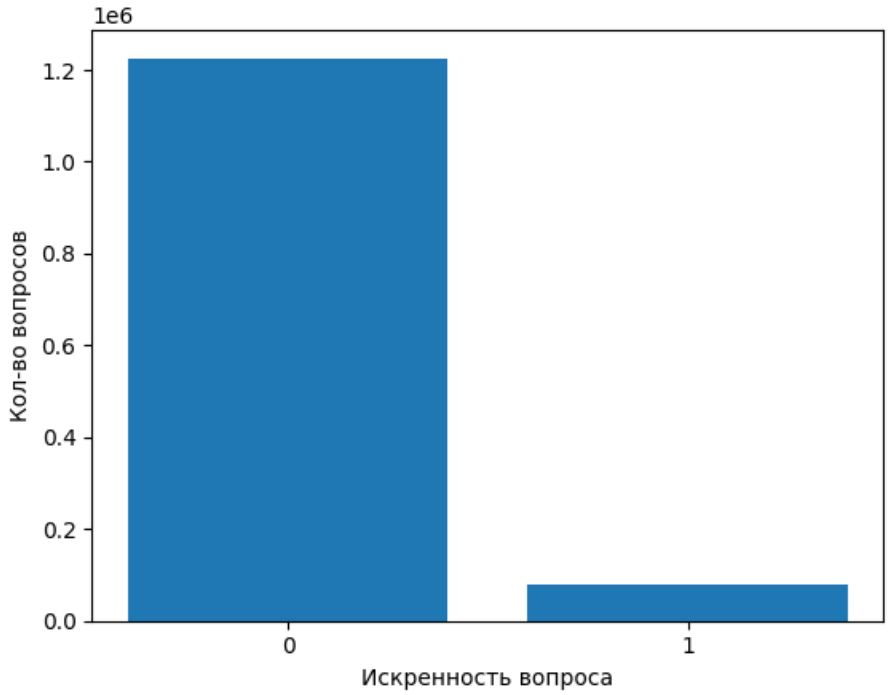

Рисунок 3 — Классовое неравенство

Отношение отрицательных примеров (вопрос искренний) к положительным (вопрос неискренний) примерно 15 к 1 — на лицо дисбаланс классов. Неудивительно, что в качестве метрики была выбрана F1.

Далее следует заняться предобработкой текста.

Нужно поставить до и после знаков пунктуации по пробелу, то есть «,» заменить на « , ». Это необходимо для дальнейшей токенизации — разбиения исходного текста на отдельные токены, в данном случае слова и знаки.

Затем следует замена слов-сокращений по типу *aren't* на полные слова *are not*. Это также нужно для того, чтобы в дальнейшем было удобнее разбивать строки на токены.

Теперь нужно посмотреть, сколько слов максимум вмещает в себя один вопрос (см. рис. 4). Ответ — 603. Значит, нужно дозаполнить нулями другие вопросы, количество слов в которых <603.

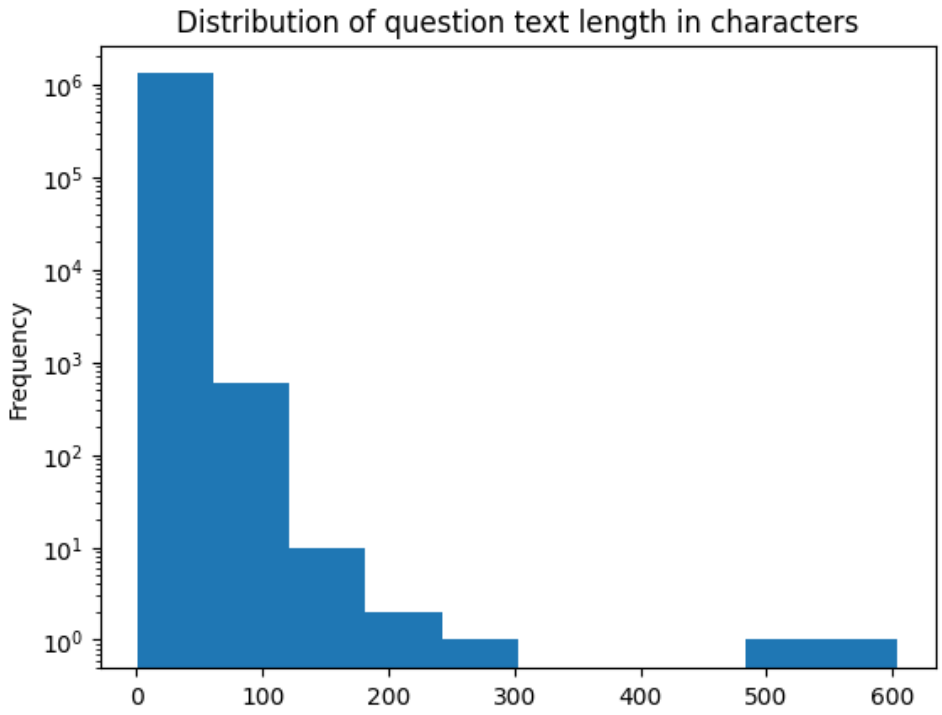

Рисунок 4 — Распределение длин вопросов

Теперь на основе предобученных эмбеддингов нужно составить матрицу эмбеддингов согласно индексам, которые были присвоены словам из вопросов.

Поскольку имеется явный дисбаланс, то в качестве разбивки на train и test множества применяется параметр stratify, чтобы в каждой разбивке сохранялось отношение классов, как в исходной выборке (15 к 1), для улучшения качества обучения (см. рис. 5):

```
X_train, X_test, y_train, y_test = train_test_split(X_train,
                                                    y_train,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y_train)
```

Рисунок 5 — Применение stratify при разбивке на train, test

Архитектура сети

Нейронная сеть принимает на вход последовательности индексов слов (например, X_train), преобразует их в векторы эмбеддингов, обрабатывает через LSTM (двунаправленный слой), а затем через несколько полносвязных

слоев для получения выходного значения (вероятности принадлежности к классу). Так же имеется dropout-слой для придания большей обобщающей способности сети (p=0.1), и слой батч-нормализации для стабилизации обучения.

К сожалению, залить результаты submission.csv на Kaggle технически почему-то не дает. Тем не менее, результаты на валидационном множестве такие (см. рис. 6):

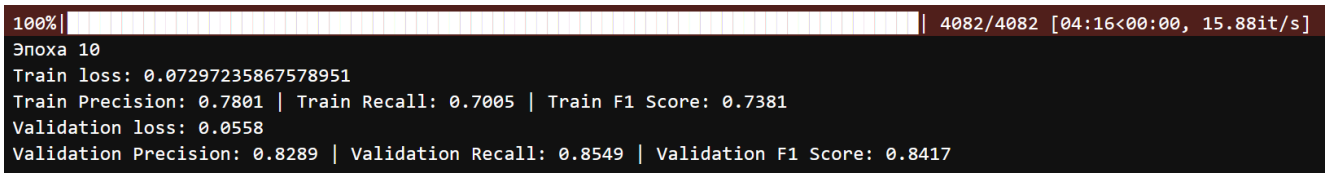

Рисунок 6 — Результаты обучения по прошествии 10 эпох

Вывод.

В ходе работы была проведена классификация «Искренности» текстов Quora на основе нейронной сети с использованием предобученных эмбеддингов, двунаправленного LSTM-слоя и полносвязного слоя с 16 нейронами. Код программы приведен в приложении А.

ПРИЛОЖЕНИЕ А
КОД ПРОГРАММЫ

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

from nltk.tokenize import TweetTokenizer
import datetime
import lightgbm as lgb
from scipy import stats
from scipy.sparse import hstack, csr_matrix
from sklearn.model_selection import train_test_split, cross_val_score
from wordcloud import WordCloud
from collections import Counter
from nltk.corpus import stopwords
from nltk.util import ngrams
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.svm import LinearSVC
from sklearn.multiclass import OneVsRestClassifier
import time
pd.set_option('max_colwidth',400)

from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from sklearn.preprocessing import OneHotEncoder

import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader
from torch.nn.utils.rnn import pack_padded_sequence, pad_packed_sequence
from torch.autograd import Variable
import torch.utils.data
import random
import warnings
warnings.filterwarnings("ignore", message="F-score is ill-defined and being set to 0.0 due to
no predicted samples.")
import re
from torch.optim.lr_scheduler import StepLR, ReduceLROnPlateau, CosineAnnealingLR
```

```
# Установка сида для воспроизводимости эксперимента
def seed_torch(seed = 1029):
    random.seed(seed)
    os.environ['PYTHONHASHSEED'] = str(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    torch.cuda.manual_seed(seed)
    torch.backends.cudnn.deterministic = True
```

```
train = pd.read_csv("train.csv")
val = pd.read_csv("test.csv")
sub = pd.read_csv('sample_submission.csv')
```

```
puncts = [',', '.', '"', ':', ')', '(', '-', '!', '?', '|', ';', "'", '$', '&', '/', '[', ']',
'>', '%', '=', '#', '*', '+', '\\', '•',  '~', '@', '£',
 '·', '_', '{', '}', '©', '^', '®', '`',  '<', '→', '°', '€', '™', '›',  '♥', '←', '×', '§',
'″', '′', 'Â', '█', '½', 'à', '…',
 '“', '★', '”', '–', '●', 'â', '►', '−', '¢', '²', '¬', '░', '¶', '↑', '±', '¿', '▾', '═',
'¦', '║', '―', '¥', '▓', '—', '‹', '─',
 '▒', '：', '¼', '⊕', '▼', '▪', '†', '■', '’', '▀', '¨', '▄', '♫', '☆', 'é', '¯', '♦', '¤',
'▲', 'è', '¸', '¾', 'Ã', '⋅', '‘', '∞',
 '∙', ')', '↓', '、', '│', '（', '»', '，', '♪', '╩', '╚', '³', '・', '╦', '╣', '╔', '╗', '━',
'❤', 'ï', 'Ø', '¹', '≤', '‡', '√', ]


def space_punkt(x):
    """Добавляет пробелы до и после знаков пунктуации — это необходимо для дальнейшей токениза-
ции"""
    x = str(x)
    for punct in puncts:
        x = x.replace(punct, f' {punct} ')
```

```
    return x

def clean_numbers(x):
    x = re.sub('[0-9]{5,}', '#####', x)
    x = re.sub('[0-9]{4}', '####', x)
    x = re.sub('[0-9]{3}', '###', x)
    x = re.sub('[0-9]{2}', '##', x)
    return x

mispell_dict = {"aren't" : "are not",
                "can't" : "cannot",
                "couldn't" : "could not",
                "didn't" : "did not",
                "doesn't" : "does not",
                "don't" : "do not",
                "hadn't" : "had not",
                "hasn't" : "has not",
                "haven't" : "have not",
                "he'd" : "he would",
                "he'll" : "he will",
                "he's" : "he is",
                "i'd" : "I would",
                "i'll" : "I will",
                "i'm" : "I am",
                "isn't" : "is not",
                "it's" : "it is",
                "it'll":"it will",
                "i've" : "I have",
                "let's" : "let us",
                "mightn't" : "might not",
                "mustn't" : "must not",
                "shan't" : "shall not",
                "she'd" : "she would",
                "she'll" : "she will",
                "she's" : "she is",
                "shouldn't" : "should not",
                "that's" : "that is",
                "there's" : "there is",
                "they'd" : "they would",
                "they'll" : "they will",
                "they're" : "they are",
                "they've" : "they have",
                "we'd" : "we would",
                "we're" : "we are",
                "weren't" : "were not",
                "we've" : "we have",
                "what'll" : "what will",
                "what're" : "what are",
                "what's" : "what is",
                "what've" : "what have",
                "where's" : "where is",
                "who'd" : "who would",
                "who'll" : "who will",
                "who're" : "who are",
                "who's" : "who is",
                "who've" : "who have",
                "won't" : "will not",
                "wouldn't" : "would not",
                "you'd" : "you would",
                "you'll" : "you will",
                "you're" : "you are",
                "you've" : "you have",
                "'re": " are",
                "wasn't": "was not",
                "we'll":" will",
                "didn't": "did not",
                "tryin'":"trying"}

def _get_mispell(mispell_dict):
    mispell_re = re.compile('(%s)' % '|'.join(mispell_dict.keys()))
    return mispell_dict, mispell_re

mispellings, mispellings_re = _get_mispell(mispell_dict)

def replace_typical_misspell(text):
    def replace(match):
        return mispellings[match.group(0)]
    return mispellings_re.sub(replace, text)
```

```
# Clean the text
train["question_text"] = train["question_text"].apply(lambda x: space_punkt(x.lower()))
val["question_text"] = val["question_text"].apply(lambda x: space_punkt(x.lower()))

# Clean numbers
train["question_text"] = train["question_text"].apply(lambda x: clean_numbers(x))
val["question_text"] = val["question_text"].apply(lambda x: clean_numbers(x))

# Clean speelings
train["question_text"] = train["question_text"].apply(lambda x: replace_typical_misspell(x))
val["question_text"] = val["question_text"].apply(lambda x: replace_typical_misspell(x))
```

```
max_features = 120000
tk = Tokenizer(lower = True, filters='', num_words=max_features)
full_text = list(train['question_text'].values) + list(val['question_text'].values)
tk.fit_on_texts(full_text)

train_tokenized = tk.texts_to_sequences(train['question_text'].fillna('missing'))
val_tokenized = tk.texts_to_sequences(val['question_text'].fillna('missing'))

train['question_text'].apply(lambda x: len(x.split())).plot(kind='hist');
plt.yscale('log');
plt.title('Distribution of question text length in characters');
```

```
max_len = 603
X_train = pad_sequences(train_tokenized, maxlen = max_len)
X_val = pad_sequences(val_tokenized, maxlen = max_len)
X_train
```

```
y_train = train['target'].values
```

```
X_train, X_test, y_train, y_test = train_test_split(X_train,
                                                    y_train,
                                                    test_size=0.2,
                                                    random_state=42,
                                                    stratify=y_train)
```

```
embed_size = 300
embedding_path = "embeddings/glove.840B.300d/glove.840B.300d.txt"
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embedding_index = dict(get_coefs(*o.split(" ")) for o in open(embedding_path, encoding='utf-8',
errors='ignore'))
# all_embs = np.stack(embedding_index.values())
# emb_mean,emb_std = all_embs.mean(), all_embs.std()
emb_mean,emb_std = -0.005838499, 0.48782197
word_index = tk.word_index
nb_words = min(max_features, len(word_index))
embedding_matrix = np.random.normal(emb_mean, emb_std, (nb_words + 1, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None: embedding_matrix[i] = embedding_vector
```

```
embedding_path = "embeddings/paragram_300_sl999/paragram_300_sl999.txt"
def get_coefs(word,*arr): return word, np.asarray(arr, dtype='float32')
embedding_index = dict(get_coefs(*o.split(" ")) for o in open(embedding_path, encoding='utf-8',
errors='ignore') if len(o)>100)
# all_embs = np.stack(embedding_index.values())
# emb_mean,emb_std = all_embs.mean(), all_embs.std()
emb_mean,emb_std = -0.0053247833, 0.49346462
embedding_matrix1 = np.random.normal(emb_mean, emb_std, (nb_words + 1, embed_size))
for word, i in word_index.items():
    if i >= max_features: continue
    embedding_vector = embedding_index.get(word)
    if embedding_vector is not None: embedding_matrix1[i] = embedding_vector
```

```
embedding_matrix = np.mean([embedding_matrix, embedding_matrix1], axis=0)
del embedding_matrix1
```

```
X_train = torch.tensor(X_train, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.float32)

X_test = torch.tensor(X_train, dtype=torch.long)
y_test = torch.tensor(y_train, dtype=torch.float32)

X_val = torch.tensor(X_val, dtype=torch.long)

train = torch.utils.data.TensorDataset(X_train, y_train)
test = torch.utils.data.TensorDataset(X_test, y_test)
val = torch.utils.data.TensorDataset(X_val)
batch_size = 256
```

```
train_loader = torch.utils.data.DataLoader(train, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test, batch_size=batch_size)
val_loader = torch.utils.data.DataLoader(val, batch_size=batch_size)
```

```
import torch
import torch.nn as nn

class NeuralNet(nn.Module):
    def __init__(self):
        super(NeuralNet, self).__init__()

        hidden_size = 128
        # Слой эмбеддингов
        self.embedding = nn.Embedding(max_features, embed_size)
        self.embedding.weight = nn.Parameter(torch.tensor(embedding_matrix,
dtype=torch.float32))
        self.embedding.weight.requires_grad = False  # Замораживаем веса эмбеддингов

        # Dropout для эмбеддингов
        self.embedding_dropout = nn.Dropout(0.1)

        # LSTM слой
        self.lstm = nn.LSTM(embed_size, hidden_size, bidirectional=True, batch_first=True)

        # Полносвязные слои
        self.linear = nn.Linear(hidden_size * 4, 16)  # hidden_size * 4 = avg_pool + max_pool
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.1)
        self.bn = nn.BatchNorm1d(16)  # Batch Normalization
        self.out = nn.Linear(16, 1)  # Выходной слой для бинарной классификации

    def forward(self, x):
        # Эмбеддинги
        h_embedding = self.embedding(x)
        h_embedding = self.embedding_dropout(h_embedding)

        # LSTM
        h_lstm, _ = self.lstm(h_embedding)  # Выход LSTM

        # Глобальные пулинги
        avg_pool = torch.mean(h_lstm, 1)   # Среднее значение по последовательности
        max_pool, _ = torch.max(h_lstm, 1) # Максимальное значение по последовательности

        # Конкатенация пулов
        conc = torch.cat((avg_pool, max_pool), 1)

        # Полносвязные слои
        conc = self.relu(self.linear(conc))
        conc = self.dropout(conc)
        conc = self.bn(conc)
        out = self.out(conc)

        return out
```

```
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
```

```
from sklearn.metrics import precision_score, recall_score, f1_score
device = 'cuda' if torch.cuda.is_available() else 'cpu'
batch_size = batch_size
from tqdm import tqdm
model = NeuralNet().to(device)
loss_fn = nn.BCEWithLogitsLoss()
optimizer = optim.AdamW(model.parameters())
best_f1 = 0
for epoch in range(10):
    model.train()
    train_loss = 0
    all_train_preds = []
    all_train_labels = []

    for x_batch, y_batch in tqdm(train_loader):
        x_batch, y_batch = x_batch.to(device).long(), y_batch.to(device).unsqueeze(1)

        optimizer.zero_grad()

        y_pred = model(x_batch)

        loss = loss_fn(y_pred, y_batch)
```

```
        loss.backward()

        optimizer.step()

        train_loss += loss.item() / len(train_loader)

        probs = torch.sigmoid(y_pred)
        pred_labels = (probs > 0.5).float()
        all_train_preds.extend(pred_labels.cpu().numpy())
        all_train_labels.extend(y_batch.cpu().numpy())

    all_train_labels = np.array(all_train_labels).flatten()
    all_train_preds = np.array(all_train_preds).flatten()

    precision_train = precision_score(all_train_labels, all_train_preds)
    recall_train = recall_score(all_train_labels, all_train_preds)
    f1_train = f1_score(all_train_labels, all_train_preds)
    print(f"Эпоха {epoch+1}")
    print(f"Train loss: {train_loss}")
    print(f"Train Precision: {precision_train:.4f} | Train Recall: {recall_train:.4f} | Train
F1 Score: {f1_train:.4f}")


    model.eval()
    val_loss = 0
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for x_val, y_val in test_loader:
            x_val, y_val = x_val.to(device).long(), y_val.to(device).unsqueeze(1)
            preds = model(x_val)
            loss = loss_fn(preds, y_val)
            val_loss += loss.item() / len(test_loader)

            probs = torch.sigmoid(preds)  # логиты → вероятности
            pred_labels = (probs > 0.5).float()
            all_preds.extend(pred_labels.cpu().numpy())  # Собираем все предсказания
            all_labels.extend(y_val.cpu().numpy())  # Собираем все истинные метки

    all_labels = np.array(all_labels).flatten()
    all_preds = np.array(all_preds).flatten()

    precision_val = precision_score(all_labels, all_preds)
    recall_val = recall_score(all_labels, all_preds)
    f1_val = f1_score(all_labels, all_preds)

    print(f"Validation loss: {val_loss:.4f}")
    print(f"Validation Precision: {precision_val:.4f} | Validation Recall: {recall_val:.4f} |
Validation F1 Score: {f1_val:.4f}")
    if f1_val > best_f1:
        best_f1 = f1_val
        torch.save(model.state_dict(), "best_model.pth")

    val_preds = np.zeros((len(val_loader.dataset)))

    for i, (x_batch,) in enumerate(val_loader):
        x_batch = x_batch.to(device).long()
        y_pred = model(x_batch).detach()

        # Записываем предсказания с учетом реального размера батча
        val_preds[i * batch_size:(i+1) * batch_size] = sigmoid(y_pred.cpu().numpy())[:, 0]
```

```
sub['prediction'] = (val_preds > 0.5).astype(int)
sub.to_csv("submission.csv", index=False)
```