Анализ текстов

Интернет-магазин запускает новый сервис. Теперь пользователи могут редактировать и дополнять описания товаров, как в вики-сообществах. То есть клиенты предлагают свои правки и комментируют изменения других. Магазину нужен инструмент, который будет искать токсичные комментарии и отправлять их на модерацию.

В нашем распоряжении набор данных с разметкой о токсичности правок.

Необходимо построить модель умеющую классифицировать комментарии на позитивные и негативные, со значением метрики качества *F1* не менее 0.75.

!pip install torch --index-url https://download.pytorch.org/whl/cu118!pip install protobuf==3.20.3 --user!pip install lightgbm

In [1]
```
# импортируем библиотеки
import os
import time
import numpy as np
import pandas as pd
import torch
import transformers
from tqdm import notebook
from sklearn.linear_model import LogisticRegression
from lightgbm import LGBMClassifier
from catboost import CatBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import f1_score
from sklearn.model_selection import train_test_split, GridSearchCV
```

Подготовка

Обзор данных

Откроим файл с данными, изучим общую информацию.

In [2]
```
pth1 = '/datasets/toxic_comments.csv'
pth2 = 'https://restricted/datasets/toxic_comments.csv'

if os.path.exists(pth1):
    data = pd.read_csv(pth1)
else:
    try:
        data = pd.read_csv(pth2)
    except:
        print('Что-то не так, файл с данными не найден!!!')
```

In [3]
```
# Уменьшим размер df в связи с аппаратными ограничениями
data = data.sample(4000, random_state=22).reset_index(drop=True)
```

In [4]
```
# Выведем общую инф о df
def df_info(df):
    display(df.head())
    display(df.info())
    display(df.describe())
```

In [5]
```
df_info(data)
```

	Unnamed: 0	text	toxic
0	111987	(1) Not that I can think of. (2) I'm not sure....	0
1	80940	"\n\n Email \n\nHiya,\n\nEmail for you. Can yo...	0
2	103441	If what you've found isn't WP:OR then please g...	0
3	122458	"\n\n Please do not vandalize pages, as you di...	0
4	58876	Danny Green article peer review \n\nThis artic...	0

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 4000 entries, 0 to 3999
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype 
---  ------      --------------  ----- 
 0   Unnamed: 0  4000 non-null   int64 
 1   text        4000 non-null   object
 2   toxic       4000 non-null   int64 
dtypes: int64(2), object(1)
memory usage: 93.9+ KB
None
```

	Unnamed: 0	toxic
count	4000.000000	4000.000000
mean	80053.370000	0.099750
std	45943.627098	0.299704
min	14.000000	0.000000
25%	39742.250000	0.000000
50%	81476.000000	0.000000
75%	118949.750000	0.000000
max	159434.000000	1.000000

Данные представлены следующими признаками:

- `text` — содержит текст комментария;
- `toxic` — целевой признак.

Пропуски в данных отсутствуют, однако наблюдается явный дисбаланс классов:

In [6]
```
data['toxic'].value_counts(normalize = 1)
```

Out[6]
```
0    0.90025
1    0.09975
Name: toxic, dtype: float64
```

Соотношение классов в целевом признаке: ~ 10 / 90. Мы имеем дисбаланс классов. Учтем этот факт в дальнейшей работе

Целевой признак `toxic` — факт токсичности коментария. Будем решать задачу методами классификации.

Подготовка данных

Подготовим признаки с помощью BERT

In [7]
```
# Используем GPU для ускорения расчетов
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
print(device)
```

```
cuda:0
```

In [8]
```
# Загружаем модель
model_name = 'unitary/toxic-bert' # Предобученная модель
tokenizer = transformers.AutoTokenizer.from_pretrained(model_name, do_lower_case
model = transformers.AutoModel.from_pretrained(model_name).to(device) # Инициали
```

```
Some weights of the model checkpoint at unitary/toxic-bert were not used when ini
tializing BertModel: ['classifier.weight', 'classifier.bias']
- This IS expected if you are initializing BertModel from the checkpoint of a mod
el trained on another task or with another architecture (e.g. initializing a Bert
ForSequenceClassification model from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the checkpoint of a
model that you expect to be exactly identical (initializing a BertForSequenceClas
sification model from a BertForSequenceClassification model).
```

In [9]
```
# Преобразуем исходные тексты в список токенов с макс длинной текста 512 символа
tokenized = data['text'].apply(
    lambda x: tokenizer.encode(x, add_special_tokens=True,truncation=True, max_l

# Найдем максимальную длину векторов после токенизации
max_len = 0
for i in tokenized.values:
    if len(i) > max_len:
        max_len = len(i)

#Применим метод padding, чтобы после токенизации длины исходных текстов в корпус
padded = np.array([i + [0]*(max_len - len(i)) for i in tokenized.values])

# Поясним модели, что нули не несут значимой информации
attention_mask = np.where(padded != 0, 1, 0)
```

In [10]
```
# преобразование текстов в эмбеддинги
batch_size = 100

# сделаем пустой список для хранения эмбеддингов твитов
embeddings = []

# цикл по батчам
```

```
for i in notebook.tqdm(range(padded.shape[0] // batch_size)):
        # преобразуем данные
        batch = torch.LongTensor(padded[batch_size*i:batch_size*(i+1)]).to(devic
        # преобразуем маску
        attention_mask_batch = torch.LongTensor(attention_mask[batch_size*i:batc

        # Получение эмбеддингов
        with torch.no_grad():
            batch_embeddings = model(batch, attention_mask=attention_mask_batch)
        embeddings.append(batch_embeddings[0][:,0,:].cpu().numpy())
```

```
0%|          | 0/40 [00:00<?, ?it/s]
```

Разделим данные на обучающую и тестовую выборки в соотношении 80:20

In [11]:
```
# Подготовим признаки
features = np.concatenate(embeddings)
target = data['toxic']

# Разделим данные
(features_train, features_test,
     target_train, target_test) = train_test_split(features, target,
                                                   test_size=0.2,
                                                   random_state=22,
                                                   stratify = target)
print("Размерность тренеровочных данных", features_train.shape)
print("Размерность тестовых данных", features_test.shape)
```

```
Размерность тренеровочных данных (3200, 768)
Размерность тестовых данных (800, 768)
```

На данном этапе произведена загрузка данных и их подготовка:

- Для решения задачи предоставлен набор данных с разметкой о токсичности правок;
- Пропуски в данных отсутствуют;
- Наблюдается дисбаланс классов в целевом признаке ~ 10 / 90;
- Признаки подготовлены с помощью нейросети BERT;
- Данные разделены на обучающую и тестовую выборки в соотношении 80:20.

Обучение

Проведем обучение трех моделей: `LogisticRegression`, `LGBMClassifier`, `RandomForestClassifier`

Модель на основе LogisticRegression

In [12]:
```
%%time
model_lgr = LogisticRegression()

param_grid_lgr = {
    'penalty': ["l1","l2"],
    'C': [0.001, 0.01, 0.1, 1, 10],
    'solver': ['liblinear'],
    'random_state': [22],
```

```
    'class_weight': ['balanced']
}

# Обучение
cv_lgr = GridSearchCV(estimator=model_lgr,
                      param_grid=param_grid_lgr,
                      cv=3,
                      n_jobs=-1,
                      scoring='f1',
                      verbose=10
                     )
cv_lgr.fit(features_train, target_train)

cv_lgr_best_params = cv_lgr.best_params_
cv_lgr_best_score = round(cv_lgr.best_score_, 3)

cv_lgr_results = ['LogisticRegression',
                  cv_lgr_best_score]

print("best params", cv_lgr_best_params, "\nscore", cv_lgr_best_score)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
best params {'C': 1, 'class_weight': 'balanced', 'penalty': 'l2', 'random_state':
22, 'solver': 'liblinear'}
score 0.932
CPU times: total: 672 ms
Wall time: 5.05 s
```

Модель на основе LGBMClassifier

In [13]:

```
%%time

model_lgb = LGBMClassifier()

param_grid_lgb = {
    'max_depth': [25, 50],
    'learning_rate' : [0.01, 0.03],
    'n_estimators': range(100, 800, 100),
    'class_weight': ['balanced']
}

cv_lgb = GridSearchCV(estimator=model_lgb,
                      param_grid=param_grid_lgb,
                      cv=3,
                      n_jobs=-1,
                      scoring='f1',
                      verbose=1
                     )
cv_lgb.fit(features_train, target_train)

cv_lgb_best_params = cv_lgb.best_params_
cv_lgb_best_score = round(cv_lgb.best_score_, 1)

cv_lgb_results = ['LightGBM',
                  cv_lgb_best_score]

print("best params", cv_lgb_best_params, "\nscore", cv_lgb_best_score)
```

```
Fitting 3 folds for each of 28 candidates, totalling 84 fits
best params {'class_weight': 'balanced', 'learning_rate': 0.03, 'max_depth': 25,
'n_estimators': 300}
score 0.9
CPU times: total: 35.9 s
Wall time: 10min 3s
```

Модель на основе RandomForestClassifier

In [14]:
```
%%time
model_rfc = RandomForestClassifier()

param_grid_rfc = {
        'n_estimators': range(10, 410, 50),
        'max_depth' : [None] + [i for i in range(2, 11)],
        'random_state': [22],
        'class_weight': ['balanced']
}

# Обучение
cv_rfc = GridSearchCV(estimator=model_rfc,
                      param_grid=param_grid_rfc,
                      cv=3,
                      n_jobs=-1,
                      scoring='f1',
                      verbose=1
                     )
cv_rfc.fit(features_train, target_train)

cv_rfc_best_params = cv_rfc.best_params_
cv_rfc_best_score = round(cv_rfc.best_score_, 3)

cv_rfc_results = ['RandomForest',
                  cv_rfc_best_score]

print("best params", cv_rfc_best_params, "\nscore", cv_rfc_best_score)
```

```
Fitting 3 folds for each of 80 candidates, totalling 240 fits
best params {'class_weight': 'balanced', 'max_depth': 5, 'n_estimators': 210, 'ra
ndom_state': 22}
score 0.941
CPU times: total: 3.84 s
Wall time: 1min 31s
```

Анализ моделей

Сведем полученные в ходе обучения и кросвалидации моделей данные в таблицу, оценим качество предсказания

In [15]:
```
model_analytics = pd.DataFrame([cv_rfc_results, cv_lgr_results, cv_lgb_results],
                               columns=['Модель', 'Качество предсказания (F1)'])
model_analytics
```

Out [15]

	Модель	Качество предсказания (F1)
0	RandomForest	0.941
1	LogisticRegression	0.932
2	LightGBM	0.900

Исследование проводилось для трех моделей: `LogisticRegression`, `LGBMClassifier`, `RandomForestClassifier`.
В результате при помощи `GridSearchCV` были подобраны оптимальные параметры, проведено обучение моделей а так же проанализировано качество моделей.
Все модели удовлетворяют заданному условию - Значение метрики F1 не меньше 0.75.
Однако по итогам кросвалидации победителем является модель `RandomForestClassifier` со значение метрики F1 - 0,941.

Опираясь на требования заказчика, для дальнейшей работы рекомендуем применять модель на основе `RandomForestClassifier` со следующими гиперпараметрами 'class_weight': 'balanced', 'max_depth': 5, 'n_estimators': 210.

Тестирование лучшей модели

Проведем финальное тестирование модели

In [16]
```
# функция для анализа моделей
def model_analysis (features_train, target_train, features_test, target_test, mo
    start = time.time()
    model.fit(features_train, target_train)
    end = time.time()
    fit_time = round(end - start, 3)

    start = time.time()
    model_pred = model.predict(features_test)
    end = time.time()
    pred_time = round(end - start, 3)

    score = round(f1_score(target_test, model_pred), 3)

    return [model_name, score, fit_time, pred_time]
```

In [17]
```
model = RandomForestClassifier(**cv_rfc_best_params)
model = model_analysis(features_train, target_train, features_test, target_test,

print('Модель:', model[0],
     '\nКачество предсказания (F1):', model[1],
     '\nВремя обучения модели:', model[2], 'сек.',
     '\nВремя предсказания модели:', model[3], 'сек.',
     )
```

```
Модель: RandomForestClassifier
Качество предсказания (F1): 0.958
Время обучения модели: 3.111 сек.
Время предсказания модели: 0.019 сек.
```

Общий вывод исследования

В нашем распоряжении набор данных с разметкой о токсичности правок.
Перед нами была поставлена задача:
Построить модель умеющую классифицировать комментарии на позитивные и негативные.

Из особенностей:

- Размер тестовой выборки - 10% от исходных данных;
- Значение метрики F1 на тестовой выборке должно быть не менее 0.75.

Работа проходила в несколько этапов:

1. Подготовка данных:
    - Обзор данных:
        - Данные представлены следующими признаками:
            - `text` — содержит текст комментария;
            - `toxic` — факт токсичности коментария, целевой признак.
        - Пропуски в данных отсутствуют;
        - Наблюдается дисбаланс классов в целевом признаке ~ 10/90, учтен при обучении моделей.
    - Подготовка данных:
        - Подготовили признаки с помощью нейросети BERT;
        - Создали признаки;
        - Разделим данные на обучающую и тестовую выборки в соотношении 80:20.
2. Обучение моделей:
    - При помощи `GridSearchCV` провели обучение с кросвалидацией для следующих моделей:
        - LogisticRegression;
        - LGBMClassifier;
        - RandomForestClassifier.
    - На основе метрики F1, выявленой при кросвалидации, выбрали лучшую модель - `RandomForestClassifier`
3. Провели финальное тестирование модели на тестовой выборке:
    - Модель RandomForestClassifier:
        - Качество предсказания (F1): 0.958;
        - Время обучения модели: 3.13 сек;
        - Время предсказания модели: 0.019 сек;
        - Гиперпараметры: class_weight': 'balanced', 'max_depth': 5, 'n_estimators': 210.

Таким образом поставленная задача реализована с учетом особенностей и пожеланий заказчика.