# Определение токсичности в комментариях с BERT

## Введение

В современном мире электронная коммерция является неотъемлемой частью повседневной жизни. Люди все чаще выбирают онлайн-покупки вместо традиционных магазинов. Однако, при выборе товара в интернет-магазине, покупатель не имеет возможности потрогать, попробовать или испробовать товар, как это можно сделать в обычном магазине. Поэтому, описание товара и отзывы других покупателей играют ключевую роль в принятии решения о покупке. В связи с этим, актуальным становится вопрос модерации комментариев и правок, оставленных пользователями, для поддержания качества контента на сайте.

**Цель исследования**

Основной целью исследования является разработка модели, которая будет классифицировать комментарии на позитивные и негативные, с целью выявления токсичных комментариев и отправки их на модерацию.

**Этапы работы:**

- Исследование данных, необходимо понять с какими данными мы имеем дело.
- Подготовка данных, создание эмбедингов для дальнейшего обучения моделей
- Создание и обучение 4х разных моделей позволит выбрать оптимальную для текущей задачи. В качетсве возможных варинатов рассмотрим 4 модели:
  - Логистическая регрессия
  - SVM
  - Градиентный бустинг lightGBM
  - DistilBERT
- Тестирование лучшей модели на тестовой выборке позволит непредвзято оценить работу модели.

# Подготовка

## Используемые библиотеки

```
!pip install scikit-learn==1.4.0 -q
!pip install matplotlib==3.8.4 -q
!pip install numpy==1.25.1 -q
!pip install seaborn -q
!pip install lightgbm -q
!pip install transformers[torch] -q
!pip install datasets -q

import torch
import transformers
from time import time
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import lightgbm as lgb
from tqdm import notebook
from datasets import Dataset
from transformers import DistilBertForSequenceClassification,
TrainingArguments, Trainer, DistilBertTokenizerFast
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.model_selection import cross_val_score,
RandomizedSearchCV, train_test_split
from sklearn.metrics import confusion_matrix, precision_score,
recall_score, f1_score
```

## Загрузка исходных данных

```
df = pd.read_csv(r'D:\down2\toxic_comments.csv', index_col=[0])
```

## Первичное ознакомление с данными

```
display(df, df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 159292 entries, 0 to 159450
Data columns (total 2 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   text    159292 non-null  object
 1   toxic   159292 non-null  int64
```

```
dtypes: int64(1), object(1)
memory usage: 3.6+ MB
```

```
                                                     text  toxic
0       Explanation\nWhy the edits made under my usern...      0
1       D'aww! He matches this background colour I'm s...      0
2       Hey man, I'm really not trying to edit war. It...      0
3       "\nMore\nI can't make any real suggestions on ...      0
4       You, sir, are my hero. Any chance you remember...      0
...                                                   ...    ...
159446  ":::::And for the second time of asking, when ...      0
159447  You should be ashamed of yourself \n\nThat is ...      0
159448  Spitzer \n\nUmm, theres no actual article for ...      0
159449  And it looks like it was actually you who put ...      0
159450  "\nAnd ... I really don't think you understand...      0

[159292 rows x 2 columns]

None
```

Явных пропусков нет, но индексы не совпадают.

```
df.reset_index(drop=True, inplace = True)

df.duplicated('text').sum()

0
```

Коментарии не повторяются.

```
df['toxic'].value_counts()

toxic
0    143106
1     16186
Name: count, dtype: int64

df['text'].head()

0    Explanation\nWhy the edits made under my usern...
1    D'aww! He matches this background colour I'm s...
2    Hey man, I'm really not trying to edit war. It...
3    "\nMore\nI can't make any real suggestions on ...
4    You, sir, are my hero. Any chance you remember...
Name: text, dtype: object
```

Похоже на типичные коментарии и что не удивтельно нормальных комментариев больше чем токсичных.

# Создание эмбеддингов

Используя кодировщик BERT создадим эмбеддинги коментариев.

Токенизируем текст.

```
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

# Токенизация текстов
encoded_texts = tokenizer(df["text"].astype('str').tolist(), padding=True, truncation=True, max_length=128, return_tensors='pt')

# Преобразование меток классов в тензор
labels = torch.tensor(df['toxic'])
```

```
C:\Users\Хозяин\AppData\Local\Programs\Python\Python311\Lib\site-packages\transformers\tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was not set. It will be set to `True` by default. This behavior will be depracted in transformers v4.45, and will be then set to `False` by default. For more details check this issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
```

Инициируем модель

```
model = transformers.DistilBertModel.from_pretrained('distilbert-base-uncased')
```

Из-за маленкой выборки для создания эмбеддингов, очень мало токсичных коментраиев попадает в обучающую выборку, модель не может полноценно ухватить закономерности, поэтому сбалансируем классы.

```
toxic_index = np.random.choice(df[df['toxic']==1].index, 600, replace = False)
no_toxic_index = np.random.choice(df[df['toxic']==0].index, 600, replace = False)
balance_toxic_index = np.concatenate([toxic_index,no_toxic_index])

padded_sample = encoded_texts.input_ids[balance_toxic_index]

attention_mask_sample = encoded_texts.attention_mask[balance_toxic_index]

# Инициализация переменной для хранения эмбеддингов
embeddings = []

# Обработка батчами
batch_size = 200
for i in notebook.tqdm(range(padded_sample.shape[0] // batch_size)):
```

```
    batch =
torch.LongTensor(padded_sample[batch_size*i:batch_size*(i+1)])
    attention_mask_batch =
torch.LongTensor(attention_mask_sample[batch_size*i:batch_size*(i+1)])

    with torch.no_grad():
        batch_embeddings = model(batch,
attention_mask=attention_mask_batch)

    embeddings.append(batch_embeddings[0][:,0,:].numpy())
```

{"model_id":"61620681c10742c4b479b9154783c3c6","version_major":2,"version_minor":0}

Созданые эмбеддинги используем для обучения моделей. После увеличения количества эмбеддингов с 1200 до 2000 метрика не изменилась.

# Создание моделей

Для получения наиболее точных и стабильных прогнозов обучим 4 различные модели. Каждая модель имеет свои сильные и слабые стороны и может лучше обрабатывать определенные аспекты данных. Например, логистическая регрессия может быть более чувствительной к линейным отношениям между признаками, в то время как градиентный бустинг может лучше обрабатывать нелинейные зависимости

## Логистическая регрессия

Простая и быстрая модель классификации, которая может быть обучена на эмбедингах, полученных с помощью word2vec, GloVe или других методов создания эмбедингов, однако не учитывает контекст комментария.

```
features = np.concatenate(embeddings)

X_train, X_test, y_train, y_test = train_test_split(features,
df['toxic'][balance_toxic_index], test_size=0.25, random_state=42,
stratify = df['toxic'][balance_toxic_index])

# Обучение модели логистической регрессии
model_log = LogisticRegression(max_iter=1000)

f1_score_log = cross_val_score(model_log, X_train, y_train, cv=5,
n_jobs=1, scoring = 'f1')
print(f'Метрика f1 для логистической регрессии на кросс валидации
{f1_score_log.mean():.2f}')
```

```
Метрика f1 для логистической регрессии на кросс валидации 0.87
```

Метрика соответсвует условию задачи, рассмотрим подробнее предсказания модели.

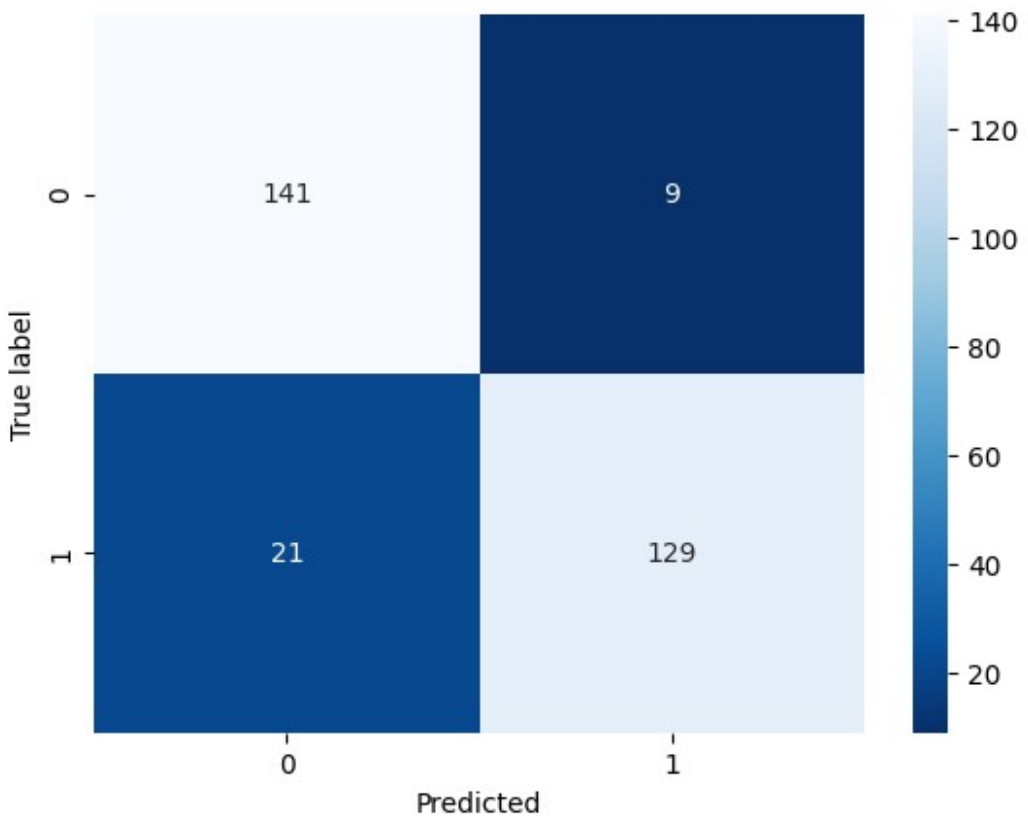
```
#Обучение модели
start = time()

model_log.fit(X_train, y_train)#Дообучаем модель на всей тренировочной выборке
end = time()
log_reg_time = (end-start)/60

print(f'Время обучения модели {log_reg_time} минуты')

start = time()

model_log.predict(X_test)

end = time()
pred_log_time = (end-start)

print(f'Время предсказания модели {pred_log_time} минуты')
```

```
Время обучения модели 0.003136785825093587 минуты
Время предсказания модели 0.0020182132720947266 минуты
```

Очень быстро

В конце создадим тестовую выборку, а пока используем test для валидации.

```
cm = confusion_matrix(y_test, model_log.predict(X_test))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues_r')
plt.ylabel('True label')
plt.xlabel('Predicted');
```




```
print(f'precision модели {precision_score(y_test,
model_log.predict(X_test)):.2f}, recall {recall_score(y_test,
model_log.predict(X_test)):.2f}')
```

```
precision модели 0.93, recall 0.86
```

Модель показывает результат удовлетворяющий требованиями задачи, посмотрим сможем ли мы достичь лучших результатов.

## SVM

SVM могут использоваться с эмбеддингами для нахождения гиперплоскости, которая разделяет классы в пространстве признаков. Однако требовательная к подбору гиперпараметров ядра и может переобучиться.

```
svc_model = SVC()

#Финашльный пайплайн
pipe_final = Pipeline([
    ('models', svc_model)
])

#Параметры пайплайна
param_grid = {
```

```
    'models__C': np.logspace(-2, 2, 20),  # Регуляризационный параметр
    'models__kernel': ['linear', 'poly', 'rbf', 'sigmoid'],  # Тип
ядра
    'models__degree': [3, 4, 5],  # Степень полинома, используется
только при kernel='poly'
    'models__gamma': ['scale', 'auto'], # Параметр ядра RBF
    'models__coef0': [0.0, 0.5, 1.0],  # Константа в ядре полинома и
сигмоиде
}


svc_cv = RandomizedSearchCV(
    pipe_final,
    param_grid,
    random_state = 42,
    cv=5,
    scoring='f1',
    n_jobs=1
)

svc_cv.fit(X_train, y_train)

print('Лучшие параметры:', svc_cv.best_estimator_)
print (f'Метрика модели на кросс-валидации:,
{svc_cv.best_score_:.2f}')
```

```
Лучшие параметры: Pipeline(steps=[('models',
                 SVC(C=37.92690190732246, coef0=1.0, degree=5,
gamma='auto'))])
Метрика модели регрессии на кросс-валидации:, 0.87
```

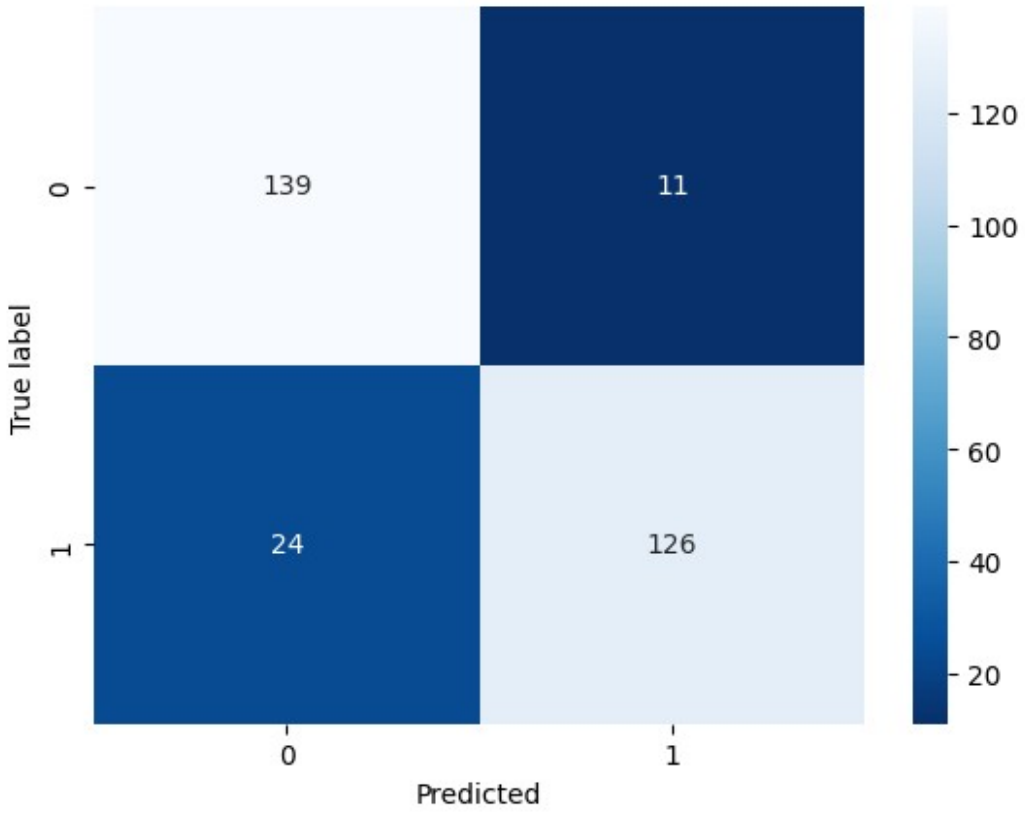
f1 идентична логистической регрессии

```
print(f'Время обучения модели  {svc_cv.refit_time_ :.2f} секунд')

start = time()

svc_cv.predict(X_test)

end = time()
pred_svc_time = (end-start)/60

print(f'Время предсказания модели {pred_svc_time} минуты')
```

```
Время обучения модели  0.13 секунд
Время предсказания модели 0.0016869425773620606 минуты
```

Обучение чуть дольше, предсказание чуть быстрее

```
cm = confusion_matrix(y_test, svc_cv.predict(X_test))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues_r')
plt.ylabel('True label')
plt.xlabel('Predicted');
```




```
print(f'precision модели {precision_score(y_test,
svc_cv.predict(X_test)):.2f}, recall {recall_score(y_test,
svc_cv.predict(X_test)):.2f}')
```

```
precision модели 0.92, recall 0.84
```

Модель показывает результат схожий с логистической регрессией.

## Градиентный бустинг LightGBM

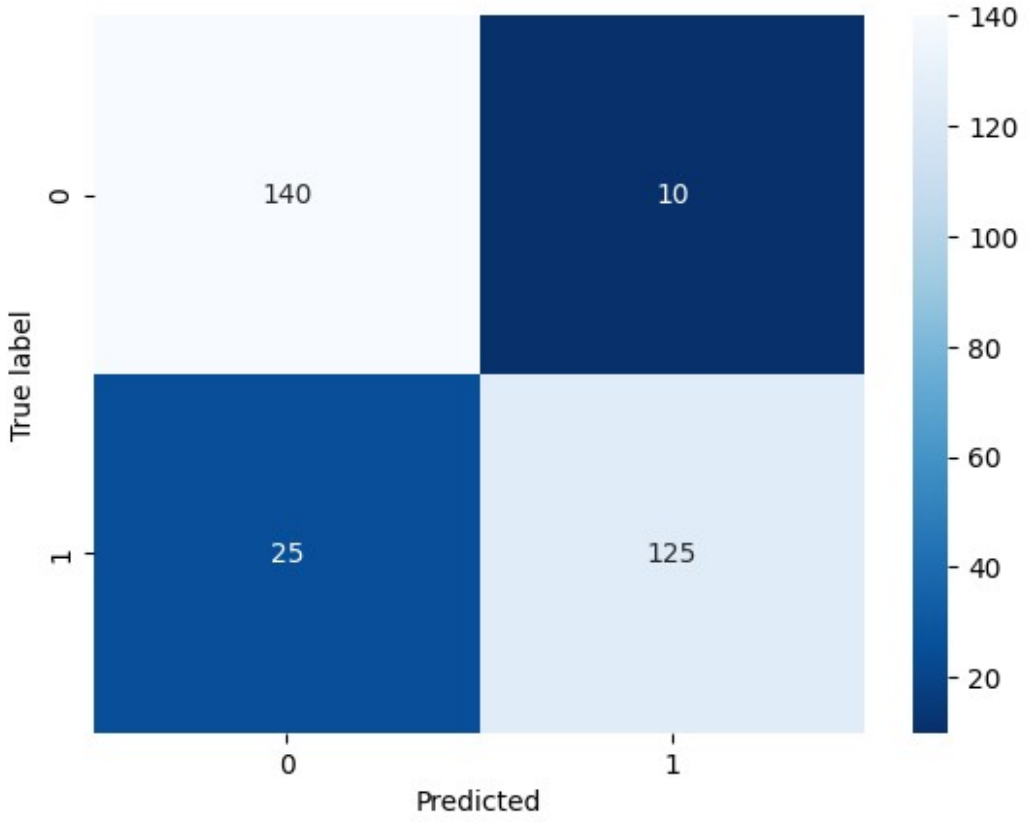
Быстрая и эффективная модель, которая может обрабатывать большие объемы данных и учитывать контекст комментария, если использовать эмбединги, полученные с помощью методов, основанных на контексте.

```
lgb_model = lgb.LGBMClassifier(verbose = -1)

#Финашльный пайплайн
pipe_final = Pipeline([
```

```
    ('models', lgb_model)
])

#Параметры пайплайна
param_grid = {
    'models__boosting_type': ['gbdt','dart'],  # Тип бустинга
    'models__metric': ['f1'],  # Метрика для оценки модели
    'models__num_leaves': range(2,16),  # Количество листьев в дереве
    'models__max_depth': range(1,16),# Максимальная глубина дерева
    'models__n_estimators': [100, 150, 200, 300]
}


GBM = RandomizedSearchCV(
    pipe_final,
    param_grid,
    random_state = 42,
    cv=5,
    scoring='f1',
    n_jobs=1
)

GBM.fit(X_train, y_train)

print('Лучшие параметры:', GBM.best_estimator_)
print ('Метрика модели на кросс-валидации:', GBM.best_score_)

Лучшие параметры: Pipeline(steps=[('models',
                 LGBMClassifier(max_depth=9, metric='f1',
n_estimators=150,
                                num_leaves=6, verbose=-1))])
Метрика модели регрессии на кросс-валидации: 0.8439076421933382

print(f'Время обучения модели  {GBM.refit_time_ :.2f} секунд')

start = time()

GBM.predict(X_test)

end = time()
pred_svc_time = (end-start)/60

print(f'Время предсказания модели {pred_svc_time} минуты')

Время обучения модели  1.50 секунд
Время предсказания модели 6.664594014485678e-05 минуты
```

Модель дольше обучается но быстрее предсказывает.

```
cm = confusion_matrix(y_test, GBM.predict(X_test))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues_r')
plt.ylabel('True label')
plt.xlabel('Predicted');
```




f1 метрика ниже чем у SVC, возможно из-за специфики обучающих данных.

```
print(f'precision модели {precision_score(y_test,
GBM.predict(X_test)):.2f}, recall {recall_score(y_test,
GBM.predict(X_test)):.2f}')
```

```
precision модели 0.93, recall 0.83
```

При обучении на эмбедингах, градиентный бустинг показал худшие результаты, чем предыдущие модели. Вероятно так могло произойти из-за склонности к переобучению, сложности настройки параметров или специфики самих данных.

## BERT

Модель классификации, основанная на трансформаторе, которая может обрабатывать большие объемы данных и учитывать контекст комментария. Требует больших вычислительных ресурсов и может быть медленной на больших наборах данных.

```
print(f'Время обучения модели  {GBM.refit_time_ :.2f} секунд')

start = time()

GBM.predict(X_test)

end = time()
pred_svc_time = (end-start)/60

print(f'Время предсказания модели {pred_svc_time} минуты')

# Создаем ноовую выборку для обучения
texts = list(df['text'][balance_toxic_index])
labels = list(df['toxic'][balance_toxic_index])

# Инициализация токенизатора
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-
uncased')

# Токенизация текстов
encoded_texts = tokenizer(texts, padding=True, truncation=True,
max_length=128, return_tensors='pt')

# Преобразование меток классов в тензор
labels = torch.tensor(labels)

# Создание датасета для обучения
class CustomDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels=None):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: tensor[idx] for key, tensor in
self.encodings.items()}
        if self.labels is not None:  # Для тренировочного набора
данных
            item['labels'] = self.labels[idx]
        return item

    def __len__(self):
        return len(self.encodings['input_ids'])

# Создание экземпляра датасета
dataset = CustomDataset(encoded_texts, labels)

# Определение аргументов обучения
training_args = TrainingArguments(
    output_dir='./results',
    per_device_train_batch_size=40,
```

```
    per_device_eval_batch_size=40,
    num_train_epochs=3,
    weight_decay=0.01,
    logging_dir='./logs',
)

# Инициализация модели и тренера
model =
DistilBertForSequenceClassification.from_pretrained('distilbert-base-
uncased', num_labels=2)
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=dataset,
)

start = time()

# Обучение модели
trainer.train()

end = time()
train_BERT_time = (end-start)/60
```

```
C:\Users\Хозяин\AppData\Local\Programs\Python\Python311\Lib\site-
packages\transformers\tokenization_utils_base.py:1601: FutureWarning:
`clean_up_tokenization_spaces` was not set. It will be set to `True`
by default. This behavior will be depracted in transformers v4.45, and
will be then set to `False` by default. For more details check this
issue: https://github.com/huggingface/transformers/issues/31884
  warnings.warn(
Some weights of DistilBertForSequenceClassification were not
initialized from the model checkpoint at distilbert-base-uncased and
are newly initialized: ['classifier.bias', 'classifier.weight',
'pre_classifier.bias', 'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able
to use it for predictions and inference.
```

```
<IPython.core.display.HTML object>
```

Создадим датасет для кросс валидации.

```
data_cv = []
for k in range(5):
    new_toxic_index =
np.random.choice(df[(df['toxic']==1)&(~df['toxic'].index.isin(toxic_in
dex))].index, 400, replace = False)
    new_no_toxic_index =
np.random.choice(df[(df['toxic']==0)&(~df['toxic'].index.isin(toxic_in
dex))].index, 400, replace = False)
```

```
data_cv.append(np.concatenate([new_toxic_index,new_no_toxic_index]))

scores = []
pred_BERT_time = []

for i in data_cv:
    # Новый набор данных
    new_texts = list(df['text'][i])
    new_labels = list(df['toxic'][i])

    # Токенизация новых данных
    new_encoded_texts = tokenizer(new_texts, padding=True,
truncation=True, max_length=128, return_tensors='pt')

    # Создание датасета
    new_dataset = CustomDataset(new_encoded_texts, labels=None)

    start = time()

    # Предсказание
    predictions = trainer.predict(new_dataset)

    # Преобразование логитов в метки классов
    preds = torch.argmax(torch.tensor(predictions.predictions), dim=1)

    end = time()
    pred_BERT_time.append((end-start)/60)

    scores.append(f1_score(new_labels, preds))

#Средняя метрика на валидации
print ('Метрика модели BERT на валидации:', np.mean(scores))
```

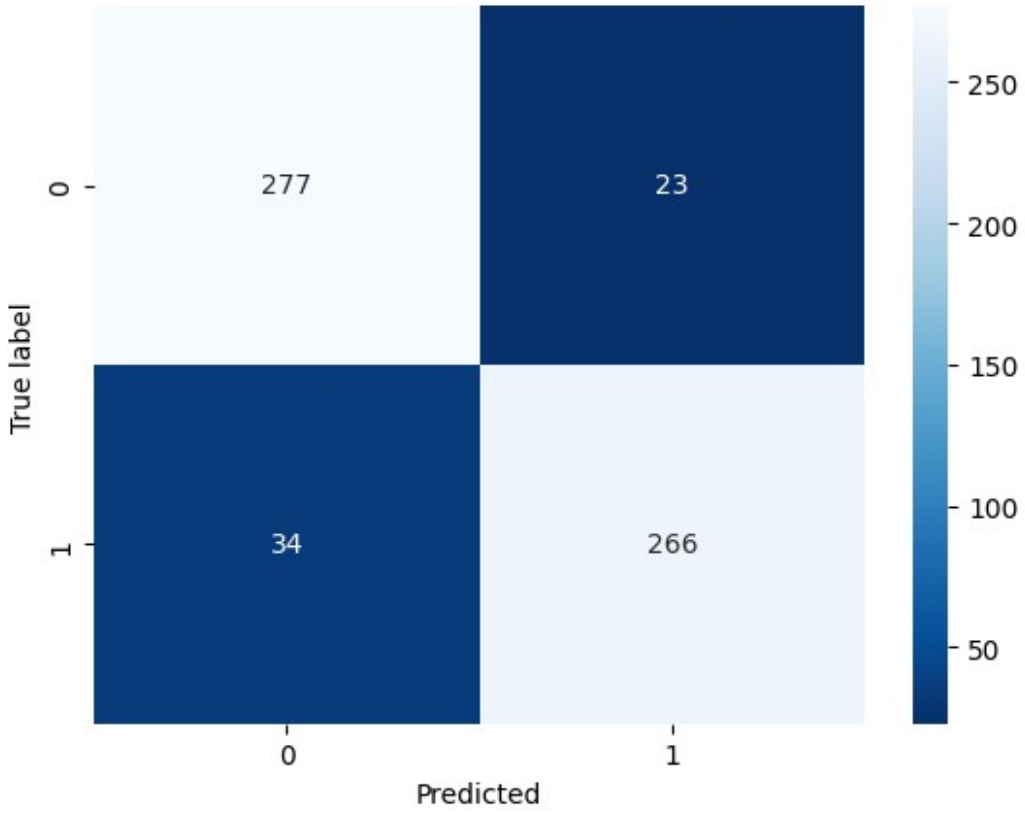
```
<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Метрика модели BERT на валидации: 0.9096883944587649
```

```
print(f'Время обучения модели  {train_BERT_time :.2f} минут')
print(f'Время предсказания модели  {np.mean(pred_BERT_time) :.2f}
секунд')
```

```
Время обучения модели  39.82 секунд
Время предсказания модели  2.79 секунд
```

Модель показывает самую высокую метрику по сравнению с другими моделями, однако и время предсказания модели большое. Обучение же заняло почти 40 минут.

```
cm = confusion_matrix(new_labels, preds)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues_r')
plt.ylabel('True label')
plt.xlabel('Predicted');
```




```
print(f'precision модели {precision_score(new_labels, preds):.2f},
recall {recall_score(new_labels, preds):.2f}')
```

```
precision модели 0.92, recall 0.89
```

Хотя метрики модели выше чем у других моделей, ей требуется больше времени на обучение.

## Тестирование модели

В качетсве финальной модели для теста выбираем BERT, создадим для нее большую тестовую выборку.

```
test_toxic_index =
np.random.choice(df[(df['toxic']==1)&(~df['toxic'].index.isin(toxic_in
dex))].index, 500, replace = False)
```

```
test_no_toxic_index =
np.random.choice(df[(df['toxic']==0)&(~df['toxic'].index.isin(toxic_in
dex))].index, 500, replace = False)
test_cv = np.concatenate([new_toxic_index,new_no_toxic_index])

# Новый набор данных
test_texts = list(df['text'][test_cv])
test_labels = list(df['toxic'][test_cv])

# Токенизация новых данных
test_encoded_texts = tokenizer(test_texts, padding=True,
truncation=True, max_length=128, return_tensors='pt')

# Создание датасета
test_dataset = CustomDataset(test_encoded_texts, labels=None)

# Предсказание
predictions = trainer.predict(test_dataset)

# Преобразование логитов в метки классов
preds = torch.argmax(torch.tensor(predictions.predictions), dim=1)

#Метрика на тесте
print ('Метрика модели BERT на тестовой выборке:',
f1_score(test_labels, preds))
```

```
<IPython.core.display.HTML object>
```

```
Метрика модели BERT на тестовой выборке: 0.9032258064516129
```

Модель показывает стабильные результатты на валидации и на тесте.

## Итоги

Используя только малую часть доступных данных, были выбраны 600 случайных комментариев, по 300 каждого класса, для создания эмбедингов с помощью BERT, на которых в последствии были обучена модели. По результатам обучения 4-х различных моделей на данных о комментариях со балансированными метками классов, можно сделать следующие выводы:

- Модели опорных векторов (SVM) и логистической регрессии показали практически идентичную производительность, с метрикой F1 около 0,87.
- Градиентный бустинг показал худшую производительность среди всех моделей, с метрикой F1 около 0,85.
- Модель BERT показала наилучшую производительность среди всех моделей, с метрикой F1 около 0,91.

При выборе модели для дальнейшего использования компаниям следует учитывать доступные ресурсы и цели. Несмотря на то, что модель BERT обладает самой высокой точностью, она также требует больших вычислительных ресурсов для предсказания меток

комментариев. Кроме того, для достижения наилучших результатов может потребоваться дополнительное время на обучение модели на всех доступных данных.

Если компания имеет ограниченные ресурсы и целью является быстрая и экономичная классификация комментариев, модели логистической регрессии или опорных векторов могут быть более подходящим выбором, несмотря на их более низкую точность. Однако, важно учитывать, что дальнейшее увеличение количества эмбеддингов для обучения этих моделей, скорее всего, не приведет к существенному улучшению производительности.

В целом, выбор модели для дальнейшего использования должен основываться на балансе между точностью, ресурсами и целями компании.