Интернет-магазин «Викишоп» 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.

Обучим модель классифицировать комментарии на позитивные и негативные, построим модель со значением метрики качества *F1* не меньше 0.75. В нашем распоряжении набор данных с разметкой о токсичности правок.

Описание данных

Данные находятся в файле `toxic_comments.csv`. Столбец *text* в нём содержит текст комментария, а *toxic* — целевой признак.

1. Подготовка

In [1]:

```
import pandas as pd
import numpy as np
pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
from sklearn.model_selection import cross_val_predict
from sklearn.model_selection import train_test_split #импортируем функцию train_test_sp
lit из библиотеки sklearn

import warnings
warnings.filterwarnings('ignore')

from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
import lightgbm as lgb


from nltk.corpus import stopwords as nltk_stopwords
from sklearn.feature_extraction.text import TfidfVectorizer

from sklearn.model_selection import (cross_val_score, train_test_split,
                                     GridSearchCV, RandomizedSearchCV)
from sklearn.metrics import f1_score

from sklearn.metrics import mean_squared_error
import seaborn as sns
from sklearn.preprocessing import StandardScaler

from pylab import *
import matplotlib
import matplotlib.pyplot as plt

import re

from nltk.stem import WordNetLemmatizer


import nltk
nltk.download('wordnet')
from sklearn.datasets import load_files
nltk.download('stopwords')
import pickle
from nltk.corpus import stopwords
```

```
[nltk_data] Downloading package wordnet to /home/jovyan/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to /home/jovyan/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

In [2]:

```
#прочитаем файл с исходными данными df

df = pd.read_csv("/datasets/toxic_comments.csv")
```

In [3]:

```
df=df.head(60000)
```

In [4]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 60000 entries, 0 to 59999
Data columns (total 2 columns):
text     60000 non-null object
toxic    60000 non-null int64
dtypes: int64(1), object(1)
memory usage: 937.6+ KB
```

In [5]:

```
X = df['text'].values
```

In [6]:

```
WNlemma = nltk.WordNetLemmatizer()
from nltk.corpus import wordnet
from nltk import pos_tag
from nltk import word_tokenize, FreqDist
import nltk
nltk.download('averaged_perceptron_tagger')

def get_wordnet_pos(treebank_tag):
    if treebank_tag.startswith('V'):
        return wordnet.VERB
    elif treebank_tag.startswith('J'):
        return wordnet.ADJ
    elif treebank_tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /home/jovyan/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
```

In [7]:

```
def pre_process_with_pos_tag(text):

    #приведем к нижнему регистру
    text = text.lower()

    # уберем все специальные знаки
    text = re.sub(r'\W', ' ', str(text))

    # уберем одиночные буквы
    text = re.sub(r'\s+[a-zA-Z]\s+', ' ', text)

    # уберем одиночные буквы в начале предложения
    text = re.sub(r'\^[a-zA-Z]\s+', ' ', text)

    # заменим пробелы на одиночный пробел
    text = re.sub(r'\s+', ' ', text, flags=re.I)

    # Removing prefixed 'b'.  When you have a dataset in bytes format, the alphabet let
ter "b" is appended before every string. The regex ^b\s+ removes "b" from the start of
 a string.
    text = re.sub(r'^b\s+', '', text)

    tokens = nltk.word_tokenize(text)
    tokens = [t for t in tokens if len(t) > 2]
    tokens = [WNlemma.lemmatize(t, get_wordnet_pos(pos_tag(word_tokenize(t))[0][1])) fo
r t in tokens]
    text_after_process = " ".join(tokens)
    return text_after_process
```

df['lemm'] = df['text'].apply(pre_process_with_pos_tag)

In [8]:

```
df = df.dropna()
```

для того чтобы не тратить каждый раз время на лемматизацию, сохраним датафрейм с леммами в файл

df.to_csv("data_lemm.csv",index=False)

In [9]:

```
df_lemm = pd.read_csv('data_lemm.csv')
```

In [10]:

```
df_lemm = df_lemm.dropna()
```

In [11]:

```
df_lemm.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 59996 entries, 0 to 59999
Data columns (total 3 columns):
text     59996 non-null object
toxic    59996 non-null int64
lemm     59996 non-null object
dtypes: int64(1), object(2)
memory usage: 1.8+ MB
```

In [12]:

```
X = df_lemm['lemm'].values
y = df_lemm['toxic'].values
```

In [13]:

```
#Разделим исходные данные на обучающую, валидационную и тестовую выборки. В ходе исслед
ования изменила с 60-20-20 на 70-15-15
X_train, X_valid_40, y_train, y_valid_40 = train_test_split(X, y, test_size=0.4, random
_state=12345)
X_valid, X_test, y_valid, y_test = train_test_split(X_valid_40, y_valid_40, test_size=
0.5, random_state=12345)
```

In [14]:

```
from sklearn.feature_extraction.text import TfidfVectorizer
word_vectorizer = TfidfVectorizer(max_features = 5000, min_df = 5, max_df = 0.7, stop_w
ords = 'english', decode_error='replace', encoding='utf-8' )
```

In [15]:

```
#произведем fit_transform только для обучающей, а transform для валидационной и тестово
й выборки

features_train = word_vectorizer.fit_transform(X_train).toarray()
```

In [16]:

```
features_valid = word_vectorizer.transform(X_valid).toarray()
```

In [17]:

```
features_test = word_vectorizer.transform(X_test).toarray()
```

In [18]:

```
#https://stackabuse.com/text-classification-with-python-and-scikit-learn/
```

2. Обучение

In [19]:

```
#обучим модель линейной регрессии

model_LR = LogisticRegression()
model_LR.fit(features_train, y_train)
```

Out[19]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=Tru
e,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=
0,
                   warm_start=False)
```

In [20]:

```
#подберем гиперпараметры для модели случайного леса

model_RFC = RandomForestClassifier()
```

In [21]:

```
#для модели случаного дерева

def model_quality_rfc(features_x,target_x,features_y,target_y, estim, depth):
    estim_and_score=pd.DataFrame(columns=["estimators","max_depth",'f1'])
    i=0
    model= RandomForestClassifier(random_state=12345, n_estimators=estim, max_depth=dep
th)
    model.fit(features_x, target_x) #обучим модель
    predicted_y=model.predict(features_y) #предскажем по валидационной выборке

    f1=f1_score(target_y, predicted_y)

    estim_and_score.loc[i]=[estim, depth, f1]  #построим датафрейм с данными по F1 на в
алидационной и тестовой выборке
    i+=1
    print(estim_and_score)
```

In [22]:

```
#подберем гиперпараметры для того, чтобы достичь макс f1
for estim in range(30,600,200):
    for depth in range (5,200,50):
        model_quality_rfc(features_train,y_train,features_valid, y_valid, estim, depth)
```

```
   estimators  max_depth   f1
0        30.0        5.0  0.0
   estimators  max_depth        f1
0        30.0       55.0  0.626558
   estimators  max_depth       f1
0        30.0      105.0  0.72155
   estimators  max_depth        f1
0        30.0      155.0  0.732276
   estimators  max_depth   f1
0       230.0        5.0  0.0
   estimators  max_depth        f1
0       230.0       55.0  0.615894
   estimators  max_depth       f1
0       230.0      105.0  0.72179
   estimators  max_depth        f1
0       230.0      155.0  0.728741
   estimators  max_depth   f1
0       430.0        5.0  0.0
   estimators  max_depth        f1
0       430.0       55.0  0.619362
   estimators  max_depth        f1
0       430.0      105.0  0.724539
   estimators  max_depth        f1
0       430.0      155.0  0.732608
```

In [23]:

```
#обучим модель на лучших гиперпараметрах
model_RFC = RandomForestClassifier(max_depth= 30, n_estimators= 155)#class_weight='bala
nced', max_depth= 50, n_estimators= 500)
model_RFC.fit(features_train, y_train)
```

Out[23]:

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gin
i',
                       max_depth=30, max_features='auto', max_leaf_nodes=N
one,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=155,
                       n_jobs=None, oob_score=False, random_state=None,
                       verbose=0, warm_start=False)
```

In [24]:

```
# для модели LGBM

def model_quality_LGBM(features_x,target_x,features_y,target_y, estim):
    estim_and_score_lgbm=pd.DataFrame(columns=['n_estimators','f1'])
    i=0
    model_LGBM= lgb.LGBMClassifier(random_state=12345, n_estimators=estim)
    model_LGBM.fit(features_x, target_x) #обучим модель
    predicted_y=model_LGBM.predict(features_y) #предскажем по валидационной выборке

    f1=f1_score(target_y, predicted_y)

    estim_and_score_lgbm.loc[i]=[ estim, f1]  #построим датафрейм с данными по F1 на ва
лидационной и тестовой выборке
    i+=1
    print(estim_and_score_lgbm)
```

In [25]:

```
#подберем гиперпараметры для того, чтобы достичь макс f1
for estim in range (120,161,20):
    model_quality_LGBM(features_train,y_train,features_valid, y_valid, estim)
```

```
   n_estimators        f1
0         120.0  0.747769
   n_estimators        f1
0         140.0  0.752688
   n_estimators        f1
0         160.0  0.753138
```

In [26]:

```
#обучим модель на лучших гиперпараметрах
model_LGBM = lgb.LGBMClassifier(n_estimators = 180)#boosting_type= 'gbdt',class_weight
='balanced' , learning_rate = 0.01, n_estimators = 100, reg_lambda = 1)
model_LGBM.fit(features_train, y_train)
```

Out[26]:

```
LGBMClassifier(boosting_type='gbdt', class_weight=None, colsample_bytree=
1.0,
               importance_type='split', learning_rate=0.1, max_depth=-1,
               min_child_samples=20, min_child_weight=0.001, min_split_gai
n=0.0,
               n_estimators=180, n_jobs=-1, num_leaves=31, objective=None,
               random_state=None, reg_alpha=0.0, reg_lambda=0.0, silent=Tr
ue,
               subsample=1.0, subsample_for_bin=200000, subsample_freq=0)
```

3. Выводы

In [27]:

```
#посмотрим какая модель лучше всего себя покажет
predict_LR = model_LR.predict(features_test)
```

In [28]:

```
print(f1_score(y_test, predict_LR))
```

```
0.6977236633139228
```

In [29]:

```
predict_RFC = model_RFC.predict(features_test)
print(f1_score(y_test, predict_RFC))
```

```
0.34206896551724136
```

In [30]:

```
predict_LGBM  = model_LGBM.predict(features_test)
print(f1_score(y_test, predict_LGBM))
```

```
0.7585213634181469
```