Университет “Нетология”

Отчёт по теме
“Анализ эмоциональной окраски текста”

студент группы DS-27
Павлов Николай Геннадьевич

Санкт-Петербург
2022

Содержание

Постановка задачи

Применение методов обработки текста для анализа тональности твитов в качестве задачи классификации, на примере двух задач:

1) "Hate Speech and Offensive Language Dataset - research hate-speech detection" (перевод с англ. яз. «Набор данных о высказываниях с ненавистью и оскорбительном языке — исследование обнаружения оскорбительных высказываний»)

   Ссылка:https://www.kaggle.com/mrmorj/hate-speech-and-offensive-language-dataset

2) "Russian Troll Tweets - 3 million tweets from accounts associated with the 'Internet Research Agency'" («Твиты русских троллей — 3 миллиона твитов с аккаунтов, связанных с «Агентством интернет-исследований»)

   Ссылка: https://www.kaggle.com/fivethirtyeight/russian-troll-tweets

Необходимо заранее установить, т.к. существуют ограничение на объём отчёта в виде 15 страниц, то в тексте будут приводиться не все строки коды. За более подробной информацией необходимо отправиться в файлы:

- task-1-hate-tweets.ipynb
- task-1-hate-tweets_BERT.ipynb
- task-2-tweet-trolls.ipynb

Актуальность задачи в предметной области

Актуальность задачи определяется развитием сети Интернет и необходимостью систематизации и обработки больших объёмов текстовой информации.

Анализ тональности текста (англ. Sentiment analysis) относится к классу задач компьютерной лингвистики, заключающаяся в определении эмоциональной окраски (тональности) текста и, в частности, в выявлении эмоциональной оценки авторов по отношению к какие-либо объектам или лицам.

Данная задача относится, в общем случае, к типу задач по классификации информации, т.е. когда уже известны классы анализируемых объектов. К примеру, где категориями текстов могут быть тональные оценки. Примеры тональных оценок:

- позитивная;
- негативная;
- нейтральная.

Однако задачу можно отнести и к регрессии, когда известна числовая оценочная шкала текста. Например, эмоциональная шкала, где 0 это грустный текст, а 10 - весёлый текст. И далее модель предсказывает оценку для поступающего в неё текста.

Существует множество задач, где обработка тональности текстов позволяет упростить анализ эффективности рекламной и PR деятельности, анализ отзывов о товарах и услугах или определении языка вражды.

Перспективы на будущее моделей анализа тональности текста: к примеру, модели научатся выдавать не просто рейтинг всего текста, а полный отчёт обо всех высказанных эмоциях клиента по отношению к объектам и понятиям, интересующим заказчика. На примере отзывов о фильмах это может быть отношение зрителя к сценарию, режиссеру, оператору и актёрам, что может очень интересовать заказчика.

В дальнейшем это может изменить мир через рекомендательные системы, где располагаются огромные деньги, т.к. товары и услуги очень важно рекомендовать именно тем, кто потенциально может и хочет их купить, поэтому любая информация о потенциальном клиента востребована и будет использована.

В ближайшем будущем каждого из нас будут подстерегать сложнейшие системы, которые точно рассчитают, что если нам понравились книга X, фильм Y и актриса Z – то нам, скорее всего, понравится и новая коллекция костюмов от Q, которая сильно на любителя.

Метрики качества решения задачи

Метрики оценки качества решения используются такие же, как и обычно - для задач классификации: Precision, Recall, F1-score, Accuracy. В данном случае при расчёте вышеприведённых метрик используются следующие понятия, которые для наглядности отображены в таблице 1:

- TP — true positive, классификатор верно отнёс объект к рассматриваемому классу.
- TN — true negative, классификатор верно утверждает, что объект не принадлежит к рассматриваемому классу.
- FP — false positive, классификатор неверно отнёс объект к рассматриваемому классу.
- FN — false negative, классификатор неверно утверждает, что объект не принадлежит к рассматриваемому классу.

Таблица 1. Confusion matrix (матрица несоответствий)

	Принадлежит классу (P)	**Не принадлежит классу (N)**
Предсказана принадлежность классу	TP	FP
Предсказано отсутствие принадлежности к классу	FN	TN

- Accuracy (точность), показывает долю правильных классификаций. Несмотря на очевидность и простоту является одной из самых малоинформативных оценок классификаторов.

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- Recall (полнота), так же sensitivity и TPR (true positive rate), показывает долю найденных объектов класса к общему числу объектов класса. Иначе говоря то, насколько хорошо наш классификатор находит объекты из класса.

$$recall = \frac{TP}{TP + FN}$$

- Precision (да, тоже точность), показывает долю объектов класса среди объектов выделенных классификатором.

$$precision = \frac{TP}{TP + FP}$$

- F1-score

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера (в общем случае Fß) — среднее гармоническое precision и recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{precision \cdot recall}{(\beta^2 \cdot precision) + recall}$$

F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

Если классов становится больше двух, расчёт метрик усложняется. Если задача классификации на K классов ставится как K задач об отделении класса i от остальных (i=1, …, K), то для каждой из них можно посчитать свою матрицу ошибок. Затем есть два варианта получения итогового значения метрики из K матриц ошибок:

1. Усредняем элементы матрицы ошибок (TP, FP, TN, FN) между бинарными классификаторами, например, $TP = \frac{1}{K}\sum_{i=1}^{K} TP_i$. .Затем по одной усреднённой матрице ошибок считаем Precision, Recall, F-меру. Это называют **микроусреднением**.

2. Считаем Precision, Recall для каждого классификатора отдельно, а потом усредняем. Это называют **макроусреднением**.

Анализ

При анализе аналогичных примеров по тональности текста [6,7, 8] были переняты основные подходы к решению данной задачи: это алгоритмы векторизации, алгоритмы моделирования и метрики оценки качества решения, которые применялись в ходе исследования двух задач.

При анализе первой и второй задачи, видим следующие данные (см.рис.1, рис. 2):

	count	hate_speech	offensive_language	neither	class	tweet
0	3	0	0	3	2	!!! RT @mayasolovely: As a woman you shouldn't...
1	3	0	3	0	1	!!!!! RT @mleew17: boy dats cold...tyga dwn ba...
2	3	0	3	0	1	!!!!!! RT @UrKindOfBrand Dawg!!!! RT @80sbaby...
3	3	0	2	1	1	!!!!!!!! RT @C_G_Anderson: @viva_based she lo...
4	6	0	6	0	1	!!!!!!!!!! RT @ShenikaRoberts: The shit you...

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 24783 entries, 0 to 25296
Data columns (total 6 columns):
 #   Column              Non-Null Count  Dtype 
---  ------              --------------  ----- 
 0   count               24783 non-null  int64 
 1   hate_speech         24783 non-null  int64 
 2   offensive_language  24783 non-null  int64 
 3   neither             24783 non-null  int64 
 4   class               24783 non-null  int64 
 5   tweet               24783 non-null  object
dtypes: int64(5), object(1)
memory usage: 1.3+ MB
```

Рис.1. Данные первого датасета

	external_author_id	author	content	region	language	publish_date	harvested_date	fol	rs	updates	post_type	account_type	new_june_2018	retweet	account_category
0	2.385425e+09	MARRINABEREZKA	Обама принял решение по санкциям против Ирана ...	United States	Russian	11/11/2015 6:33	11/11/2015 6:34	26		4160	RETWEET	Russian	1	1	NonEnglish
1	2.534361e+09	ANETTANOVGOROD	Встреча Лаврова и Керри стартовала в Нью-Йорке...	Azerbaijan	Russian	9/27/2015 15:11	9/27/2015 15:11	16		1900	RETWEET	Russian	1	1	NonEnglish
2	1.612107e+09	LILJORDAMN	#IndieAdvancement Slim The Phenom @therealslim...	United States	English	12/3/2016 22:36	12/3/2016 22:36	60		2531	RETWEET	left	0	1	LeftTroll
3	3.254274e+09	FINDDIET	'@ozzycaceres ozzy @laurengodfreyx1 Lauren @dj...	United States	English	8/5/2015 17:39	8/5/2015 17:39	3		21960	NaN	Commercial	1	0	Commercial
4	1.647457e+09	COLINSNEVERLAND	This, BTW is why I don't instantly dismiss the...	United States	English	1/6/2016 18:02	1/6/2016 18:02	36		127	RETWEET	Right	0	1	RightTroll

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2973371 entries, 0 to 2973370
Data columns (total 15 columns):
 #   Column              Dtype  
---  ------              -----  
 0   external_author_id  float64
 1   author              object 
 2   content             object 
 3   region              object 
 4   language            object 
 5   publish_date        object 
 6   harvested_date      object 
 7   following           int64  
 8   followers           int64  
 9   updates             int64  
 10  post_type           object 
 11  account_type        object 
 12  new_june_2018       int64  
 13  retweet             int64  
 14  account_category    object 
dtypes: float64(1), int64(5), object(9)
memory usage: 340.3+ MB
```

Рис.2. Данные второго датасета

В данном решении автором было принято решение использовать в обоих задачах в качестве целевой переменной - категорию тональности текста (class и account_category), а в качестве признакового пространства текст твита на английском языке (tweet и content). Типичные примеры текстов обеих задач представлены на рисунке 3.

```
df_1.tweet.iloc[0]
```

```
"!!! RT @mayasolovely: As a woman you shouldn't complain about cleaning up your house. &amp; as a man you should always take th
e trash out..."
```

```
data_total[data_total['account_category'] == 'Commercial']['content'].iloc[0]
```

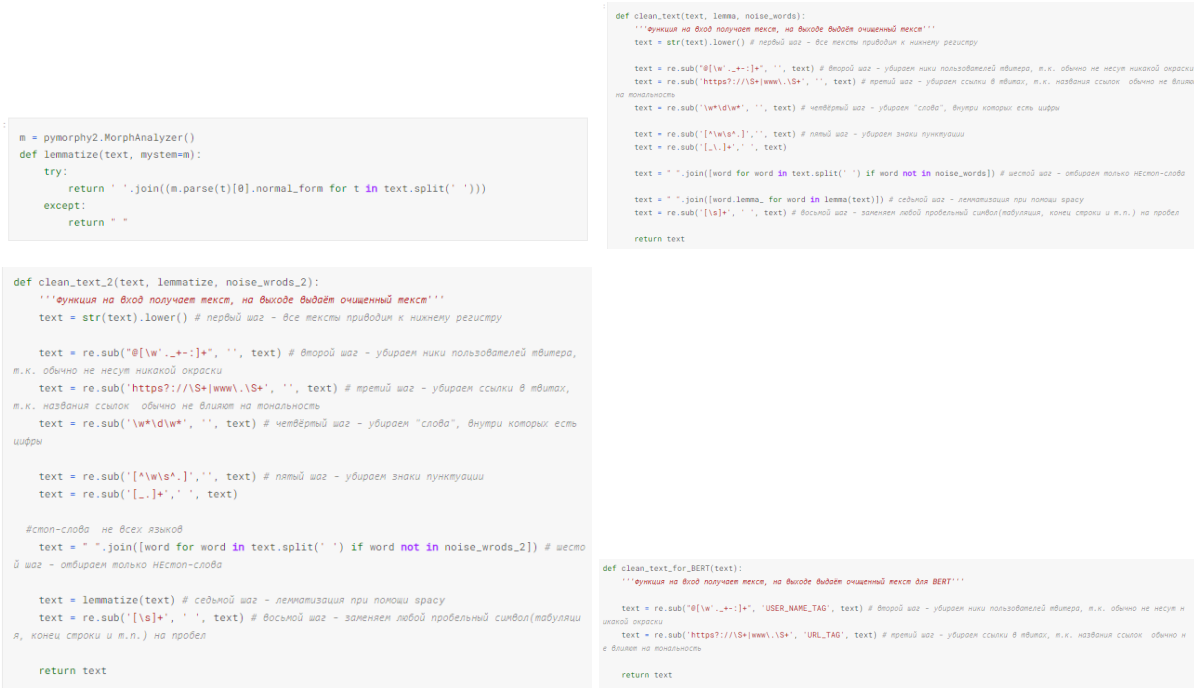
```
"'@ozzycaceres ozzy @laurengodfreyx1 Lauren @djhawes Danners @karleighwoelmer Karleigh h
ttp://t.co/QeKnVmkfxw https://t.co/ae3ItmkKin'"
```

Рис.3. Типичные примеры текстов решаемых задач

Типичные выявленные особенности текстов, которые необходимо было устранить или обработать представлены ниже:

1. Векторизация
2. Регистр
3. Ники пользователей
4. Нехорошие ссылки
5. Слова с цифрами
6. Знаки пунктуации
7. Стоп-слова
8. Лемматизация
9. Множественные пробелы
10. Дисбаланс классов
11. Разные языки (для второй задачи)
12. Категориальные переменные(для второй задачи)

Были написаны функции для подготовки и обработки текста к алгоритмам векторизации см. рис.4.



```
m = pymorphy2.MorphAnalyzer()
def lemmatize(text, mystem=m):
    try:
        return ' '.join((m.parse(t)[0].normal_form for t in text.split(' ')))
    except:
        return " "
```

```
def clean_text(text, lemma, noise_words):
    '''функция на вход получает текст, на выходе выдаёт очищенный текст'''
    text = str(text).lower() # первый шаг - все тексты приводим к нижнему регистру

    text = re.sub("@[\w'._+-:]+", '', text) # второй шаг - убираем ники пользователей твитера, т.к. обычно не несут никакой окраски
    text = re.sub('https?://\S+|www\.\S+', '', text) # третий шаг - убираем ссылки в твитах, т.к. названия ссылок  обычно не влияют
на тональность
    text = re.sub('\w*\d\w*', '', text) # четвёртый шаг - убираем "слова", внутри которых есть цифры

    text = re.sub('[^\w\s^.]','', text) # пятый шаг - убираем знаки пунктуации
    text = re.sub('[_.]+',' ', text)

    text = " ".join([word for word in text.split(' ') if word not in noise_words]) # шестой шаг - отбираем только нестоп-слова

    text = " ".join([word.lemma_ for word in lemma(text)]) # седьмой шаг - лемматизация при помощи spacy
    text = re.sub('[\s]+', ' ', text) # восьмой шаг - заменяем любой пробельный символ(табуляция, конец строки и т.п.) на пробел

    return text
```

```
def clean_text_2(text, lemmatize, noise_wrods_2):
    '''функция на вход получает текст, на выходе выдаёт очищенный текст'''
    text = str(text).lower() # первый шаг - все тексты приводим к нижнему регистру

    text = re.sub("@[\w'._+-:]+", '', text) # второй шаг - убираем ники пользователей твитера, т.к. обычно не несут никакой окраски
    text = re.sub('https?://\S+|www\.\S+', '', text) # третий шаг - убираем ссылки в твитах, т.к. названия ссылок  обычно не влияют на тональность
    text = re.sub('\w*\d\w*', '', text) # четвёртый шаг - убираем "слова", внутри которых есть цифры

    text = re.sub('[^\w\s^.]','', text) # пятый шаг - убираем знаки пунктуации
    text = re.sub('[_.]+',' ', text)

  #стоп-слова  не всех языков
    text = " ".join([word for word in text.split(' ') if word not in noise_wrods_2]) # шестой шаг - отбираем только нестоп-слова

    text = lemmatize(text) # седьмой шаг - лемматизация при помощи spacy
    text = re.sub('[\s]+', ' ', text) # восьмой шаг - заменяем любой пробельный символ(табуляция, конец строки и т.п.) на пробел

    return text
```

```
def clean_text_for_BERT(text):
    '''функция на вход получает текст, на выходе выдаёт очищенный текст для BERT'''

    text = re.sub("@[\w'._+-:]+", 'USER_NAME_TAG', text) # второй шаг - убираем ники пользователей твитера, т.к. обычно не несут никакой окраски
    text = re.sub('https?://\S+|www\.\S+', 'URL_TAG', text) # третий шаг - убираем ссылки в твитах, т.к. названия ссылок  обычно не влияют на тональность

    return text
```

Рис.4. Функции для решения некоторых из особенностей

Так например для функции clean_text из рис.4 осуществляется следующая поэтапная подготовка текста с приведёнными строками кода:

1. *text = str(text).lower()* # первый шаг - все тексты приводим к нижнему регистру
2. *text = re.sub("@[\w'._+-:]+", '', text)* # второй шаг - убираем ники пользователей твитера, т.к. обычно не несут никакой окраски
3. *text = re.sub('https?://\S+|www\.\S+', '', text)* # третий шаг - убираем ссылки в твитах, т.к. названия ссылок обычно не влияют на тональность
4. *text = re.sub('\w*\d\w*', '', text)* # четвёртый шаг - убираем "слова", внутри которых есть цифры
5. *text = re.sub('[^\w\s^.]','', text)* # пятый шаг - убираем знаки пунктуации

   text = re.sub('[_\.]+',' ', text)
6. *text = " ".join([word for word in text.split(' ') if word not in noise_words])* # шестой шаг - убираем стоп-слова
7. *text = " ".join([word.lemma_ for word in lemma(text)])* # седьмой шаг - лемматизация при помощи spacy
8. *text = re.sub('[\s]+', ' ', text)* # восьмой шаг - заменяем любой пробельный символ(табуляция, конец строки и т.п.) на пробел

В ходе исследования были использованы нижеперечисленные библиотеки:

Загрузка данных

- os

Анализ

- numpy
- pandas
- matplotlib.pyplot
- tqdm

Предобработка

- re
- string.punctuation
- nltk.corpus.stopwords
- spacy

- sklearn.feature_extraction.text.CountVectorizer
- sklearn.feature_extraction.text.TfidfVectorizer
- nltk.word_tokenize
- imblearn.over_sampling.SMOTE
- sklearn.preprocessing.LabelEncoder

Метрики

- sklearn.metrics.classification_report

Моделирование

- sklearn.model_selection.train_test_split
- sklearn.pipeline.Pipeline
- xgboost.XGBClassifier
- sklearn.linear_model.LogisticRegression
- catboost.CatBoostClassifier
- sklearn.svm.SVC
- transformers
- torch
- tensorflow.keras.models.Model
- tensorflow.keras.layers (LSTM, Activation, Dense, Dropout, Input, Embedding, SpatialDropout1D, Flatten)
- tensorflow.keras.optimizers.Adam
- tensorflow.keras.preprocessing.text.Tokenizer
- tensorflow.keras.preprocessing.sequence
- tensorflow.keras.utils.to_categorical
- tensorflow.keras.callbacks.EarlyStopping
- tensorflow.keras.models.Sequential
- tensorflow.keras.callbacks.EarlyStopping, ModelCheckpoint

Методика решения

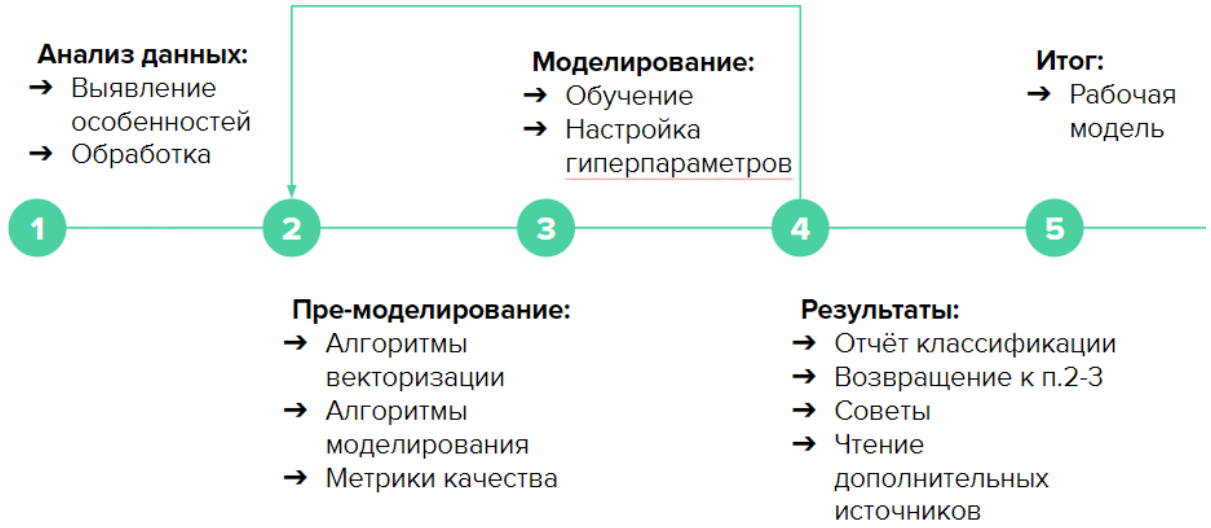


Рис.5. Ход решения

На рисунке 5 представлен ход решения при работе с задачами. Задачи решались несколькими способами. При этом выбор лучшего решения делался в пользу того, у которого при высоком качества, также было высокое качества у наименее представленного класса. В ходе разделения данных на тренировочные и тестовые было выбрано соотношение 80 и 20 %, соответственно. Надо отметить, что так как задача решалась несколькими способами, то здесь будет представлен самый лучший для каждой задачи по результатам метрик, однако в разделе результаты будут приведены метрики и других способов(Рис.6-9).

TfidfVectorizer + LogisticRegression

```
pipe = Pipeline([
                ('tf-idf', TfidfVectorizer()),
                ('LogReg', LogisticRegression(random_state=42,
                                              solver='liblinear',
                                              class_weight = 'balanced'))
                ])
pipe.fit(X_train, y_train)
y_pred = pipe.predict(X_test)
print(classification_report(y_pred, y_test))
```

```
              precision    recall  f1-score   support

           0       0.63      0.68      0.65       253
           1       0.83      0.90      0.87       724
           2       0.97      0.88      0.92       942

    accuracy                           0.86      1919
   macro avg       0.81      0.82      0.81      1919
weighted avg       0.87      0.86      0.86      1919
```

Рис.6. TfidfVectorizer и LogisticRegression

CountVectorizer + CatBoostClassifier

```
pipe7 = Pipeline([
                ('CountVectChar', CountVectorizer(ngram_range=(1, 1))),
                ('CBC', CatBoostClassifier(learning_rate=0.6, depth=4,
                                           loss_function='MultiClass'))
                ])
pipe7.fit(X_train, y_train)
y_pred7 = pipe7.predict(X_test)
print(classification_report(y_pred7, y_test))
```

```
              precision    recall  f1-score   support

           0       0.55      0.75      0.64       210
           1       0.89      0.89      0.89       809
           2       0.98      0.89      0.93       900

    accuracy                           0.88      1919
   macro avg       0.81      0.84      0.82      1919
weighted avg       0.89      0.88      0.88      1919
```

Рис.7. CountVectorizer и CatBoostClassifier

CountVectorizer + LogisticRegression

```
vec_2 = CountVectorizer(ngram_range=(1, 1))
vec_2.fit(data_total_4_English['content'].values.astype('U'))
bow_2 = vec_2.transform(X_train_E_2)
```

```
clf_2_1 = LogisticRegression(random_state=42, solver='liblinear',
                             class_weight = 'balanced')
clf_2_1.fit(bow_2, y_train_E_2)
pred_2_1 = clf_2_1.predict(vec_2.transform(X_test_E_2))
print(classification_report(pred_2_1, y_test_E_2))
```

```
              precision    recall  f1-score   support

           0       0.95      0.94      0.94     10628
           1       0.68      0.28      0.40      2544
           2       0.83      0.80      0.81     22712
           3       0.67      0.69      0.68     38740
           4       0.88      0.86      0.87     57489
           5       0.59      0.29      0.39      5420
           6       0.72      0.83      0.77     58206
           7       0.25      0.04      0.07      4261

    accuracy                           0.78    200000
   macro avg       0.69      0.59      0.62    200000
weighted avg       0.77      0.78      0.77    200000
```

Рис.8. CountVectorizer и LogisticRegression

SMOTE + CountVectorizer + LogisticRegression (300 000)

```
X_train_E_2, X_test_E_2, y_train_E_2, y_test_E_2 = train_test_split(data_total_4_English['content'].values.astype('U'),
                                                                    data_total_4_English['account_category'] ,
                                                                    test_size = 0.2)
```

```
vec_10_E = CountVectorizer( ngram_range=(1, 1))
vec_10_E.fit(data_total_4_English['content'].values.astype('U'))
bow_10_E = vec_10_E.transform(X_train_E_2)
```

```
sm = SMOTE (#sampling_strategy = 0.9,
        random_state=0,
        k_neighbors=25)
X_train_res_E, y_train_res_E = sm.fit_resample(bow_10_E, y_train_E_2)
```

```
print('\t\tДО балансировки \tПОСЛЕ балансировки ')
print('y :\t\t{}\t\t{}'.format(y_train_E_2.shape, y_train_res_E.shape))
print('X :\t\t{}\t\t{}'.format(bow_10_E.shape, X_train_res_E.shape))
```

```
                ДО балансировки         ПОСЛЕ балансировки
y :             (800000,)               (2137584,)
X :             (800000, 212850)                (2137584, 212850)
```

```
model_8_1 = LogisticRegression(random_state=42, solver='liblinear')
model_8_1.fit(X_train_res_E, y_train_res_E)
pred_8_1 = model_8_1.predict(vec_10_E.transform(X_test_E_2))
print(classification_report(pred_8_1, y_test_E_2))
```

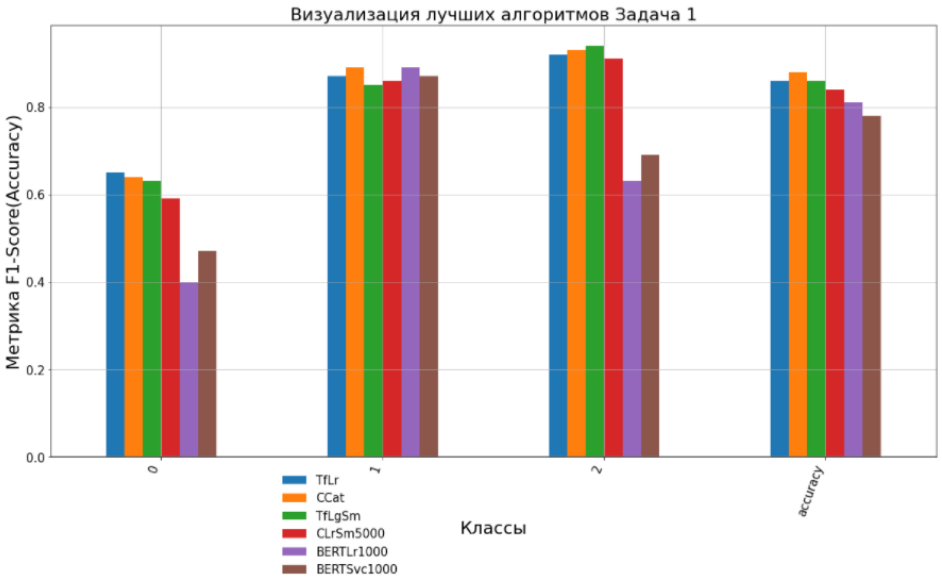
```
              precision    recall  f1-score   support

           0       0.93      0.94      0.94     10575
           1       0.58      0.33      0.42      1768
           2       0.83      0.81      0.82     22708
           3       0.66      0.69      0.68     38335
           4       0.89      0.85      0.87     58710
           5       0.45      0.29      0.35      4008
           6       0.74      0.82      0.77     60677
           7       0.14      0.03      0.05      3219

    accuracy                           0.78    200000
   macro avg       0.65      0.60      0.61    200000
weighted avg       0.77      0.78      0.78    200000
```

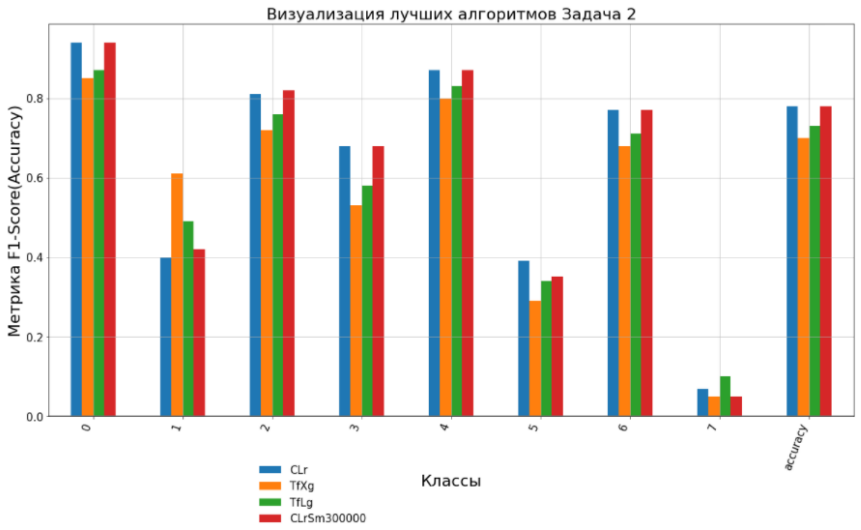
Рис.9. SMOTE с CountVectorizer и LogisticRegression

Результаты



	TfLr	CCat	TfLgSm	CLrSm5000	BERTLr1000	BERTSvc1000
0	0.65	0.64	0.63	0.59	0.40	0.47
1	0.87	0.89	0.85	0.86	0.89	0.87
2	0.92	0.93	0.94	0.91	0.63	0.69
accuracy	0.86	0.88	0.86	0.84	0.81	0.78

Рис.10. Лучшие результаты метрик качества по Задаче 1



	CLr	TfXg	TfLg	CLrSm300000
0	0.94	0.85	0.87	0.94
1	0.40	0.61	0.49	0.42
2	0.81	0.72	0.76	0.82
3	0.68	0.53	0.58	0.68
4	0.87	0.80	0.83	0.87
5	0.39	0.29	0.34	0.35
6	0.77	0.68	0.71	0.77
7	0.07	0.05	0.10	0.05
accuracy	0.78	0.70	0.73	0.78

Рис.11. Лучшие результаты метрик качества по Задаче 2

Таким образом, хорошо себя показал логистическая регрессия совместно алгоритмами векторизации, дав по качеству для первой задачи по метрике f1-score тройку и accuracy: 0 - 65%, 1 - 87%, 2 - 92% и 86%, для второй задачи - по метрике f1-score тройку и accuracy: 0 - 94%, 1 - 40%, 2 - 81%, 3 - 68%, 4 - 87%, 5 - 39%, 6 - 77%, 7 - 7% и 78%. Однако по первой задачи всё же лучше показал

себя градиентный бустинг CatBoostClassifier с значениями по метрике f1-score тройку и accuracy: 0 - 64%, 1 - 89%, 2 - 93% и 88%

Надо заметить, что возможно качество получилось не столь большим из-за неучтённых стоп-слов, слов неологизмов, разговорных слов и слов со специально сделанными опечатками и т.п.

Полученные модели подходят для внедрения в социальных сетях для выявления негативных текстов/комментариев/сообщений, а также для обучения/ознакомления основным методам обработки естественного языка. Однако модели получились достаточно низкого качества, поэтому их реальное внедрение находится под сомнением, т.к существуют модели, которые дают качества более 92%.

Выводы и заключение

1. Лучше всего для задач определения тональности текста подходит алгоритм:
   a. LogisticRegression
2. Ансамблевые модели показывают не лучшие результаты на большом количестве данных, чем логистическая регрессия.
3. В дальнейшем, повысить качество можно :
   a. путём использованием всех столбцов данных
   b. настройкой гиперпараметров при помощи GridSearchCV, RandomizedSearchCV
   c. запуском обучения, используя большее количество ресурсов

Список источников

1. http://neerc.ifmo.ru/wiki/
2. http://neerc.ifmo.ru/wiki/index.php
3. https://proglib.io/p/analiz-tonalnosti-teksta-proshloe-nastoyashchee-i-budushchee-2020-11-30
4. https://webiomed.ru/blog/osnovnye-metriki-zadach-klassifikatsii-v-mashinnom-obuchenii/
5. https://ml-handbook.ru/chapters/model_evaluation/intro#%D0%BC%D0%BD%D0%BE%D0%B3%D0%BE%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%BE%D0%B2%D0%B0%D1%8F-%D0%BA%D0%BB%D0%B0%D1%81%D1%81%D0%B8%D1%84%D0%B8%D0%BA%D0%B0%D1%86%D0%B8%D1%8F
6. https://nauchkor.ru/pubs/analiz-tonalnosti-tekstov-s-ispolzovaniem-neyrosetevyh-modeley-587d36545f1be77c40d58cc1
7. http://ceur-ws.org/Vol-2233/Paper_8.pdf
8. https://dspace.spbu.ru/bitstream/11701/25732/1/diplom.pdf