Итоговое задание

Ольга Дейкина
Группа DSN-CP-2

Классификация русскоязычных спортивных текстов

1

Описание проекта

Исходная задача:

Контекст:
На основе заданного набора постов определите, какой вид спорта обсуждается в выбранном сообществе.
Список доступных категорий:

- легкая атлетика;
- автоспорт;
- баскетбол;
- настольные игры;
- киберспорт;
- крайний;
- футбол;
- хоккей;
- боевые искусства;
- мотоспорт;
- большой теннис;
- волейбол;
- зимний вид спорта.

Данные: https://www.kaggle.com/datasets/mikhailma/russian-social-media-text-classification

Описание проекта

Актуальность задачи, ее место в предметной области:

Проект по созданию мультиклассового классификатора текстовых документов. В контексте социальных сетей эта задача безусловно является актуальной. Например, результаты работы классификатора можно использовать для настройки рекламы в сообществах социальных сетей.

Целевая метрика:

оценочная метрика выглядит так:

```
def score(true, pred, n_samples):
    counter = 0
    if true == pred:
        counter += 1
    else:
        counter -= 1
    return counter / n_samples
```

Итоги обучения

4

Описание итоговой модели

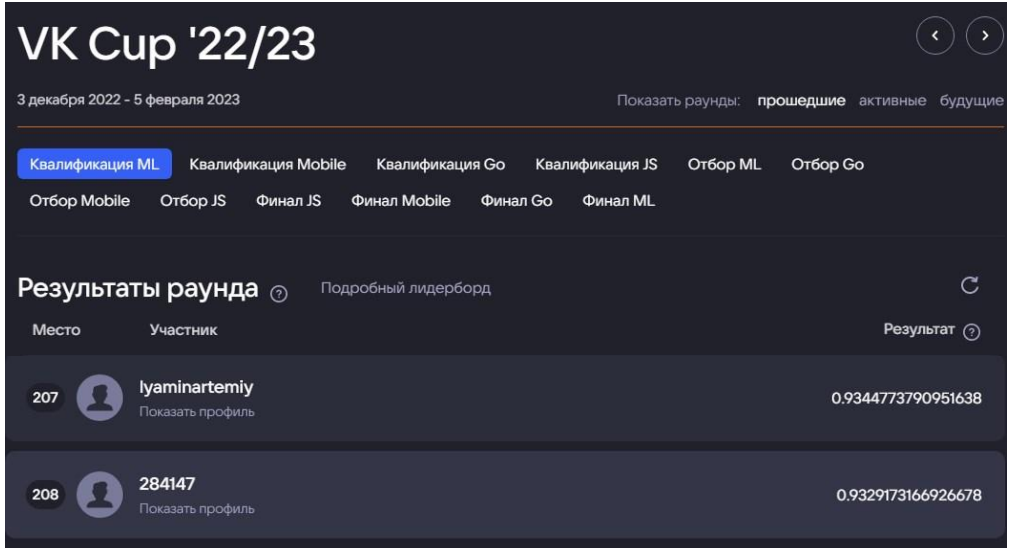
Целевая метрика итоговой модели составила:

Считаем долю правильных ответов (количество значений True поделить на количество объектов):

```
df_my_concat['category_right'].sum() / df_my_concat.shape[0]
```

```
0.9329779131759329
```

Сравнение качества итоговой модели с другими решениями:



Анализ данных

2

Анализ данных

1. **Источник данных**

   В качестве исходных данных даны 3 файла:
   - sample_submission.csv (эталон финального датасета)
   - train.csv (набор данных для обучения)
   - test.csv (тестовый набор данных)

2. **Тип данных**

   Данные представлены целыми числами (int64) и текстом (object).

3. **Описание своими словами**

   В наборе данных для обучения находятся размеченные по категориям посты спортивных сообществ. В тестовом наборе данных разметки нет. В финальном датасете не должно быть текста постов, только идентификатор сообщества и его тематика (категория). Финальный датасет нужно сравнить с эталоном и получить долю правильных ответов.

4. **Размер датасета**

   - sample_submission.csv - 2626 строк, 2 столбца
   - train.csv - 38740 строк, 3 столбца
   - test.csv - 26260 строк, 2 столбца

```
print(df.shape)
print(df_train.shape)
print(df_test.shape)
```

```
(2626, 2)
(38740, 3)
(26260, 2)
```

5. Результаты разведочного анализа данных

Проверка наличия пропусков в данных методом .info()

```
[ ]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2626 entries, 0 to 2625
Data columns (total 2 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   oid       2626 non-null   int64
 1   category  2626 non-null   object
dtypes: int64(1), object(1)
memory usage: 41.2+ KB
```

```
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38740 entries, 0 to 38739
Data columns (total 3 columns):
 #   Column    Non-Null Count  Dtype
---  ------    --------------  -----
 0   oid       38740 non-null  int64
 1   category  38740 non-null  object
 2   text      38740 non-null  object
dtypes: int64(1), object(2)
memory usage: 908.1+ KB
```

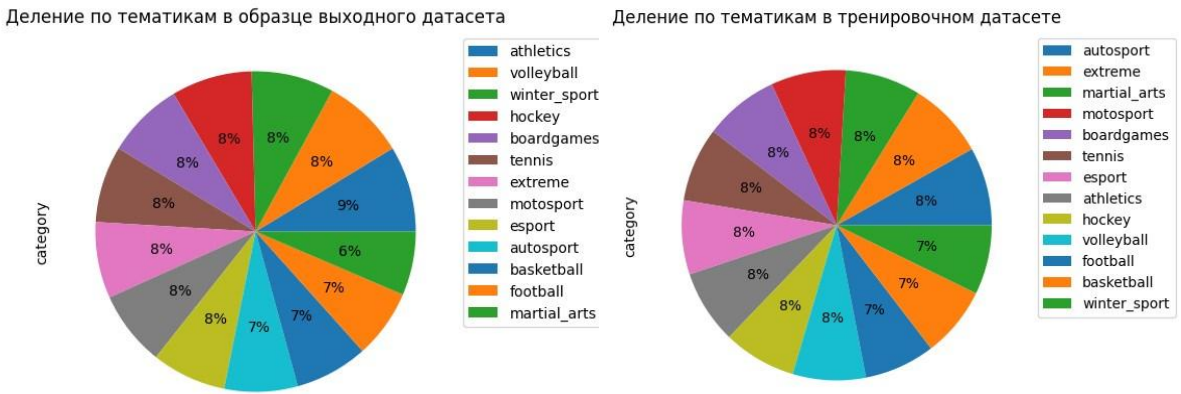
```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26260 entries, 0 to 26259
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   oid     26260 non-null  int64
 1   text    26260 non-null  object
dtypes: int64(1), object(1)
memory usage: 410.4+ KB
```

Пропусков в исходных данных нет.

Распределение категорий:

Данные сбалансированы



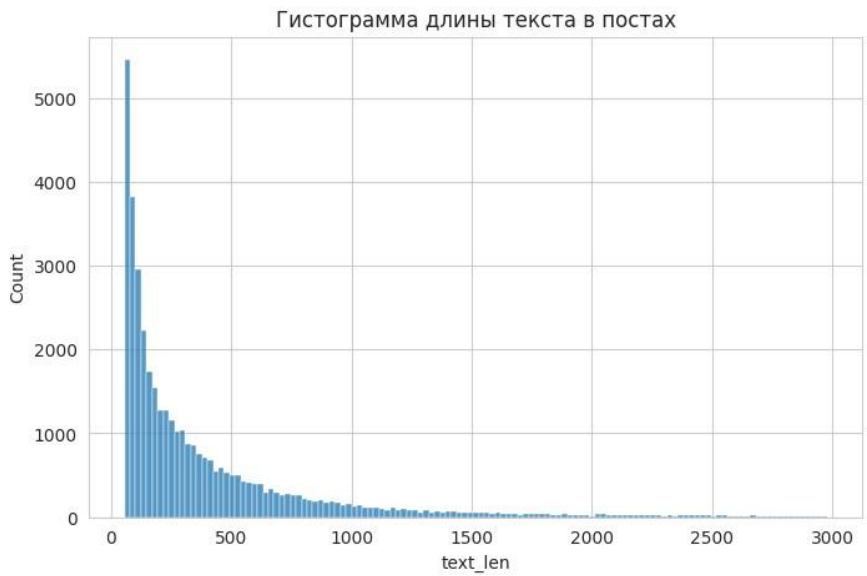
Анализ по сообществам

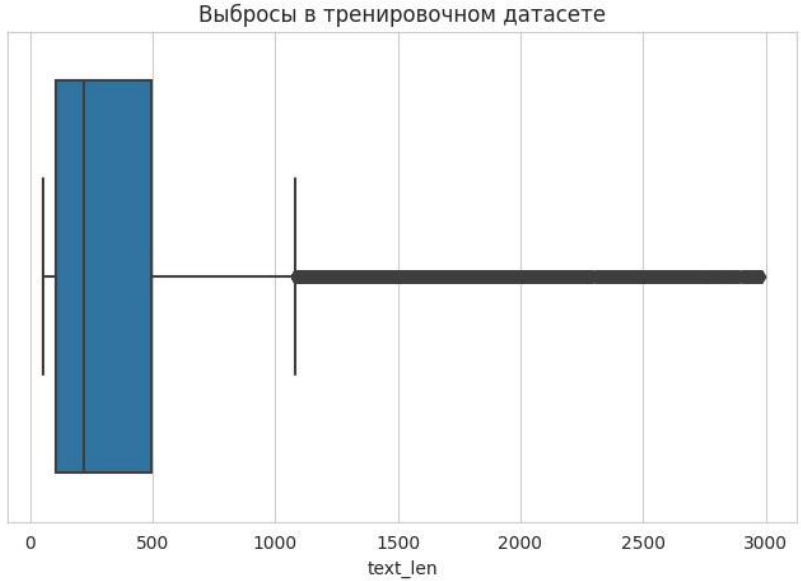
В ‘train’ содержится 3874 уникальных сообществ, на них приходится 38740 текстов, по 10 на каждое сообщество. Из общего количества текстов: 35744 - уникальные, 2966 - повторы.

В ‘test’ содержится 2626 уникальных сообществ, на них приходится 26260 текстов, по 10 на каждое сообщество. Из общего количества текстов: 24833 - уникальные, 1427 - повторы.

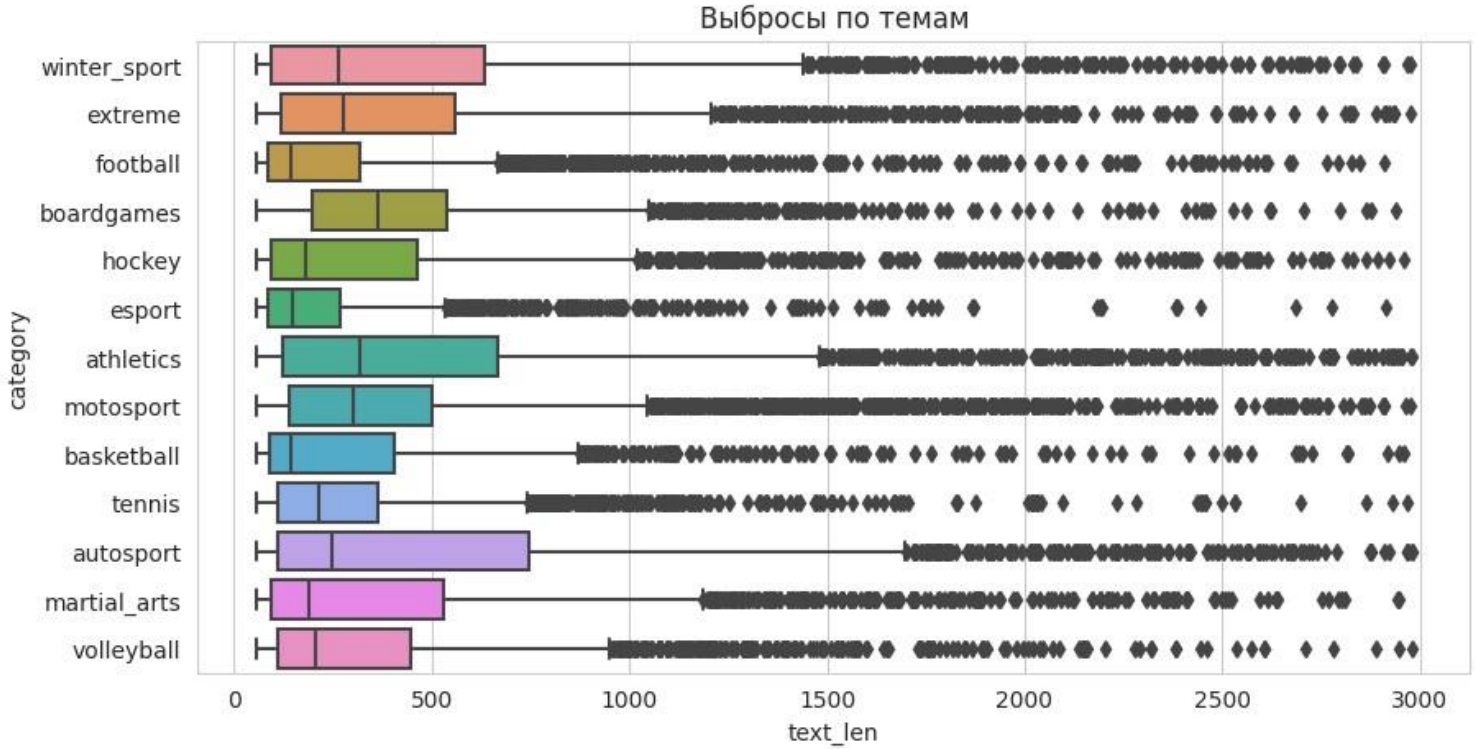
В ‘sample_submission’ содержится 2626 уникальных сообществ.

Проверка наличия выбросов в train



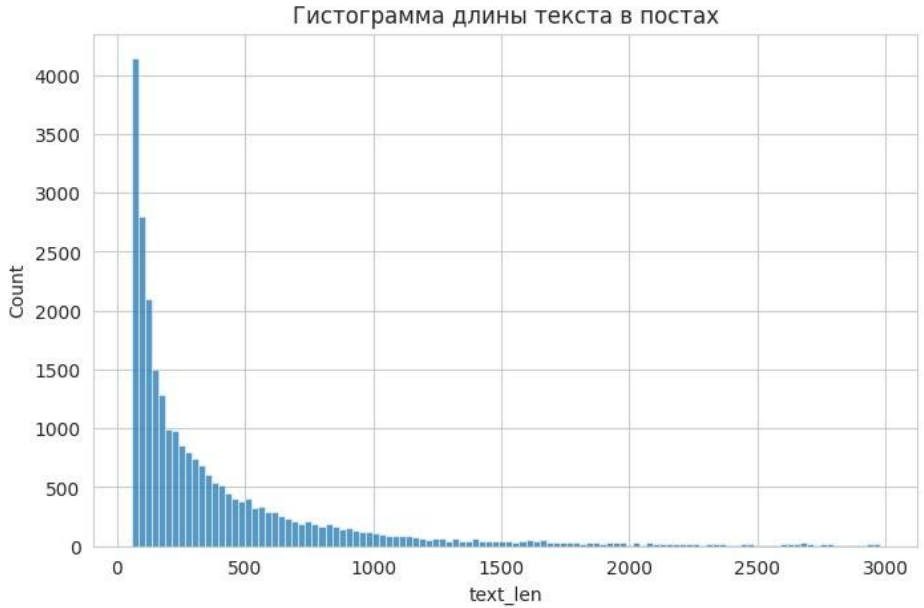


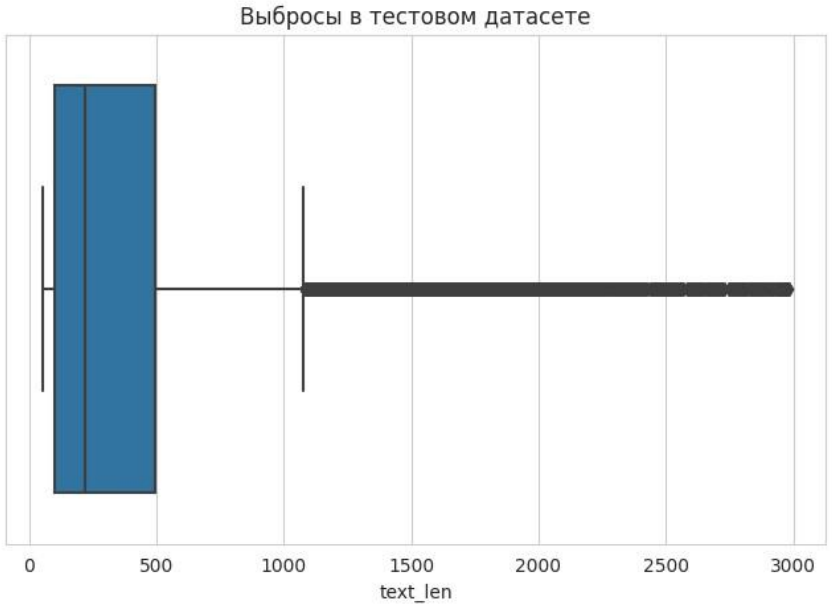
Проверка наличия выбросов в train



Посты на тему футбола, киберспорта и тенниса короче постов остальных тем. Много выбросов по всем темам.

Проверка наличия выбросов в test





В тестовом датасете также много выбросов.

Проверка наличия дубликатов в train

В train найдено 2966 дубликатов текста, что составляет около 8% от всего объема датасета.

```
# смотрим пример повторов
df_train[df_train.text=='Шахматы и нарды ручной работы с любым рисунком и фото 33
```

	oid	category	text
9835	886977381	martial_arts	Шахматы и нарды ручной работы с любым рисунком...
11887	884148780	football	Шахматы и нарды ручной работы с любым рисунком...
23724	410518766	hockey	Шахматы и нарды ручной работы с любым рисунком...
31019	643475045	autosport	Шахматы и нарды ручной работы с любым рисунком...
35563	442208575	basketball	Шахматы и нарды ручной работы с любым рисунком...

```
# смотрим пример повторов
df_train[df_train.text=='Для тех кто хочет побеждать 33 Четыре ключа к твоим победам Ж
```

	oid	category	text
6343	548764343	athletics	Для тех кто хочет побеждать 33 Четыре ключа к ...
33345	646711387	extreme	Для тех кто хочет побеждать 33 Четыре ключа к ...

Можно сделать вывод, что повторяющиеся посты - это реклама и спам. Одинаковые тексты отправляются от разных пользователей с указанием разной тематики, не соответствующей содержанию текста.

Проверка наличия дубликатов в test

В тестовом датасете найдено 1427 дубликатов текста, что составляет около 5% от всего объема датасета. Анализ показал, что повторяющиеся посты - это реклама и спам. Одинаковые тексты отправляются от разных пользователей.

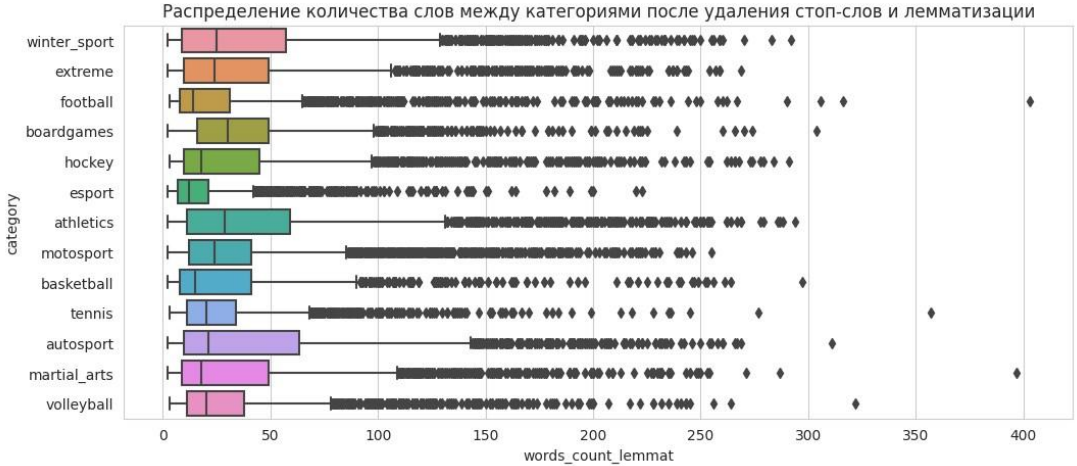
```
# смотрим пример повторов
df_test[df_test.text=='За кроссовками в Баскетбольный магазин Ghetto Basket Shop Консультация tokenoidtokenoid Стрельски
```

	oid	text
407	595894929	За кроссовками в Баскетбольный магазин Ghetto ...
512	162043401	За кроссовками в Баскетбольный магазин Ghetto ...
888	148389602	За кроссовками в Баскетбольный магазин Ghetto ...
1141	595894929	За кроссовками в Баскетбольный магазин Ghetto ...
1198	227716111	За кроссовками в Баскетбольный магазин Ghetto ...
...	...	...
25084	959923306	За кроссовками в Баскетбольный магазин Ghetto ...
25343	588021734	За кроссовками в Баскетбольный магазин Ghetto ...
25402	410422939	За кроссовками в Баскетбольный магазин Ghetto ...
25612	194289811	За кроссовками в Баскетбольный магазин Ghetto ...
25619	622543412	За кроссовками в Баскетбольный магазин Ghetto ...

105 rows × 2 columns

6. Основные шаги предобработки данных, их результаты:

- удалены дубликаты;
- проведена очистка, токенизация и лемматизация текста;
- выбросы не удалены, поскольку это не дало положительных результатов в эксперименте, при этом объем потерянных данных мог составить 22% (для обучающей выборки).



Методика реализации

3

Алгоритм реализации задачи

1. Разработка базовой модели - Baseline.
2. Разработка модели с применением алгоритмов машинного обучения на базе Scikit-Learn.
3. Разработка модели нейронной сети на базе PyTorch.
4. Выбор итоговой модели.
5. Обучение итоговой модели на полных данных train.
6. Получение оценочной метрики.

Baseline

Базовая модель строилась на “грязных” данных столбца ‘text’ из train после удаления дубликатов. Категориальные метки преобразованы с использованием LabelEncoder.

Первым этапом было разделение данных train на обучающую и валидационную выборки методом train_test_split (в соотношении 75 на 25%):

```
y = pd.Series (le.transform(df_train_1.category))
```

```
[45] y.head(13)
```

```
0     12
1      5
2      6
3      3
4      7
5      3
6      7
7      4
8      0
9      9
10     2
11    10
12     9
dtype: int64
```

```
from sklearn.model_selection import train_test_split
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.25, random_state=42)
```

```
print(df_train_1.shape[0])
print(X_train.shape)
print(X_val.shape)
```

```
35774
(26830,)
(8944,)
```

Baseline выбиралась методом классификации необработанных n-грамм.

Использовалось три варианта векторизации текста: униграммы, биграммы и триграммы.

В качестве векторизаторов применялись CountVectorizer и TfidfVectorizer.

В качестве классификатора применялась модель LogisticRegression.

Поскольку данные сбалансированы, в качестве метрики была выбрана accuracy - доля объектов, для которых мы правильно предсказали класс.

Лучший результат показала модель с векторизатором TfidfVectorizer на униграммах.

	accuracy
униграммы CountVectorizer	0.82
биграммы CountVectorizer	0.70
триграммы CountVectorizer	0.52
униграммы TfidfVectorizer	0.83
биграммы TfidfVectorizer	0.73
триграммы TfidfVectorizer	0.58

	precision	recall	f1-score	support
athletics	0.81	0.94	0.87	550
autosport	0.85	0.90	0.87	739
basketball	0.78	0.96	0.86	497
boardgames	0.90	0.91	0.91	643
esport	0.78	0.74	0.76	715
extreme	0.79	0.57	0.66	953
football	0.79	0.75	0.77	731
hockey	0.85	0.87	0.86	697
martial_arts	0.77	0.75	0.76	718
motosport	0.85	0.91	0.88	647
tennis	0.94	0.97	0.95	718
volleyball	0.81	0.89	0.85	648
winter_sport	0.86	0.82	0.84	688
accuracy			0.83	8944
macro avg	0.83	0.84	0.83	8944
weighted avg	0.83	0.83	0.83	8944

Модель на методах sklearn

Первым этапом была проведена предобработка данных: очистка, токенизация и лемматизация текста.
В качестве признаков использовались обработанные данные нового столбца ‘lemma’.
Данные разделены на обучающую и валидационную выборки методом train_test_split (в соотношении 75 на 25%).
В результате предобработки данных метрика улучшилась на 1%.

	accuracy
Baseline	0.83
Baseline + обработка	0.84

Вторым этапом было проведено удаление выбросов: Объём выбросов составил 22% от всего объема данных. В результате данные стали несбалансированы. В модель был добавлен параметр class_weight='balanced'
В результате удаления выбросов метрика ухудшилась на 4%.

	accuracy
Baseline	0.83
Baseline + обработка	0.84
Baseline + обработка + удаление выбросов	0.80

Удаление выбросов нецелесообразно, оставляем предыдущий вариант модели.

Третьим этапом было проведено добавление признаков: В качестве дополнительных признаков использовались данные столбцов 'oid', 'words_count'.
В результате метрика улучшилась на 2%.

	accuracy
Baseline	0.83
Baseline + обработка	0.84
Baseline + обработка + удаление выбросов	0.80
Baseline + обработка + признаки	0.86

Четвёртым этапом было проведено добавление признака с извлечением имён с использованием библиотеки для извлечения именованных сущностей “Natasha”.

	accuracy
Baseline	0.83
Baseline + обработка	0.84
Baseline + обработка + удаление выбросов	0.80
Baseline + обработка + признаки	0.86
Baseline + обработка + признаки + имена	0.94

В результате метрика улучшилась на 8%.

Пятым этапом была проведена оптимизация гиперпараметров классификатора модели с использованием GridSearchCV.

```
Оптимальные параметры для grid:
0.927879142013715 {'clf__class_weight': 'balanced', 'clf__max_iter': 200, 'clf__multi_class': 'ovr', 'clf__penalty': None, 'clf__solver': 'sag'}
```

	accuracy
Baseline	0.83
Baseline + обработка	0.84
Baseline + обработка + удаление выбросов	0.80
Baseline + обработка + признаки	0.86
Baseline + обработка + признаки + имена	0.94
Baseline + обработка + признаки + имена + оптимизация	0.94

Шестым этапом был проведен выбор оптимального классификатора.

	accuracy
LogisticRegression	0.94
SVC	0.71
RandomForestClassifier	0.78
MLPClassifier	0.85
CatBoostClassifier	0.86
XGBClassifier	0.88

Лучший результат показала модель с классификатором LogisticRegression.

Модель на методах PyTorch

Токенизируем текст, признак - предобработанные данные столбца ‘lemma’. Векторизуем с использованием Word2vec эмбедингов. Преобразовываем признаки в тензоры PyTorch. Разбиваем данные на обучающую и валидационную выборки методом train_test_split (в соотношении 75 на 25%).
Задаём нейросеть со следующими параметрами:

```
MulticlassClassificationWithNonLinearTransformations(
  (linear_layer_stack): Sequential(
    (0): Linear(in_features=300, out_features=100, bias=True)
    (1): ReLU()
    (2): Linear(in_features=100, out_features=100, bias=True)
    (3): ReLU()
    (4): Linear(in_features=100, out_features=13, bias=True)
  )
```

В качестве оптимизатора применялись SGD и Rectified Adam PyTorch. Лучший результат показал оптимизатор SGD:

```
Epoch: 0 | Loss: 2.89331, Acc: 5.60% | Test Loss: 2.37537, Test Acc: 28.31%
Epoch: 50 | Loss: 1.09699, Acc: 68.64% | Test Loss: 1.09445, Test Acc: 67.36%
Epoch: 100 | Loss: 0.97889, Acc: 71.24% | Test Loss: 0.96021, Test Acc: 72.33%
Epoch: 150 | Loss: 0.86786, Acc: 74.52% | Test Loss: 0.91021, Test Acc: 73.36%
Epoch: 200 | Loss: 0.79690, Acc: 76.48% | Test Loss: 0.87681, Test Acc: 74.59%
Epoch: 250 | Loss: 0.76253, Acc: 77.39% | Test Loss: 0.85366, Test Acc: 75.12%
Epoch: 300 | Loss: 0.74574, Acc: 77.74% | Test Loss: 0.85503, Test Acc: 75.11%
Epoch: 350 | Loss: 0.84256, Acc: 75.14% | Test Loss: 0.91125, Test Acc: 73.33%
Epoch: 400 | Loss: 0.71239, Acc: 78.30% | Test Loss: 0.86565, Test Acc: 74.90%
Epoch: 450 | Loss: 0.68660, Acc: 78.84% | Test Loss: 0.85595, Test Acc: 75.41%
Epoch: 500 | Loss: 0.64562, Acc: 80.70% | Test Loss: 0.81744, Test Acc: 76.29%
Epoch: 550 | Loss: 0.65519, Acc: 80.08% | Test Loss: 0.84147, Test Acc: 75.87%
Epoch: 600 | Loss: 0.62042, Acc: 81.09% | Test Loss: 0.81572, Test Acc: 76.35%
Epoch: 650 | Loss: 0.62464, Acc: 80.85% | Test Loss: 0.82025, Test Acc: 76.30%
Epoch: 700 | Loss: 0.64470, Acc: 80.09% | Test Loss: 0.85059, Test Acc: 75.64%
Epoch: 750 | Loss: 0.56949, Acc: 82.82% | Test Loss: 0.81773, Test Acc: 76.64%
Epoch: 800 | Loss: 0.55186, Acc: 83.32% | Test Loss: 0.81159, Test Acc: 76.73%
Epoch: 850 | Loss: 0.56387, Acc: 82.86% | Test Loss: 0.83331, Test Acc: 76.47%
Epoch: 900 | Loss: 0.54214, Acc: 83.48% | Test Loss: 0.82917, Test Acc: 76.83%
Epoch: 950 | Loss: 0.58680, Acc: 81.72% | Test Loss: 0.99026, Test Acc: 73.75%
```

Выбор итоговой модели

Лучше всего себя показала модель с применением алгоритмов машинного обучения на базе Scikit-Learn:

- проведена предобработка данных и feature engineering;
- применен векторизатор TfidfVectorizer;
- применен классификатор LogisticRegression с оптимизацией гиперпараметров.

	accuracy
Методы sklearn	0.94
Методы PyTorch	0.77

Выводы

5

1. Лучшая модель на базе `TfidfVectorizer` и `LogisticRegression`
2. Удачные эксперименты с извлечением именованных сущностей
3. Неудачные эксперименты с удалением выбросов
4. У некоторых участников с высокими результатами было применен микс из нескольких моделей, с дальнейшим получением построчно среднего значения предсказания. Этот вариант можно попробовать.

Использованные источники

https://github.com/natasha/natasha
https://www.learnpytorch.io/02_pytorch_classification/
https://radimrehurek.com/gensim/models/word2vec.html
https://habr.com/ru/articles/498144/
https://neurohive.io/ru/tutorial/bert-klassifikacya-teksta/
https://habr.com/ru/articles/669674/
https://huggingface.co/cointegrated/rubert-tiny
https://huggingface.co/cointegrated/rubert-tiny2
https://github.com/shitkov/bert4classification
https://github.com/Lojaleto/VK_Cup_2022_quali/blob/main/VK_Cup_2022_quali.ipynb

Спасибо за внимание!