Определение тематики статьи на habr.ru

Годовой проект МОВС23
Чекпоинт 5

Карнакова Ксения
Маханова Наталья
Гужвина Любовь

ML задачи

Было сделано до чекпоинта 5:
Предсказание **рейтинга статьи**
Сделано к чекпоинту 5:
Предсказание **хабов** по текстам публикаций

Предсказание рейтинга статьи

Результаты работы над этой задачей в ноутбуке **ML.ipynb**

целевая переменная: рейтинг статьи (votes)

признаки:

- числовые: количество закладок (bookmarks), количество комментариев (comments_count), количество просмотров (views), время прочтения статьи (reading_time), рейтинг автора (rating), карма автора (karma), количество голосов за карму автора (karma_votes), год (year), месяц (month), день (day)
- категориальные: день недели (weekday), время суток публикации (time_category), корпоративность публикации (is_corporative), тип публикации (posttype)

- **метрики качества:** R^2, MAE

Предсказание рейтинга статьи

Из даты публикации получены признаки: год, месяц, день, день недели, время суток (утро, день, вечер, ночь)

Для категориальных признаков применялось кодирование **OneHotEncoder**

Для числовых признаков проводилась стандартизация при помощи **StandardScaler**

Проводились эксперименты с подбором лучших гиперпараметров при помощи **GridSearchCV** для следующих моделей: линейная регрессия (**LinearRegression, Lasso, ElasticNet, Ridge**), деревья решений (**DecisionTreeRegressor**), случайный лес (**RandomForestRegressor**)

Для некоторых числовых признаков дополнительно применялась логарифморование/полиномизация 2 степени

Метрики моделей после подбора гиперпараметров

	R^2 train	MAE train	R^2 test	MAE test
Linear Regression	0.40	10.92	0.41	10.89
Lasso	0.40	10.91	0.41	10.88
Ridge	0.40	10.91	0.41	10.88
Elastic net	0.40	10.91	0.41	10.88
Decision Tree Regressor	0.47	9.88	0.46	9.97
Random Forest	0.61	8.23	0.64	8.03

Предсказание рейтинга статьи

Мы столкнулись с проблемой предсказания рейтинга для еще не размещенных на сайте публикаций, так как использовали в моделях признаки, которые есть только в опубликованных на сайте статьях (количество просмотров, количество закладок, количество комментариев и т. д.).

В связи с этим, было решено отказаться от использования таких числовых признаков и применять в дальнейшем только признаки, которые существуют еще до публикации статьи на Хабре, например, **тексты статей**.

Предсказание хабов по текстам публикаций: предобработка

Предобработка текстов приведена в EDA/**preprocessing_data.ipynb**

Функции для предобработки текста были занесены в файл EDA/**functions.py**

В спарсенных данных с Хабра в столбце `text` помимо текста статьи содержатся теги (например img, a, p, code) и разделители (\n, \t, \xa0 и т.д.): мы очистили тексты от лишних символов.

Помимо этого, в текстах содержатся различные грамматические формы одного и того же слова. Для приведения слов к одной форме мы проводили **лемматизацию** при помощи библиотеки от Яндекса - **pymystem3**. Для удаления стоп-слов использовалась библиотека **nltk**.

Обработанные тексты статей (после удаления стоп-слов, лишних символов и лемматизации) записаны в новый столбец `lemmatized_text`.

Предсказание хабов по текстам публикаций: метрики

- **F1-micro**
- **F1-macro**
- **F1-weighted**
- **accuracy**

Разница между метриками F1-macro и F1-weighted будет показывать, насколько хорошо распознаются хабы с небольшим количеством объектов

Предсказание хабов по текстам публикаций: предобработка

Дальнейшая работа приведена в ML/**ML_hubs_prediction.ipynb**

Произведена векторизация текста **tf-idf** по словам с разными параметрами

- min_df=20, max_df=1; длина получившегося вектора: 50606
- min_df=12, max_df=0.95; длина получившегося вектора: 67809

с целью оценить как векторизация влияет на качество модели

Предсказание хабов по текстам публикаций: таргет

На Хабре хабы делятся на профильные (на сайте отмечены *) и непрофильные (на сайте без *).

В качестве таргета было решено применять **профильные хабы**, так как в непрофильных содержится корпоративные хабы, относящиеся к деятельности компаний (gemaltorussia, okmeter и т.д.): в нашей задаче их применение не имеет смысла.

Из собранного датасета были удалены публикации, в которых не оказалось профильных хабов.

Самое большое количество упомянутых профильных хабов в одной публикации − **пять**.

Чаще всего в одной публикации указывают **два** профильных хаба.

Исходя из предположения, что первый из указанных хабов − наиболее точно описывает тематику статьи, оставим **в качестве таргета первый профильный хаб** и перейдем к задаче многоклассовой классификации.

Предсказание хабов по текстам публикаций: multi-class classification

Тексты после **TF-IDF** с параметрами min_df=20, max_df=1

Модель	Гиперпараметры	test F1-micro (accuracy)	test F1-macro	test F1-weighted
Logistic Regression	solver='lbfgs' multi_class='multinomial' C=1	0.48	0.16	0.44
Logistic Regression	solver='lbfgs' multi_class='multinomial' C=2	0.49	0.19	0.46
LinearSVC	class_weight='balanced',C=0.1, multi_class='ovr'	0.43	0.26	0.44
LinearSVC	подобраны с помощью GridSearchCV: class_weight=None, C=0.1, multi_class='ovr'	0.50	0.20	0.46
MultinomialNB	подобраны с помощью GridSearchCV: 'alpha': 0.001	0.44	0.22	0.42

В **логистической регрессии**, в целом, крупные хабы предсказываются хорошо благодаря большому количеству наблюдений и ключевым словам, характерным для конкретного хаба. Малочисленные хабы предсказываются намного хуже, т.к. или очень мало наблюдений, или в статьях мало специальной характерной для хаба лексики. Каждая модель считалась ~2-2.5 часа, дождаться подбора лучших гиперпараметров с помощью GridSearchCV не удалось.

В **методе опорных векторов с линейным ядром** ощутимо быстрее работает дефолтная мультиклассовая стратегия «One-vs-rest» (multi_class='ovr') по сравнению с multi_class='crammer_singer'. При указании class_weight='balanced', как рекомендуется для несбалансированных классов, немного повысился F1-macro, остальные метрики у аналогичной модели без указания этого параметра оказались лучше.

Метрики метода опорных векторов с линейным ядром с гиперпараметрами, подобранными GridSearchCV, получились немного выше, чем у логистической регрессии, при этом модель обучается очень быстро (за несколько минут).

Быстрее всего обучается **наивный Байесовский классификатор**, но метрики у такой модели получились хуже всех.

Предсказание хабов по текстам публикаций: multi-class classification

Тексты после **TF-IDF** с параметрами min_df=12, max_df=0.95

Модель	Гиперпараметры	test F1-micro (accuracy)	test F1-macro	test F1-weighted
Logistic Regression	solver='lbfgs' multi_class='multinomial' C=2	0.48	0.16	0.44
LinearSVC	подобраны с помощью GridSearchCV: class_weight=None, C=0.05, multi_class='ovr'	0.50	0.25	0.48
Random Forest	max_depth=15, n_estimators=300	0.24	0.02	0.15

После векторизации TF-IDF с min_df=12 и max_df=0.95 метрики **метода опорных векторов с линейным ядром** с гиперпараметрами, подобранными гридсерчем, получились выше, чем после после векторизации TF-IDF с min_df=20, max_df=1. Время обучения на датасете ~80 тыс. статьях составило около 4 минут.
Одна модель **логистической регрессии** после векторизации с новыми параметрами обучалась также несколько часов. Метрики логистической регрессии получились такими же, как и с параметрами min_df=20, max_df=1.

Случайный лес требует больших вычислительных и временных ресурсов. Одна модель с заранее заданными гиперпараметрами в виде 300 деревьев и глубины 15 считалась ~3 часа. По результатам получилось, что предсказываются только совсем крупные хабы, метрики на тесте намного хуже, чем у других моделей. Чтобы улучшить метрики, нужно работать над гиперпараметрами, но это не позволяет сделать ограничение по памяти.

Предсказание хабов по текстам публикаций: multi-label classification

До этого мы выделяли из списка хабов только один хаб − первый.

Теперь оставим все хабы и попробуем предсказывать их (multi-label классификация).

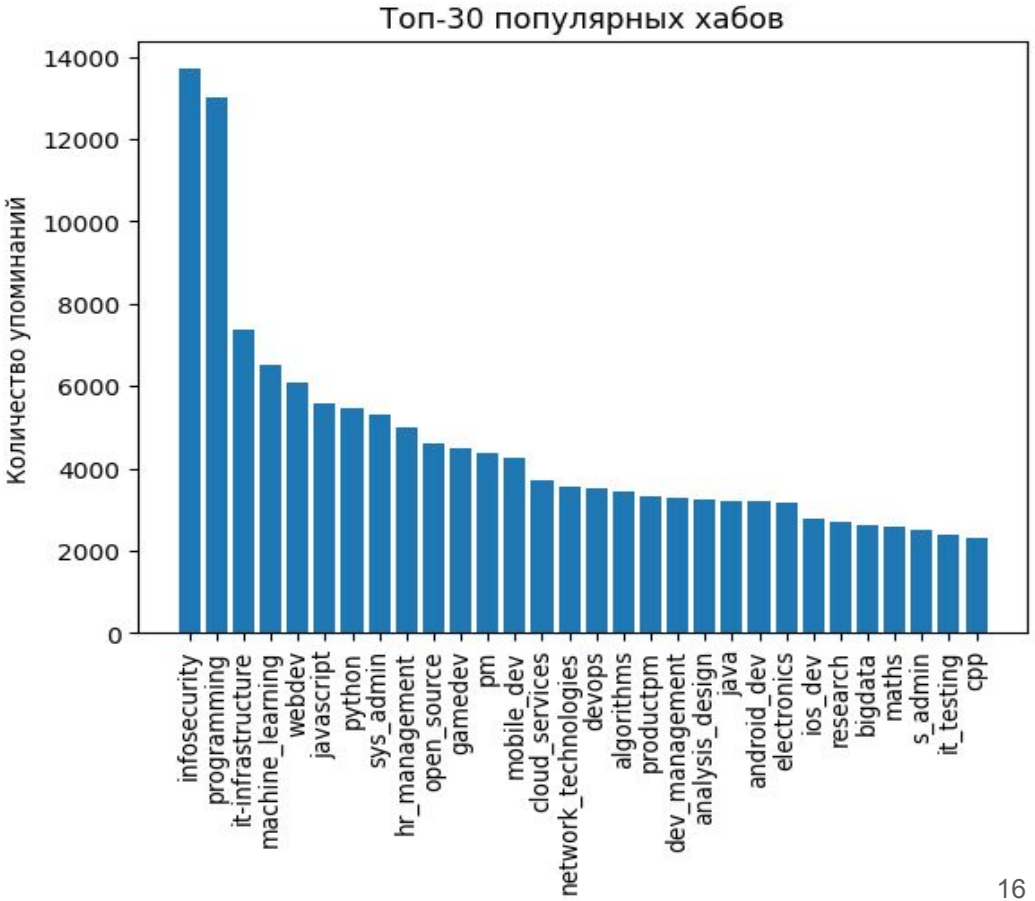
Модель	OneVsRestClassifier(LogisticRegression)
Подобранные гиперпараметры	penalty='l2',C=10
Метрики на тесте	accuracy 0.19 F1-macro 0.34 F1-micro 0.50 F1-weighted 0.48

Предсказывать сразу 321 хаба − не лучшее решение. Часто получается так, что в статье указано несколько хабов, а модель не предсказывает ни один из них.
Выберем для предсказания только самые популярные хабы.

Предсказание хабов по текстам публикаций: топ-10

Эта часть экспериментов приведена в ML/**ML_hubs_top10.ipynb**

Теперь мы выбрали **10 самых часто встречающихся хабов**.
В датасете оставили тексты только с соответствующими хабами:
'infosecurity' (13694),
'programming' (12993),
'**it-infrastructure**'(7377),
'machine_learning' (6512),
'webdev' (6094),
'javascript' (5572),
'python' (5450),
'sys_admin' (5314),
'hr_management' (4976),
'open_source' (4602).



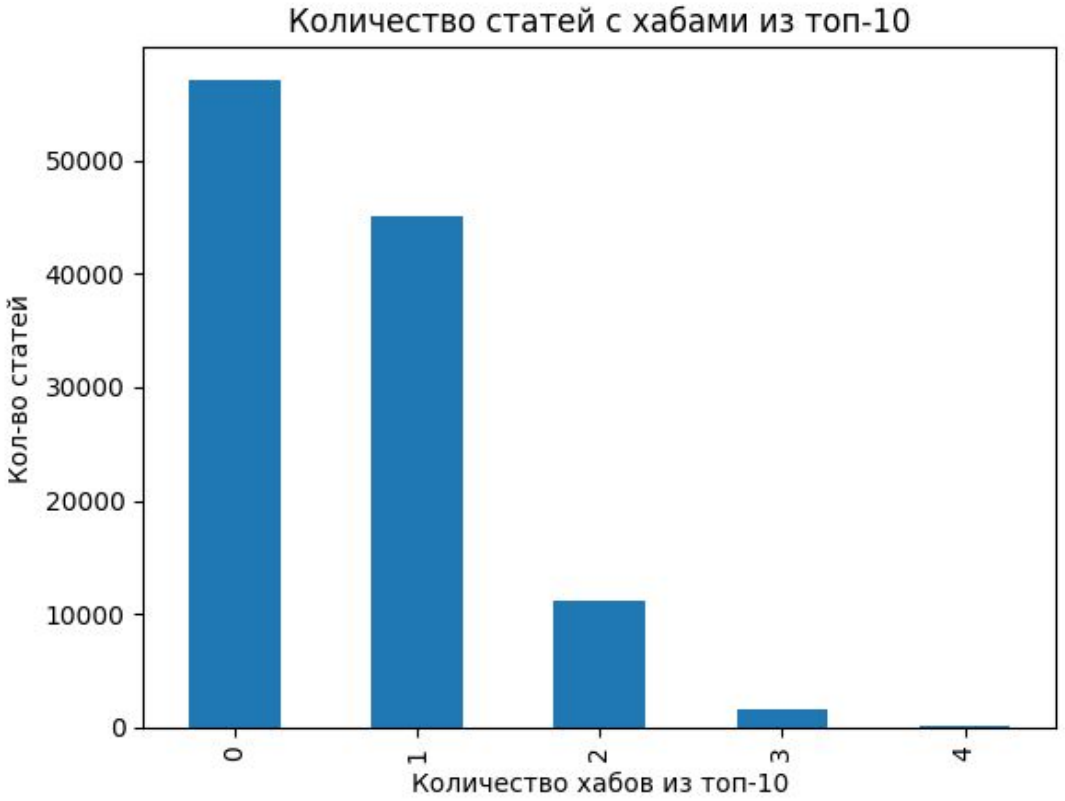
Предсказание хабов по текстам публикаций: топ-10

В датасете **45108** статей с 1 хабом из топ-10, указанным в одной публикации.

В датасете **11130** статей с 2 хабами из топ-10, указанными в одной публикации.

В датасете **1644** статей с 3 хабами из топ-10, указанными в одной публикации.

В датасете **71** статья с 4 хабами из топ-10, указанными в одной публикации.



Предсказание топ-10 хабов по текстам публикаций: multi-label classification

При использовании не всех хабов, а только топ-10 качество моделей значительно выросло:

Модель	**RidgeClassifier**	**OvR(LinearSVC)**	**OvR(LogisticRegression)**
Подобранные гиперпараметры	alpha=1	C=0.5	C=10
Метрики на тесте	F1-micro = 0.71 F1-macro = 0.69 F1-weighted = 0.70	F1-micro = 0.73 F1-macro = 0.71 F1-weighted = 0.72	F1-micro = 0.72 F1-macro = 0.71 F1-weighted = 0.72

С помощью метода **show_weights()** из библиотеки **eli5** вывели значения весов для слов у модели **RidgeClassifier:**

y=infosecurity top features		**y=programming** top features		**y=it-infrastructure** top features		**y=machine_learning** top features		**y=webdev** top features	
Weight[?]	**Feature**	**Weight[?]**	**Feature**	**Weight[?]**	**Feature**	**Weight[?]**	**Feature**	**Weight[?]**	**Feature**
+2.329	безопасность	+2.595	программирование	+2.077	инфраструктура	+2.860	машинный	+1.755	microblog
+1.676	кибербезопасность	+2.482	voximplant	+1.826	serverspace	+1.826	нейросеть	+1.690	voximplant
+1.614	конфиденциальность	+1.953	программист	+1.696	selectel	+1.783	интеллект	+1.671	сайт
+1.575	антифрод	+1.736	skillbox	+1.555	splunk	+1.372	распознавать	+1.670	уважаемый
+1.528	security	+1.653	surf	+1.507	regionsoft	+1.369	chatgpt	+1.607	joomla
+1.514	ctf	+1.556	кодинг	+1.468	ведомость	+1.348	классификатор	+1.554	webrtc
+1.475	уязвимый	+1.534	jetbrains	+1.457	атс	+1.313	bing	+1.547	laravel
+1.455	ida	+1.463	интерпретатор	+1.441	вкс	+1.294	рекомендательный	+1.446	коллбэк
+1.414	ghidra	+1.430	webrtc	+1.437	cores	+1.262	engines	+1.416	веб
+1.369	уязвимость	+1.425	clrium	+1.433	интерфакс	+1.254	bpy	+1.397	некий
+1.366	siem	+1.407	xcode	+1.402	сбой	+1.247	саентист	+1.389	recaptcha
+1.334	мошеннический	+1.360	integer	+1.385	snom	+1.233	tensorflow	+1.349	todos
+1.303	безопасный	+1.340	programming	+1.381	абс	+1.227	datalore	+1.253	hugo
+1.298	crowdsec	+1.268	разработчик	+1.358	crm	+1.219	нейронный	+1.222	свойство
+1.297	suricata	+1.252	антипаттерн	+1.349	apnic	+1.204	онтология	+1.222	редизайн
... 33505 more positive ...		*... 34954 more positive ...*		*... 33520 more positive ...*		*... 33995 more positive ...*		*... 32701 more positive ...*	
... 43524 more negative ...		*... 42075 more negative ...*		*... 43509 more negative ...*		*... 43034 more negative ...*		*... 44328 more negative ...*	
-0.902	panda	-1.170	phpstorm	-0.872	реагирование	-0.707	recommendations	-0.840	elem
-0.922	adaptive	-1.190	некий	-0.873	ise	-0.720	совокупность	-0.868	отображать
-1.002	checkup	-1.199	webgl	-0.890	минкомсвязь	-0.826	<BIAS>	-0.904	поскольку
-1.033	тспа	-1.280	конструкция	-0.891	межсетевой	-0.848	гудфеллоу	-0.964	loadable
-1.087	санкция	-1.537	holyjs	-0.967	завершать	-0.892	nnapi	-1.193	деструктуризация

Сравнение подходов извлечения признаков из текста

- **Примененная в дальнейшем модель после обработки признаков - это LogisticRegression с параметрами solver = 'lbfgs',multi_class = 'multinomial', max_iter=1000, random_state=42, n_jobs = -1**
- **Word2Vec использовался для построения эмбеддинга, то есть построили эмбеддинги текстов на основе эмбеддингов слов, которые в свою очередь уже отбирались по весам, которые можно увидеть ниже (это было проведено в тестовом режиме, то есть в дальнейшем мы планируем улучшить их согласно каждому лейблу в таблице на слайде 19), так как сейчас это было сделано для всех таргетов**
- **Ноутбук можно найти в ML/HABR_ML_text_v2_second.ipynb**

In [119]:
```
tokenized_ = [i.split() for i in fr_top['lemmatized_text'].values]

%time w2v = word2vec.Word2Vec(tokenized_, workers=4, vector_size=200, min_count=10, window=3, sample=1e-3)
```

```
CPU times: user 12min 29s, sys: 12.1 s, total: 12min 41s
Wall time: 3min 27s
```

In [120]:
```
w2v.wv.most_similar(positive=['плюс'], topn=10)
```

Out[120]:
```
[('минус', 0.8484389781951904),
 ('преимущество', 0.6751542687416077),
 ('недостаток', 0.6578919291496277),
 ('достоинство', 0.6275618076324463),
 ('профит', 0.5190838575363159),
 ('во-вторых', 0.5104820132255554),
 ('зато', 0.5076562166213989),
 ('во-первых', 0.498101145029068),
 ('несомненный', 0.49752572178840637),
 ('смотря', 0.4908961653709411 6)]
```

In [121]:
```
def get_text_embedding(lemmas, model=w2v.wv, embedding_size=200):

    res = np.zeros(embedding_size)
    cnt = 0
    for word in lemmas.split():
        if word in model:
            res += np.array(model[word])
            cnt += 1
    if cnt:
        res = res / cnt
    return res
```

Сравнение подходов извлечения признаков из текста

- **Текущее применение описанного метода Word2Vec не улучшило скор, но было не хуже остальных подходов, что дает в перспективе шанс на улучшение скора посредством улучшения эмбеддинга, взяв за основу веса со слайда 19.**
- **То есть текущее распределение метрик показывает, почему был использован TF-IDF в предыдущих слайдах с моделями, так как он наиболее эффективен на данный момент без улучшения эмбеддингов и применения DL способов извлечния признаков таких, как BERT, например.**

Подход	**f1-score accuracy**	**f1-score weighted avg**
BOW	0.67	0.65
TF-IDF	0.71	0.70
Word2Vec	0.69	0.69

Сравнение подходов извлечения признаков из текста

- Нужно отметить, что для попытки улучшить предсказательную силу, был применен метод объединения названия статьи и текста статьи, так как в названии обычно содержится краткая выжимка всей сути статьи.
- Также была попытка применения FastText, однако не получилось дождаться результатов вычислений, либо по причине большого набора данных, либо по причине допущения ошибки, однако это будет реализовываться и изучаться в дальнейшем.
- Ноутбук можно найти в ML/HABR_ML_text_v2_second.ipynb

7.1.1.4 FastText

In [129]:
```
!pip install fasttext
```

In [132]:
```
import fasttext
import fasttext.util
fasttext.util.download_model('ru', if_exists='ignore')
```

In [133]:
```
ft = fasttext.load_model('cc.ru.300.bin')
```

```
Warning : `load_model` does not return WordVectorModel or SupervisedModel any more, but a `FastText` object which is very simil
ar.
```

In [134]:
```
X_train['ft_embedding'] = X_train['lemmatized_text'].apply(lambda x: get_text_embedding(x, model=ft, embedding_size=300))
print('train done')

X_test['ft_embedding'] = X_test['lemmatized_text'].apply(lambda x: get_text_embedding(x, model=ft, embedding_size=300))
```

Дальнейшие планы

- Увеличение количества хабов с топ-10 до ±150 хабов
- Применение эмбеддингов из DL-модели для обучения моделей в усовершенствованном виде
- Разработка полноценного сервиса (streamlit/телеграм-бот) для предсказания по введенному тексту подходящих хабов. Подразумевается возможность определения количества выдаваемых хабов
- Применение DL-моделей BERT, FastText и т.д.
- Посмотреть различия на разных солверах в моделях (как в логистической регрессии, например)