Отчёт: Классификация хабов

1. **Подготовка данных**

Перед моделированием были проведены несколько этапов очистки и нормализации данных.

   a. **Удаление ненужных колонок**

Удалены поля status, id и time, не влияющие на решение задачи.

   b. **Преобразование списков**

- Колонки keywords и hubs содержали списки в виде строк, поэтому была выполнена безопасная конвертация:
- Парсинг строки через ast.literal_eval
- Замена на пустой список при ошибке парсинга
- Это гарантировало единый формат для всех объектов.

   c. **Очистка текста**

Для колонки text была выполнена комплексная очистка:

- удаление HTML (BeautifulSoup)
- удаление ссылок
- удаление эмодзи
- удаление чисел
- приведение к нижнему регистру
- токенизация
- удаление стоп-слов
- сборка очищенного текста обратно в строку
- Создано новое поле **clean_text**.

   d. **Очистка ключевых слов**

Ключевые слова были объединены в строку (clean_keywords), чтобы их можно было подавать в векторизатор.

   e. **Нормализация хабов**

Все хабы были приведены к нижнему регистру, чтобы не возникало дубликатов:["Python", "python", "PYTHON"] → ["python"]

   f. **Фильтрация редких хабов**

Всего исходно было **1417 уникальных хабов**. Оставлены только те, что встречаются **не менее 20 раз**, чтобы модель имела достаточно обучающих примеров.

- Было хабов: **1417**
- После фильтрации осталось: **764**
- Размер датасета после фильтрации: **151 792 записей**

Это сильно улучшает качество и устойчивость моделей.

2. Векторизация текстов

Использовались три независимых HashingVectorizer:

Поле	**n_features**	**ngram_range**
clean_text	150 000	(1,2)
title	1 000	(1,2)
keywords_str	500	(1,1)

HashingVectorizer выбран из-за:

- высокой скорости
- отсутствия необходимости хранить словарь
- устойчивости на больших данных

Все три матрицы объединяются через hstack() в единый sparse-вектор.

3. Подготовка таргета

Для мультихабовой классификации применён MultiLabelBinarizer.

Каждая строка конвертируется в многомерный one-hot-вектор размером #hubs.

4. Обучение моделей

Использовались четыре классические модели в рамках OneVsRestClassifier:

1. **MultinomialNB**
2. **SGDClassifier (hinge)** — аналог Linear SVM
3. **SGDClassifier (log_loss)** — логистическая регрессия
4. **SGDClassifier (modified_huber)** — гибридная устойчивая функция потерь

Все модели были обучены на корпусе векторизованных текстов.

5. Метрики качества

Оценка проводилась по Jaccard micro и Jaccard macro:

- Micro — взвешивает частые хабы сильнее
- Macro — усредняет по всем хабам, независимо от их частоты (поэтому всегда ниже micro)

6. Полученные результаты

Ниже — итоговые значения Jaccard для каждой модели:

Модель	**Jaccard micro**	**Jaccard macro**
MultinomialNB	0.0818	0.0058
SGD_hinge	0.1399	0.0347
SGD_log	0.0834	0.0111
SGD_huber	**0.2473**	**0.0857**

Лучший результат: SGDClassifier (modified_huber)

Он превосходит все остальные модели.

7. Анализ результатов

A. **MultinomialNB**

Показал самые слабые метрики.

Причины:

- Наивный байес плохо работает на разнотипных фичах (текст + ключевые слова + title)
- Мультихабовая разреженная матрица снижает качество

- Байес чувствителен к коррелированным признакам (a ngram их создаёт много)

B. SGD_hinge (SVM)

Дал средний результат:

- SVM хорошо работает на высокоразмерных данных
- Но hinge-loss плохо справляется с шумными текстовыми данными

C. SGD_log (логистическая регрессия)

Метрика хуже SVM:

- log-loss неустойчив к классовому дисбалансу
- модель стремится выбирать только самые частые хабы

D. SGD_huber

Это гибридная функция потерь, устойчивая к выбросам и дисбалансу.

Поэтому она:

- справляется с большим количеством редких и средних классов
- менее чувствительна к “шумным” текстам
- лучше обучается на разреженных матрицах высокой размерности

Итог: modified_huber оказался оптимальным для больших многоклассовых задач.

8. Почему результаты всё ещё низкие?

лучший Jaccard ≈ **0.25**, но это очень мало :

- 764 уникальных хабов — очень много
- Распределение хабов сильно неравномерное
- Тексты разной длины, разного качества, много шума
- HashingVectorizer не даёт интерпретируемых признаков
- Много хабов семантически близки, но модель об этом не знает
- Некоторые хабы требуют понимания контекста, а не n-gram

9. Что можно улучшить

- Использовать TF-IDF вместо Hashing

  Даже при 150k фичей TF-IDF может дать лучшее качество.

- Уменьшить количество хабов

  Сгруппировать редкие или объединить похожие.

- Добавить FastText / Word2Vec среднее по эмбеддингам