

Реализация модели для обнаружения сгенерированных текстов

1 Постановка задачи

В последние годы широкое распространение получили крупные языковые модели, способные генерировать тексты, трудноотличимые от написанных человеком. Такие модели могут быть использованы не только для полезных задач, но и для создания фейковых новостей, спама, пропаганды и других видов недобросовестного контента. В связи с этим возникает практическая задача: обнаружение сгенерированных текстов.

В рамках данной работы задача формулируется как задача бинарной классификации: по входному тексту необходимо определить, был ли он написан человеком (Human) или сгенерирован одной из языковых моделей (Non-Human). В качестве эмбедингов используется предобученная модель DeepPavlov/rubert-base-cased, после чего классификатор обучается на признаковом пространстве.

2 Используемый набор данных и их подготовка

Для обучения и оценки модели используется датасет RuATD 2022, предназначенный для задачи детектирования искусственно сгенерированных текстов на русском языке. Он содержит тексты, сгенерированные различными языковыми моделями (например, ruGPT3, mT5, ruT5, M-BART и др.), а также тексты, написанные человеком (Human). Подготовка данных включает следующие этапы:

- Загрузка и чтение CSV-файлов: используются два файла — train.csv (обучающая выборка) и val.csv (валидационная выборка). Каждая строка содержит идентификатор (Id), текст (Text) и исходный класс (Class).
- Преобразование меток для бинарной классификации. Все тексты с меткой Human получают значение метки 0, отражающее тексты, написанные человеком. Все остальные классы (генерированные моделями) объединяются в метку 1, что позволяет свести задачу к

бинарной классификации и сосредоточиться на выявлении «нечеловеческих» текстов.

- Токенизация и векторизация. Для каждого текста выполняется токенизация с использованием модели `rubert-base-cased`. Далее тексты подаются в модель, где для каждого токена вычисляется скрытое представление. Затем эмбединг всего текста формируется методом `mean pooling` — усреднением векторов по всем токенам с учётом маски внимания (`attention_mask`).
- Сохранение признаков. Полученные эмбединги (векторы размерности 768) и соответствующие метки сохраняются в формате `.pt` (бинарные файлы библиотеки `torch`) для последующего использования на этапе обучения модели классификации.

Такой подход позволяет получить компактное, фиксированной размерности представление для каждого текста, подходящее для подачи в любую классическую модель машинного обучения.

Id	Text	Class
2	Как насчёт ещё одного раунда?	OPUS-MT
8	К декабрю 1943 года старший лейтенант Николай Сириченко командовал истребительно-противотанковой батареей 45-миллиметровых ор	
24	Соловьев рассказал о просмотре в эфире одного выпуска шоу с Малаховым и Корчевниковым.	rubPT3-Large
47	"Доброе утро, девочки и мальчики", mT5-Small	
56	В октябре 2005 — докладчица на Европейском Конвенте Открытого ПО организованного O'Reilly Media в Амстердаме.	M2M-100
63	"Также большая проблема состоит в том, что мы не можем быть уверены на 100%. В этом плане у нас есть очень большой потенциал. — Что касается вашего отношения к "СуперФакту" и других проектов?— Мы с вами уже говорили об этой теме несколько лет назад...	
70	"Половину субъектов бизнес-сообщества, представляют предприниматели, занятые в сфере торговли."	Human

Рисунок 1 — демонстрация разметки набора данных

3 Использование предобученной модели `DeepPavlov/rubert-base-cased`

Для извлечения признаков из текстов используется предобученная трансформерная модель `DeepPavlov/rubert-base-cased`, основанная на архитектуре BERT и адаптированная под русский язык. Данная модель обучена на больших русскоязычных корпусах, что делает её хорошо подходящей для понимания и представления текстов на русском языке.

Модель `rubert-base-cased` реализует механизм само-внимания (`self-attention`), позволяющий учитывать контекст слов в пределах всего предложе-

ния. На выходе модель формирует векторные представления для каждого токена. Для получения одного вектора на весь текст используется агрегация по токенам с помощью `mean pooling` — усреднение скрытых состояний по токенам с учётом их значимости, заданной маской внимания. Достоинствами использования данной модели можно назвать:

- Поддержку кириллического алфавита и морфологии русского языка;
- Глубокое представление семантики текста, подходящее для задач классификации;
- Отсутствие необходимости дополнительного обучения модели: она используется как функция преобразования текста в вектор признаков.

Таким образом, `rubert-base-cased` выступает в качестве эмбедингового блока, обеспечивая качественное и устойчивое представление текстов для последующего обучения модели классификации.

3.4 Обучение модели

Для классификации текстов на сгенерированные искусственным интеллектом и написанные человеком была реализована глубокая нейросетевая модель на основе полносвязного многослойного перцептрона (MLP). Архитектура модели реализована с использованием библиотеки `PyTorch` и представлена в виде класса `DeepMLClassifier`:

Листинг 1 – класс `DeepMLClassifier`

```
class DeepMLClassifier(nn.Module):
    def __init__(self, input_dim, dropout=0.4):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, 768),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(768, 256),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(256, 64),
            nn.ReLU(),
            nn.Dropout(dropout),
            nn.Linear(64, 2)
        )
    def forward(self, x):
        return self.net(x)
```

Модель принимает на вход эмбединги размерностью 768, полученные предварительно (например, с помощью BERT-подобной модели). Далее данные проходят через серию линейных слоёв с функцией активации ReLU и регуляризацией в виде Dropout (с вероятностью отключения нейронов 0.4). Последний слой возвращает логиты для двух классов: "человеческий текст" и "сгенерированный ИИ".

Обучение проводилось на 30 эпохах, с использованием кросс-энтропийной функции потерь (CrossEntropyLoss) и оптимизатора Adam. Процесс обучения сопровождался валидацией модели на отложенной выборке. Ниже представлены примеры логов из процесса обучения:

```
[EPOCH 1/30] Train loss: 0.4611 | Val loss: 0.4185 | Val acc: 0.8046 | Time: 33.4 sec
[EPOCH 2/30] Train loss: 0.4189 | Val loss: 0.3966 | Val acc: 0.8085 | Time: 50.9 sec
[EPOCH 3/30] Train loss: 0.4013 | Val loss: 0.3860 | Val acc: 0.8224 | Time: 30.9 sec
[EPOCH 4/30] Train loss: 0.3899 | Val loss: 0.3809 | Val acc: 0.8144 | Time: 31.0 sec
[EPOCH 5/30] Train loss: 0.3777 | Val loss: 0.3863 | Val acc: 0.8219 | Time: 31.3 sec
[EPOCH 6/30] Train loss: 0.3704 | Val loss: 0.3825 | Val acc: 0.8258 | Time: 50.8 sec
[EPOCH 7/30] Train loss: 0.3600 | Val loss: 0.3780 | Val acc: 0.8233 | Time: 84.3 sec
[EPOCH 8/30] Train loss: 0.3518 | Val loss: 0.3820 | Val acc: 0.8220 | Time: 82.5 sec
[EPOCH 9/30] Train loss: 0.3439 | Val loss: 0.3785 | Val acc: 0.8196 | Time: 81.0 sec
[EPOCH 10/30] Train loss: 0.3359 | Val loss: 0.3888 | Val acc: 0.8246 | Time: 48.4 sec
[EPOCH 11/30] Train loss: 0.3290 | Val loss: 0.3808 | Val acc: 0.8230 | Time: 33.0 sec
[EPOCH 12/30] Train loss: 0.3213 | Val loss: 0.3866 | Val acc: 0.8206 | Time: 49.1 sec
[EPOCH 13/30] Train loss: 0.3128 | Val loss: 0.3979 | Val acc: 0.8242 | Time: 90.7 sec
[EPOCH 14/30] Train loss: 0.3050 | Val loss: 0.3948 | Val acc: 0.8239 | Time: 96.3 sec
[EPOCH 15/30] Train loss: 0.3004 | Val loss: 0.4194 | Val acc: 0.8282 | Time: 100.9 sec
[EPOCH 16/30] Train loss: 0.2940 | Val loss: 0.4005 | Val acc: 0.8240 | Time: 107.3 sec
[EPOCH 17/30] Train loss: 0.2878 | Val loss: 0.4023 | Val acc: 0.8236 | Time: 113.4 sec
[EPOCH 18/30] Train loss: 0.2803 | Val loss: 0.4044 | Val acc: 0.8242 | Time: 118.7 sec
[EPOCH 19/30] Train loss: 0.2741 | Val loss: 0.4068 | Val acc: 0.8192 | Time: 122.5 sec
[EPOCH 20/30] Train loss: 0.2681 | Val loss: 0.4500 | Val acc: 0.8216 | Time: 114.9 sec
[EPOCH 21/30] Train loss: 0.2640 | Val loss: 0.4166 | Val acc: 0.8181 | Time: 114.6 sec
[EPOCH 22/30] Train loss: 0.2583 | Val loss: 0.4519 | Val acc: 0.8191 | Time: 114.2 sec
[EPOCH 23/30] Train loss: 0.2546 | Val loss: 0.4751 | Val acc: 0.8222 | Time: 114.1 sec
[EPOCH 24/30] Train loss: 0.2493 | Val loss: 0.4639 | Val acc: 0.8224 | Time: 128.7 sec
[EPOCH 25/30] Train loss: 0.2442 | Val loss: 0.4376 | Val acc: 0.8180 | Time: 81.2 sec
[EPOCH 26/30] Train loss: 0.2393 | Val loss: 0.5187 | Val acc: 0.8240 | Time: 38.6 sec
[EPOCH 27/30] Train loss: 0.2365 | Val loss: 0.4290 | Val acc: 0.8187 | Time: 39.1 sec
[EPOCH 28/30] Train loss: 0.2300 | Val loss: 0.4586 | Val acc: 0.8193 | Time: 39.2 sec
[EPOCH 29/30] Train loss: 0.2291 | Val loss: 0.4864 | Val acc: 0.8192 | Time: 40.0 sec
[EPOCH 30/30] Train loss: 0.2233 | Val loss: 0.5160 | Val acc: 0.8179 | Time: 54.6 sec
[INFO] Модель сохранена в saved_model.pt
```

Рисунок 2 – демонстрация логов при обучении

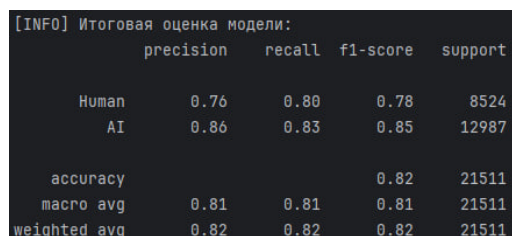
Из логов видно, что в течение первых 10 эпох наблюдается стабильное уменьшение функции потерь и рост точности. Начиная примерно с 15–20 эпохи модель начинает переобучаться: несмотря на снижение тренировочной ошибки, валидационная ошибка увеличивается. Тем не менее, точность на валидации сохраняется на высоком уровне (~82%).

Финальная модель была сохранена на диск в файл saved_model.pt и использовалась для итоговой оценки качества.

5 Оценка качества модели

Для финальной оценки модели использовалась тестовая выборка размером 21 511 объектов. Классы представлены неравномерно:

- Человеческие тексты: 8 524 объекта;
- Сгенерированные ИИ: 12 987 объектов.



```
[INFO] Итоговая оценка модели:
      precision    recall  f1-score   support

   Human         0.76      0.80      0.78       8524
      AI         0.86      0.83      0.85      12987

 accuracy              0.82       21511
  macro avg           0.81      0.81      0.81       21511
 weighted avg           0.82      0.82      0.82       21511
```

Рисунок 3 – демонстрация итоговой оценки

Общая точность модели составила 0.82 (82%). Дополнительно были рассчитаны усреднённые показатели:

- Macro average: precision = 0.81, recall = 0.81, F1 = 0.81
- Weighted average: precision = 0.82, recall = 0.82, F1 = 0.82

Эти метрики показывают, что модель хорошо различает оба класса, несмотря на дисбаланс данных. Особенно высокие значения precision и recall для класса "AI" (0.86 и 0.83 соответственно) указывают на способность модели эффективно определять сгенерированные тексты.

6 Сохранение и использование модели

После завершения обучения классификационной нейросети модель сохраняется в файл saved_model.pt. Это позволяет исключить необходимость повторного обучения при каждом запуске, обеспечить воспроизводимость результатов и упростить переносимость модели между средами. Сохранение выполняется с использованием средств библиотеки PyTorch, которая предоставляет компактное и удобное средство сохранения весов нейронной сети в формате .pt. Для демонстрации работы модели и возможности её практического применения было разработано простое веб-приложение на основе микрофреймворка Flask. Его задачей является организация пользовательского интерфейса для анализа текстов, поданных на вход, и визуализация результата — вероятности того, что текст сгенерирован искусственным интеллектом.

После запуска сервера Flask пользователь может открыть страницу по адресу `http://127.0.0.1:5000`. Интерфейс позволяет ввести произвольный текст и отправить его на обработку. Далее выполняются следующие шаги:

- Токенизация текста с использованием того же токенизатора, что применялся на этапе обучения.
- Преобразование текста в эмбединги с помощью модели RuBERT.
- Классификация с помощью заранее обученного MLP, загруженного из `saved_model.pt`.
- Отображение результата — пользователю показывается вероятность того, что текст был сгенерирован ИИ.

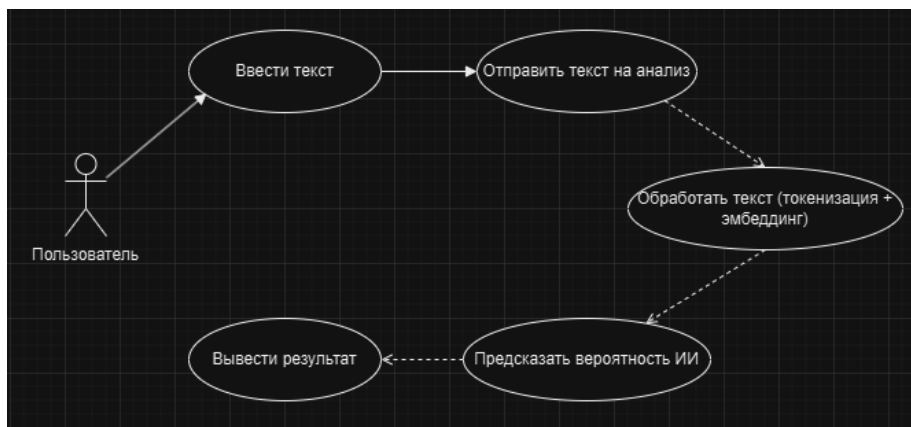


Рисунок 4 – диаграмма активности

Этот подход обеспечивает полноценную интеграцию модели в интерактивную среду и демонстрирует её применение в условиях, приближенных к реальным. Благодаря отделению этапов обучения и инференса достигается высокая гибкость, масштабируемость и простота использования.

Детектор сгенерированного текста

В Татарии начали проверку по факту восточных танцев на фоне мечети.

Проверить

Вероятность генерации ИИ: 73.88%

The screenshot shows a web application titled 'Детектор сгенерированного текста'. It features a text input field containing the sentence 'В Татарии начали проверку по факту восточных танцев на фоне мечети.' Below the input field is a button labeled 'Проверить'. At the bottom, the application displays the result: 'Вероятность генерации ИИ: 73.88%'.

Рисунок 5 – демонстрация работы приложения