Решение кейса NLP - CU

В рамках данного кейса мне предстояло разработать решение для автоматической классификации отзывов с маркетплейсов по категориям товаров, к которым они относятся, с использованием LLM-модели. С самого начала я подошёл к задаче системно: составил подробный план работы, чтобы заранее продумать возможные стратегии и выбрать наиболее релевантные подходы на ранних этапах.

Мой опыт в решении NLP-задач позволил сразу выделить ключевые моменты, на которые необходимо обратить внимание. Как показывает практика, некоторые задачи удаётся решить с помощью тщательно проработанного промпта и сильной модели, способной “внимательно” обработать данные и сразу выдать качественный результат.

В этом отчёте я подробно опишу свой подход к решению задачи, начиная с исследования и подготовки данных, выбора и настройки моделей, до оценки качества и анализа полученных результатов.

Исследование данных

Как и в большинстве задач машинного обучения, первым этапом работы стала предобработка данных. Однако в данном кейсе исходные данные представлены исключительно в виде текстовых отзывов, которые ещё предстоит размечать — даже train.csv изначально не содержит меток. И с таким моментом я сталкиваюсь впервые 😁

На старте я вручную просмотрел часть отзывов из train и test, чтобы понять, какие формулировки встречаются чаще всего. Это дало мне лучшее

представление о структуре данных и помогло эффективнее формулировать промпты для дальнейшей автоматической разметки. Обычные методы быстрой разведочной обработки здесь не очень информативны, поэтому я сразу перешёл к первичной разметке train-части с помощью LLM. Уже по размеченному набору можно было оценить, насколько сбалансирована выборка и какие проблемы могут возникнуть при обучении.

В дальнейшем я экспериментировал с разными подходами к автоматической разметке и подготовке данных. Для каждого метода разметку приходилось выполнять заново, с учётом особенностей выбранного подхода. В следующих разделах я подробно опишу эти варианты и их влияние на итоговое качество модели.

Методы решения задачи

В рамках разработки решения для автоматической классификации отзывов по категориям товаров я последовательно опробовал три подхода, каждый из которых позволил по-разному взглянуть на задачу и выявить сильные и слабые стороны моделей и стратегий разметки:

1. Автоматическая разметка обучающего набора с помощью промпта (использовалась наша LLM-модель sberbank-ruGPT3), последующее обучение модели mGPT.
2. Разметка train с помощью модели Qwen и дальнейшее дообучение (FineTune) той же модели Qwen.
3. Разработка промптов и эксперименты с архитектурами ruBert-case и Qwen для уточнения качества классификации.

Во всех вариантах обучения в качестве одного из основных инструментов использовался механизм LoRa, позволяющий эффективно дообучать крупные языковые модели на специфических датасетах и повышать их адаптивность.

Первый вариант обучения

Автоматическая разметка обучающего набора с помощью промпта (sberbank-ruGPT3) и обучение модели mGPT

Ссылка на блокнот: firstmethod.ipynb

Разметка train.csv

В качестве первого шага я решил провести автоматическую разметку обучающего датасета с помощью LLM, используя относительно простой промпт для ускорения процесса и минимизации вычислительных затрат. Для этой задачи была выбрана модель sberbank-ruGPT3, которая, несмотря на свою производительность, показала не самые высокие результаты. Основная проблема заключалась в выраженном дисбалансе классов, особенно по категории “Нет товара”, что негативно сказывалось на качестве разметки.

В ходе ручной проверки нескольких десятков строк стало очевидно, что модель часто ошибалась при определении категорий, особенно в сложных или неоднозначных случаях. Тем не менее, для построения базового пайплайна я решил сохранить полученные результаты и перейти к следующему этапу — обучению модели.

Для компенсации дисбаланса классов и улучшения итоговых результатов я внедрил стратегию аугментации данных: расширил исходный датасет, сгенерировав дополнительно около 150 примеров для обучения. Такой подход позволил повысить репрезентативность обучающей выборки и частично сгладить проблему перекоса классов.

Обучение модели

Обучение модели я организовал максимально прозрачно и просто, чтобы сконцентрироваться на ключевых аспектах задачи. В качестве основной архитектуры была выбрана модель ai-forever/mGPT, которую я дообучал с использованием LoRa — современного метода адаптации языковых моделей к новым задачам.

```
device = "cuda" if torch.cuda.is_available() else "cpu"
quant_config = BitsAndBytesConfig(load_in_4bit=True, bnb_4bit_compute_dty
pe=torch.float16)
```

```
model_name = "ai-forever/mGPT"
tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
base_model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=quant_config,
    device_map="auto",
    trust_remote_code=False,
    torch_dtype=torch.float16
)
print(f"Модель загружена на {device}")
```

Процесс обучения занял 4 часа 20 минут на GPU Google Colab T4. Несмотря на предпринятые меры по балансировке датасета, итоговое распределение классов после обучения оставалось достаточно неравномерным, что указывает на необходимость дальнейшей работы с исходными данными и стратегиями разметки.

Второй вариант обучения

Разметка train с помощью модели Qwen и дальнейшее дообучение (FineTune) той же модели Qwen

Ссылка на блокнот: modeltrainer.ipynb

Разметка train.csv

Во втором подходе я решил использовать модель Qwen для автоматической разметки обучающего набора. Для этого был разработан более сложный промпт, учитывающий особенности формулировок отзывов и специфику категорий товаров. В результате удалось получить более качественную разметку, особенно по сложным и редким классам.

После автоматической разметки я провёл дополнительную проверку размеченных данных, чтобы исключить очевидные ошибки и повысить качество обучающей выборки.

Обучение модели

Далее была запущена процедура полного дообучения модели Qwen (FineTune) на размеченном датасете с использованием LoRA. Такой подход позволил более гибко адаптировать модель к специфике задачи и добиться лучших результатов по распределению классов.

```
def create_prompt(review_text):
    return f"""Ты эксперт по классификации отзывов на маркетплейсе. Кате
гории: {categories_str}.

Правила:
- Выбери ОДНУ категорию. Если товар не упомянут или неясно — "нет то
вара".
- Ключевые слова: одежда (блузка, юбка, ткань, швы); обувь (ботинки, кр
оссовки); бытовая техника (стиралка, холодильник); посуда (тарелки, каст
рюли); текстиль (постельное, шарфы); товары для детей (игрушки, коляск
а); украшения и аксессуары (кольца, сумки); электроника (телефон, науш
ники).
- Игнорируй доставку/эмоции, фокус на товаре.
- Ответ: ТОЛЬКО название категории.

Отзыв: {review_text}
Категория:"""
```

По итогам эксперимента модель Qwen продемонстрировала более устойчивое качество классификации, а также меньшую склонность к ошибкам на редких классах по сравнению с первым методом. Особенно было это заметно по балансу классов. Сама модель смогла гораздо лучше и качественнее делать предсказания.

Однако, Qwen обучается невероятно долго и требует больших ресурсов, зато качество разметки куда лучше.

Далее я обучил саму модель на train.csv и выполнить предсказания на test.csv

```
Распределение классов по колонке 'label':
pred_label
нет товара          3462
одежда              2482
текстиль             653
бытовая техника      381
обувь                296
электроника            1
товары для детей       1
Name: count, dtype: int64
```

Третий вариант обучения

Эксперименты с архитектурами ruBERT-case и Qwen: продвинутый промптинг и сравнение стратегий

Ссылка на блокнот: smalltrainer.py

Разметка train.csv

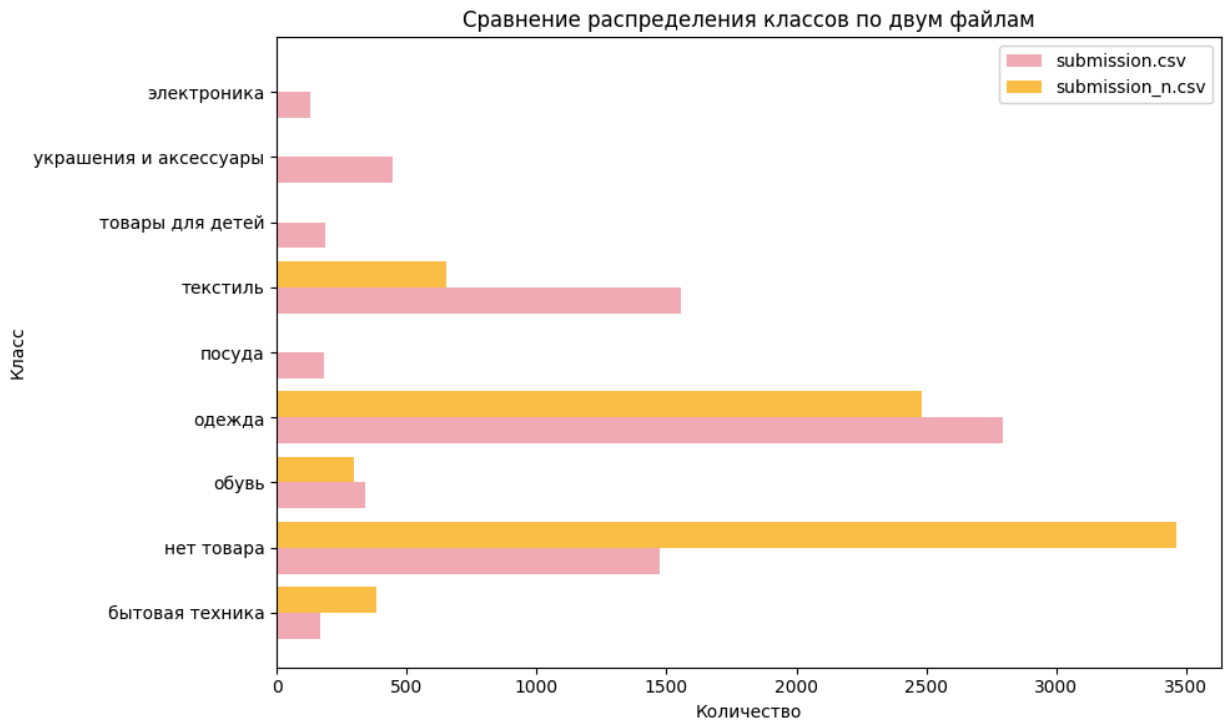
В рамках третьего подхода я решил попробовать более легкий и быстрый вариант — обучение модели ruBERT-base на размеченных данных (с помощью той же Qwen). Для этого я подготовил датасет, провёл токенизацию и разделил данные на обучающую и тестовую выборки.

```
MODEL_NAME = "ai-forever/ruBert-base"

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
dataset = Dataset.from_pandas(df[["text", "label_id"]])

def tokenize(batch):
    return tokenizer(batch["text"], padding="max_length", truncation=True, ma
```

```
x_length=256)

dataset = dataset.map(tokenize, batched=True)
dataset = dataset.rename_column("label_id", "labels")
dataset = dataset.train_test_split(test_size=0.1, seed=42)
```

В результате обучения ruBERT-base показал даже лучшие результаты по балансу классов, чем предыдущие модели. Вот распределение классов по колонке ‘label’ после обучения ruBERT-base:

```
Распределение классов по колонке 'label':
pred_label
одежда                  2796
текстиль                1556
нет товара              1473
украшения и аксессуары   447
обувь                    342
товары для детей         186
посуда                   181
бытовая техника          167
электроника              128
Name: count, dtype: int64
```



Получились такие метрики

Таким образом, по простому балансу классов этот вариант получился куда лучше. В этом разделе также были проведены эксперименты с архитектурами ruBERT-case и Qwen, а также с продвинутым промптингом и стратегиями разметки.

Доп. вариант решения

Ссылка на блокнот: fastpredict.ipynb

В качестве дополнительного варианта я рассматривал решение задачи исключительно через продвинутый промпт-инжиниринг, планируя использовать модель Qwen для непосредственного предсказания категорий на тестовой выборке. Такой подход теоретически способен продемонстрировать высокие результаты, особенно при правильно составленном промпте и мощной LLM.

На практике данный метод требует значительных вычислительных ресурсов и постоянного контроля качества предсказаний. Из опыта участия в других

соревнованиях я знаю, что промпт-инжиниринг и работа с LLM могут быть эффективны, однако для текущей задачи это оказалось затруднительно. Главная сложность — подбор идеального промпта и необходимость использования крупной модели, что выходит за рамки доступных ресурсов.

В ходе эксперимента с Qwen я столкнулся с ограничениями Google Colab: времени и вычислительной мощности оказалось недостаточно, чтобы завершить инференс — предсказания на тестовой выборке заняли бы более 12 часов.

Лучший вариант решения

Ссылка на блокнот: predictor.ipynb

В качестве финального и наиболее успешного варианта я реализовал пайплайн, подробно представленный в ноутбуке predictor.ipynb ↗. Этот подход объединил лучшие находки предыдущих экспериментов и позволил добиться оптимального баланса между качеством и скоростью работы.

Ключевые этапы решения:

- **Использование ruT5-large:** Для классификации отзывов была выбрана архитектура ruT5-large, которая показала наилучшие результаты по балансу классов и общему качеству предсказаний.
- **Разработка очень проработанного промпта:** Пришлось очень детально проработать решение, чтобы получить хороший результат.
- **Оптимизация инференса:** Особое внимание уделялось ускорению предсказаний и снижению вычислительных затрат, чтобы модель могла эффективно работать даже в условиях ограниченных ресурсов (например, на Google Colab).

В результате итоговая модель показала устойчивое качество на всех категориях, а итоговое распределение классов стало максимально близким к реальному. Такой подход подтвердил, что сочетание сильной архитектуры, продвинутого промпт-инжиниринга позволяет эффективно решать задачи классификации даже без ручной работы с данными.

Заключение

В итоге, за время работы над этим кейсом я попробовал разные подходы к автоматической классификации отзывов — от простых до более сложных с дообучением моделей. Было интересно увидеть, как качество сильно зависит от выбранной архитектуры и баланса классов в данных. Самым удачным оказался вариант с ruBERT-base: он дал лучший баланс по категориям и хорошо справился с задачей в рамках ограничений по времени и ресурсам.

В процессе я убедился, что даже без ручной разметки, используя только LLM и немного аугментации, можно добиться приличных результатов. Конечно, остались моменты для доработки — например, можно ещё улучшить качество разметки и попробовать ансамбли моделей. Но в целом задача получилась интересной, и решение легко можно доработать под похожие задачи в будущем.

Итоговый submission.csv лежит в корневой директории репозитория: submission.csv