Разбор задачи "Сбор данных"

Пример реализации и пояснения

---

1. Пример реализации сбора данных через Telegram (Pyrogram)

1.1. Настройка клиента и зависимостей

```
from pyrogram import Client
import asyncio
import nest_asyncio
import csv
nest_asyncio.apply()  # Решение для работы asyncio в Jupyter/Colab

# Инициализация клиента Pyrogram
app = Client(
    "my_account",
    api_id="ВАШ_API_ID",  # Получить на https://my.telegram.org
    api_hash="ВАШ_API_HASH"
)
```

Пояснение:

- **Pyrogram** — библиотека для работы с Telegram API.
- **api_id** и **api_hash** — уникальные ключи для доступа к API.
- `nest_asyncio` позволяет запускать асинхронные циклы в средах с уже активным event loop (например, Jupyter Notebook).

1.2. Функция сбора данных

```
async def TelegramBatchLoader(author_url):
    async with app:
        async for message in app.get_chat_history(author_url, limit=300):
            # Извлечение текста из сообщения
            text = message.text or message.caption

            # Пропуск сообщений без текста или коротких текстов
            if not text or len(text) < 200:
                continue

            # Формирование записи
            result = {
                'id': message.id,  # Уникальный ID сообщения в Telegram
```

```
            'text': text[:3000],  # Обрезка до 3000 символов
            'category': 'прочее',  # Категория задаётся вручную для каждого канала
            'source': author_url,
            'length': len(text),
            'date': message.date.date().isoformat()  # Дата в формате YYYY-MM-DD
        }

        # Запись в CSV
        with open('collected_data.csv', 'a', newline='', encoding='utf-8') as f:
            writer = csv.DictWriter(f, fieldnames=result.keys())
            writer.writerow(result)

# Пример запуска для канала
asyncio.run(TelegramBatchLoader("@olympic_russia"))
```

Пояснения:

- **Извлечение текста:**
  Используется `message.text` (текст сообщения) или `message.caption` (подпись к медиафайлу).
- **Фильтрация данных:**
  Сообщения короче 200 символов игнорируются (`len(text) < 200`).
- **Уникальный ID:**
  Вместо ручного счётчика используется `message.id` — уникальный идентификатор сообщения в Telegram.
- **Категории:**
  Категория `прочее` задаётся вручную. Например, для канала `@gabenews` следует изменить на `киберспорт`.
- **Формат даты:**
  `message.date.date().isoformat()` гарантирует запись даты в формате `YYYY-MM-DD`.

---

2. Постобработка данных (Pandas)

2.1. Удаление дубликатов

```
import pandas as pd
df = pd.read_csv('collected_data.csv')
df = df.drop_duplicates(subset=['id'])  # Удаление записей с одинаковым id
```

Пояснение:

- `drop_duplicates` удаляет строки, где совпадает значение в колонке `id`.

2.2. Валидация дат

```
# Конвертация строки в datetime
df['date'] = pd.to_datetime(df['date'], errors='coerce')

# Фильтрация данных за 2023-2024 годы
df = df[(df['date'] >= '2023-01-01') & (df['date'] <= '2024-12-31')]
```

Пояснение:

- `errors='coerce'` превращает невалидные даты в NaN, которые позже удаляются.
- Фильтр оставляет только записи в заданном временном диапазоне.

---

3. Альтернативные методы сбора данных

3.1. Сбор через VK API

```
import vk_api
from datetime import datetime

vk_session = vk_api.VkApi(token='VK_API_TOKEN')
vk = vk_session.get_api()

# Пример получения постов из группы
posts = vk.wall.get(domain='csru_official', count=100)['items']

for post in posts:
    text = post['text']
    date = datetime.fromtimestamp(post['date']).strftime('%Y-%m-%d')

    # Формирование записи аналогично Telegram-примеру
    result = {
        'id': post['id'],  # Уникальный ID поста в VK
        'text': text[:3000],
        'category': 'киберспорт',
        'source': 'https://vk.com/csru_official',
        'length': len(text),
        'date': date
    }
```

Пояснение:

- **VK API** требует токен доступа, который можно получить через VK Dev.

- `post['id']` гарантирует уникальность в рамках группы.

3.2. Веб-скрейпинг (BeautifulSoup)

```
import requests
from bs4 import BeautifulSoup

url = "https://example-news-site.com/sport"
response = requests.get(url)
soup = BeautifulSoup(response.text, 'html.parser')

# Пример парсинга новостных блоков
articles = soup.find_all('div', class_='news-item')
for article in articles:
    text = article.find('p').text.strip()
    date = article.find('time')['datetime']

    # Формирование записи
    result = {
        'id': hash(text),  # Хеш текста как временный ID
        'text': text[:3000],
        'category': 'спорт',
        'source': url,
        'length': len(text),
        'date': date
    }
```

Пояснение:

- **BeautifulSoup** парсит HTML-страницы.
- `hash(text)` генерирует уникальный идентификатор на основе текста (не идеален, но подходит для демо).
- Для сложных сайтов может потребоваться обработка JavaScript (например, через **Selenium**).

Итог

Пример реализации решает задачу сбора данных через Telegram, но требует:

1. **Ручной настройки категорий** для каждого канала.
2. **Контроля уникальности ID** (в примере используется `message.id`).
3. **Дополнительной обработки** для балансировки данных (например, равное количество записей в категориях).

Альтернативные методы (VK API, веб-скрейпинг) интегрируются в ту же структуру, сохраняя формат CSV. Для больших объёмов данных рекомендуется:

- Использовать базы данных (SQLite, PostgreSQL).
- Добавить обработку ошибок (например, повторные запросы при сбоях).
- Реализовать параллельный сбор данных через асинхронные запросы.

Разбор задачи "Обработка текстов"

Пример решения и вариативность методов

---

1. Базовая очистка текста

Цель: Удалить HTML-теги, URL, эмодзи и лишние символы.
Ключевой код:

```
def clean_text(self, text: str) -> str:
    text = BeautifulSoup(text, "html.parser").get_text()  # Удаление HTML
    text = re.sub(r'http\S+|www.\S+', '', text)  # Удаление URL
    text = re.sub(r'[^\w\s\-\.]', ' ', text)  # Удаление спецсимволов (кроме дефисов и точек)
    text = ' '.join(text.split())  # Нормализация пробелов
    return text
```

Альтернативы:

- Для HTML: `lxml` (быстрее, чем `BeautifulSoup`).
- Для эмодзи: библиотека `emoji` с методом `replace_emoji()`.

---

2. Извлечение сущностей

Цель: Сохранить названия команд, турниров, имена.
Использование Natasha:

```
def extract_entities(text: str) -> List[Dict]:
    doc = Doc(text)
    doc.segment(segmenter)  # Разбивка на токены
    doc.tag_ner(ner_tagger)  # Поиск сущностей
    return [
        {"text": span.text, "type": span.type}
        for span in doc.spans
    ]
```

Особенности:

- Распознаёт типы: `ORG` (организации), `PER` (имена), `LOC` (локации).
- Сохраняет нормализованные формы (например, "НХЛ" → "Национальная хоккейная лига").

Альтернативы:

- `spaCy` с моделью `ru_core_news_lg` для NER.
- Регулярные выражения для паттернов (например, "Team [A-Za-z]+").

---

3. Лемматизация

Цель: Привести слова к словарной форме.
Код с pymorphy2:

```
def lemmatize_text(self, text: str) -> str:
    words = word_tokenize(text.lower())
    return ' '.join([
        morph.parse(word)[0].normal_form  # Лемматизация
        for word in words
        if word not in self.stopwords  # Удаление стоп-слов
    ])
```

Альтернативы:

- `spaCy` (быстрее для больших текстов).
- `Stanza` (контекстная лемматизация).

---

4. Технические аспекты

Соответствие требованиям:

- **Форматы:**
    - `entities` сохраняется как JSON-строка (`json.dumps`).
    - CSV-файл включает все исходные тексты (цикл по `df.iterrows()`).

Сохранение данных:
pd.DataFrame(processed_data).to_csv('processed_data.csv', index=False)

---

5. Вариативность решения

Для улучшения качества:

1. **Извлечение сущностей:**
    - Использовать `spaCy` с предобученной моделью для русского языка.
    - Добавить кастомные паттерны для турниров (например, `r"Кубок [A-Za-z]+"`).
2. **Лемматизация:**

    - Заменить `pymorphy2` на `Stanza` для учета контекста.
3. **Обработка аббревиатур:**
    - Добавить словарь сокращений (например, "НХЛ" → "Национальная хоккейная лига").

Итог

Предложенное решение — надежный базовый вариант. Для максимизации баллов рекомендуется:

- Использовать комбинацию Natasha и кастомных правил для сущностей.
- Протестировать альтернативные лемматизаторы (`spaCy`, `Stanza`).
- Добавить обработку исключений для иностранных названий.

Разбор задачи "Классификация текстов"

Пример решения и альтернативные подходы

---

1. Предобработка данных

Цель: Подготовить тексты для обучения модели.

Например:

- **Лемматизация:** Приведение слов к словарной форме (например, "забросил" → "забросить").
- **Удаление стоп-слов:** Исключение незначимых слов (предлоги, частицы).
- **Приведение к нижнему регистру.**

Код:

```
processed_texts = [TextProcessor().lemmatize_text(text) for text in texts]
```

Альтернативы:

- **Стемминг** (например, через `nltk.stem.SnowballStemmer`) — менее точный, но быстрый метод.
- **Глубинные методы:** Использование предобученных моделей (например, BERT для контекстной обработки).

---

2. Векторизация текста

Цель: Преобразовать текст в числовые признаки.

Например:

```
self.vectorizer = TfidfVectorizer(max_features=5000)
X = self.vectorizer.fit_transform(df['lemmatized_text'])
```

Пояснение:

- **TF-IDF** учитывает важность слов в тексте относительно всего корпуса.

- `max_features=5000` ограничивает размерность данных для ускорения обучения.

Альтернативы:

- **Word2Vec/GloVe:** Векторные представления слов.
- **BERT:** Контекстные эмбеддинги через трансформеры (библиотека `transformers`).

---

3. Выбор модели

Например:

self.classifier = RandomForestClassifier(n_estimators=100)

Преимущества:

- Устойчивость к переобучению.
- Работает с разреженными данными (например, TF-IDF матрицей).

Альтернативы:

- **Логистическая регрессия:** Быстрая и интерпретируемая.
- **CatBoost/XGBoost:** Градиентный бустинг для улучшения точности.
- **Нейронные сети:** Многослойные сети через `keras` или `pytorch`.

---

4. Обучение и валидация

Ключевые этапы:

- **Разделение данных:** 80% — обучение, 20% — тест.
- **Кросс-валидация:** Оценка стабильности модели (5 фолдов).

Код:

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
cv_scores = cross_val_score(self.classifier, X_train, y_train, cv=5)

---

5. Формирование отчётов

Требуемые файлы:

predictions.csv:
pd.DataFrame({'id': test_data['id'], 'predicted_class': predictions}).to_csv('predictions.csv')

model_report.json:

```
{
  "model_description": {
    "vectorization": "TF-IDF",
    "classifier": "RandomForestClassifier",
    "preprocessing": ["лемматизация", "удаление стоп-слов"]
  },
  "training_process": {
    "split_ratio": "80/20",
    "cross_validation_scores": [0.85, 0.83, 0.86]
  }
}
```

Итог

Приведенное решение — базовый рабочий вариант. Для максимизации баллов рекомендуется:

- Экспериментировать с векторными представлениями (например, BERT).
- Тестировать разные модели (CatBoost, нейронные сети).
- Добавить обработку дисбаланса классов.