МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»

Факультет компьютерных наук и информационных технологий

УТВЕРЖДАЮ

Зав. кафедрой

_____

_____

(ФИО)

_____

(подпись, дата)

ОТЧЕТ О ПРАКТИКЕ

студента 2 курса, 241 группы, факультета КНиИТ

Петрова Егора Дмитриевича
(ФИО полностью, в родительном падеже)

вид практики: учебная ("Научно-исследовательская работа")

кафедра: кафедра информатики и программирования
(наименование кафедры)

курс: 2
(номер курса)

семестр: 3
(номер семестра)

продолжительность: 18 нед. с 02.09.2024 по 31.12.2024
(кол-во недель) (дата начала) (дата окончания)

Руководитель практики:

зав. кафедрой _____ Огнева М. В.
(должность) (подпись, дата) (ФИО)

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Возможность преобразовать текст из разговорного стиля речи в художественный востребована в литературном творчестве, журналистике, маркетинге, а также в образовательной и культурной деятельности, что подтверждается растущим использованием ИИ технологий в этих сферах [1]. Преобразование текстов из разговорного стиля в художественный позволяет создавать контент, который лучше воспринимается аудиторией, повышает вовлечённость и качество взаимодействия. Автоматизация этого процесса позволит людям с недостаточным уровнем грамотности создавать тексты, которые будут легче восприниматься, а тем, кто уверенно владеет письменной речью, — значительно ускорить написание текстов. Целью курсовой работы является создание и обучение модели машинного обучения, способной преобразовать текст из разговорного стиля в художественный, а также её внедрение в телеграмм бота для удобного взаимодействия с ней.

Задачи курсовой работы:

1. Изучить возможные подходы к созданию такой модели, примеры подобных моделей, решающий проблему переноса стилей, например — в изображениях.
2. Изучить теоретический материал по устройству GAN моделей, глубокому обучению.
3. Определить формат данных, подходящий для обучения подели.
4. Найти подходящие данные для обучения модели.
5. Построить базовую модель и обучить её.
6. Скорректировать гипер параметры модели, решить возникшие проблемы при обучении, улучшенить результаты.
7. Реализовать телеграмм бота и внедрить в него обученную модель.

Задачи на 3-ий семестр:

1. Изучить возможные подходы к созданию такой модели, примеры подобных моделей, решающих проблему преобразования изображений и текстов между заданными доменами.
2. Определить формат данных, подходящий для обучения подели.
3. Найти подходящие данные для обучения модели.
4. Сделать начальную предобработку данных и векторизовать их.

1 Методы преобразования изображений и текстов

В одной из первых и наиболее известных работ, где используется CycleGAN, она используется для перевода изображений между двумя доменами [2]. Такая модель была предложена как подход к решению проблемы отсутствия параллельных данных для некоторых задач по компьютерному зрению и переводов одних изображений в другие. Основная идея заключается в использовании двух генераторов и двух дискриминаторов, а также циклических потерь для сохранения исходных характеристик при преобразовании. Этот подход можно легко адаптировать и для работы с текстом.

Авторы следующей статьи предложили метод переноса стиля текста, который не требует параллельных данных для обучения [3]. Он основан на использовании генератора, который реализован с архитектурой encoder-decoder, и дискриминатора, оценивающего стиль генерируемого текста. Было использовано обучение с подкреплением, где дискриминатор играет роль оценщика стиля, помогая модели генерировать текст, который сохраняет исходное содержание, но меняет стиль. В качестве награды в процессе обучения используется совместная оценка дискриминатором стиля и генерации текстов с корректным содержанием.

Авторы следующей статьи адаптировали классическую архитектуру CycleGAN для работы с текстами [4]. В статье приводятся детализированные эксперименты, демонстрирующие, как использование архитектуры CycleGAN позволяет эффективно обучаться на непараллельных корпусах текстов. Основное внимание уделено проблеме разрыва между стилевыми и содержательными аспектами текста и тому, как CycleGAN помогает преодолеть этот барьер.

Отличного результата добились авторы нового метода, который основан на адаптивном извлечении паттернов переноса стилей из данных и контрастном обучении для создания более точных представлений текстов [5]. Этот метод впервые вводит концепцию паттернов переноса стилей в текстах

и может быть легко интегрирован с другими методами для повышения их производительности. Эксперименты показывают, что предложенный подход эффективен и универсален, что делает его полезным инструментом для дальнейших исследований в данной области.

2 Сбор данных

2.1 Предложения разговорного стиля

Для обучения модели, способной преобразовывать текст из разговорного стиля речи в художественный, нужно много данных обоих стилей. Подходящих размеченных датасетов на русском языке нет, а размечать большое кол-во данных вручную — слишком долго, поэтому данные нужно собрать самостоятельно, а сам датасет будет непараллельным: две колонки, в каждой находятся предложения, представляющие свой стиль, при этом предложения, находящиеся в одной строке могут быть никак не связаны между собой.

Чтобы собрать предложения, отражающие разговорный стиль, был проведён поиск возможно уже существующих датасетов, которые содержали бы подходящие данные. В процессе встречались датасеты, связанные с публикациями в различных социальных сетях, но всё же это не те данные, которые ожидалось найти, поэтому поиск продолжался. Довольно быстро нашёлся датасет, содержащий сообщения на русском языке из 177 самых популярных чатов мессенджера Telegram в формате json-файлов [6]. Структура файлов json представлена на рисунке 1. Были сомнения, можно ли использовать такие данные, ведь в них много нецензурной брани и другой шокирующей информации, но потом пришло осознание — это и есть воплощение современного разговорного стиля речи, который наша модель будет учиться преобразовывать в художественный.

Рисунок 1 — Структура json-файлов

Найденный датасет распространён под лицензией Creative Common Zeros: Public Domain. Используя такую лицензию при публикации датасета, автор отказывается от авторского права на него, поэтому его можно использовать без указания автора даже в коммерческих целях без каких-либо ограничений [7]. Общий вес этого датасета составлял 11 ГБ, а в заархивированном виде — около 3 ГБ. Было решено, что этого вполне достаточно и нужно перейти к сбору предложений художественного стиля.

2.2 Предложения художественного стиля

Источником таких предложений выступили различные литературные произведения, а точнее — проза. Был найден сайт, в одном из разделов которого можно скачать классическую прозу в формате fb2 [8]. Этот формат файлов часто используется для создания и распространения художественной, научной и другой литературы в электронном виде. Обычно именно в этом формате устройства и приложения для чтения электронных книг хранят и обрабатывают их. Некоторые литературные произведения были доступны для

скачивания в txt формате, но было решено не тратить на это время и скачать все книги в формате fb2, а после просто перенести тексты книг в файлы формата txt.

С помощью BeautifulSoup были собраны все ссылки на страницы книг, а с помощью Selenium они были скачаны. Разобраться в работе Selenium было не очень просто, так как это не просто парсинг html страницы, а полноценная автоматизация действий в браузере.

После скачивания книг, их нужно было перевести в формат txt. Формат fb2 основан на XML-разметке, благодаря структурированности которой легко получилось перенести тексты книг в файлы txt формата с помощью BeautifulSoup.

Подробнее с кодом, который использовался для произведения всех описанных действий при сборе данных, можно ознакомиться в приложении А.

3 Небольшая предобработка данных и их сбор в датасет

Все файлы хранились на Google Диске в zip-архивах, поэтому для начала их нужно было разархивировать. Вот как выглядела разархивация txt-файлов, содержащих тексты художественной литературы:

```
with zipfile.ZipFile(books_path, "r") as zip_ref:
    for filename in zip_ref.namelist():
        zip_ref.extract(filename, "/content/books")
```

То же самое было проделано с архивом, содержащим json-файлы с сообщениями из чатов Telegram.

После разархивирования нужных файлов начался парсинг предложений из художественной литературы. Каждый текстовый файл считывался в одну строку, которая с помощью ф-ции `sent_tokenize` из библиотеки nltk разбивалась на предложения. После этого начинался перебор полученных предложений. Каждое очищалось от пробельных символов в начале и конце строки методом `strip`. После этого происходило удаление всех символов, кроме букв русского алфавита, пробелов и некоторых знаков пунктуации с помощью не очень сложного регулярного выражения:

```
sentence = re.sub("[^а-яА-Я---—,.!?:«» ]", "", sentence)
```

Это было необходимо, ведь многие произведения содержали, например, французский язык. После необходимо было очистить начало строки от знаков препинания и пробелов, ведь после удаления ненужных символов они могли встать в начало строки, что было бы некорректно:

```
while len(sentence) > 0 and sentence[0] in "---—,.!?:«» ":
            sentence = sentence[1:]
```

Ну и после такой небольшой предобработки в множество сохранялись предложения, длина которых попадала в следующий интервал: $[\,9; 199\,]$. Получившиеся множество предложений, преобразовав в список, было сохранено в виде Pandas Series. Всего подходящих предложений художественного стиля было получено 1382549.

Точно такие же действия производились с сообщениями из чатов Telegram. Чтобы датасет был сбалансированным, то есть чтобы количество предложений обоих стилей было одинаковым, нужно было прервать перебор сообщений при достижении количества предложений художественного стиля.

Множество предложений разговорного стиля точно также были сохранены в Pandas Series, после чего оба Pandas Series были объединены в один Pandas DataFrame:

```
data = pd.DataFrame({"lit_text": lit, "tg_text": com})
```

В переменных `lit` и `com` сохранены Pandas Series с предложениями художественного и разговорного стилей соответственно. Датасет был сохранён в формате csv-таблицы на Google Диск.

Подробнее с кодом, который использовался для произведения всех описанных действий в этом разделе, можно ознакомиться в приложении Б.

4 Векторизация данных

4.1 Подготовка векторных представлений для слов

Чтобы модель машинного обучения могла обучиться на собранных текстовых данных, необходимо их преобразовать в векторы действительных чисел. Было решено использовать вектора Navec — коллекцию предобученных эмбеддингов для русского языка [9].

Опыт работы с этими векторными представлениями есть: при обучении модели машинного обучения, которая содержала слои LSTM, на большом количестве данных не хватало оперативной памяти, потому что каждый вектор Navec содержит 300 32-битных чисел плавающей запятой.

Было решено изменить готовые эмбеддинги в целях оптимизации и уменьшении нагрузки на оперативную память: сжать их с размерности 300 до 10, поменять тип действительных чисел с float-32 на float-16. Для начала вектора Navec были сохранены в Numpy Array по порядку:

```
path = 'navec_hudlit_v1_12B_500K_300d_100q.tar'
navec = Navec.load(path)
embeddings = list()
for word in navec.vocab.words:
        embeddings.append(navec[word])
embeddings = np.array(embeddings)
```

Сжать полученный Numpy Array помог PCA из библиотеки scikit-learn:

```
target_dim = 10
pca = PCA(n_components=target_dim)
reduced_embeddings = pca.fit_transform(embeddings)
```

После чего был сформирован словарь, содержащий сжатые эмбеддинги Navec. При его формировании, был изменён и тип данных на менее точный:

```
reduced_navec = {word: np.array(reduced_embeddings[i],
dtype=np.float16) for i, word in enumerate(navec.vocab.words)}
```

4.2 Подготовка векторных представлений для знаков пунктуации

Чтобы сохранить как можно больше информации о предложениях, нужно было векторизовать не только слова, но и знаки пунктуации. Для этого был сделан отдельный словарь, в котором хранились постоянные вектора для различных знак пунктуации. Они точно такие же, как и сжатые вектора Navec — десять 16-битных чисел с плавающей запятой, среди которых лишь последнее отличается от нуля. В предложениях встречались разные дефисы и тире, поэтому их вектора было решено сделать идентичными. Вот как выглядит словарь:

```
punkt_vectors = {
    ".": np.array([0.0] * (target_dim - 1) + [1.0], dtype=np.float16),
    "!": np.array([0.0] * (target_dim - 1) + [0.9], dtype=np.float16),
    "?": np.array([0.0] * (target_dim - 1) + [0.8], dtype=np.float16),
    ",": np.array([0.0] * (target_dim - 1) + [0.5], dtype=np.float16),
    ":": np.array([0.0] * (target_dim - 1) + [0.6], dtype=np.float16),
    "–": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "-": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "—": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "«": np.array([0.0] * (target_dim - 1) + [0.2], dtype=np.float16),
    "»": np.array([0.0] * (target_dim - 1) + [0.3], dtype=np.float16),
}
```

Константа target_dim была определена при подборе итогового размера сжатого вектора Navec и равна 10.

4.3 Подготовка к токенизации предложений

Токенами, для которых впоследствии брались соответствующие вектора из подготовленных словарей, выступали слова и знаки пунктуации. Для токенизации предложений был использован метод `word_tokenize` из библиотеки nltk. Чтобы как-то ограничить максимальное количество токенов в предложении в целях оптимизации и уменьшении нагрузки на оперативную

память, был проведён небольшой анализ распределения количества токенов в предложениях. С помощью `Counter` из модуля collections было подсчитано, сколько раз встречается то или иное количество токенов в предложениях:

```
vector_lengths = Counter(len(word_tokenize(line)) for line in
df.lit_text)
```

Построив и проанализировав график по полученным данным, который представлен на рисунке 2, число 40 было выбрано как максимальное количество токенов в предложении.

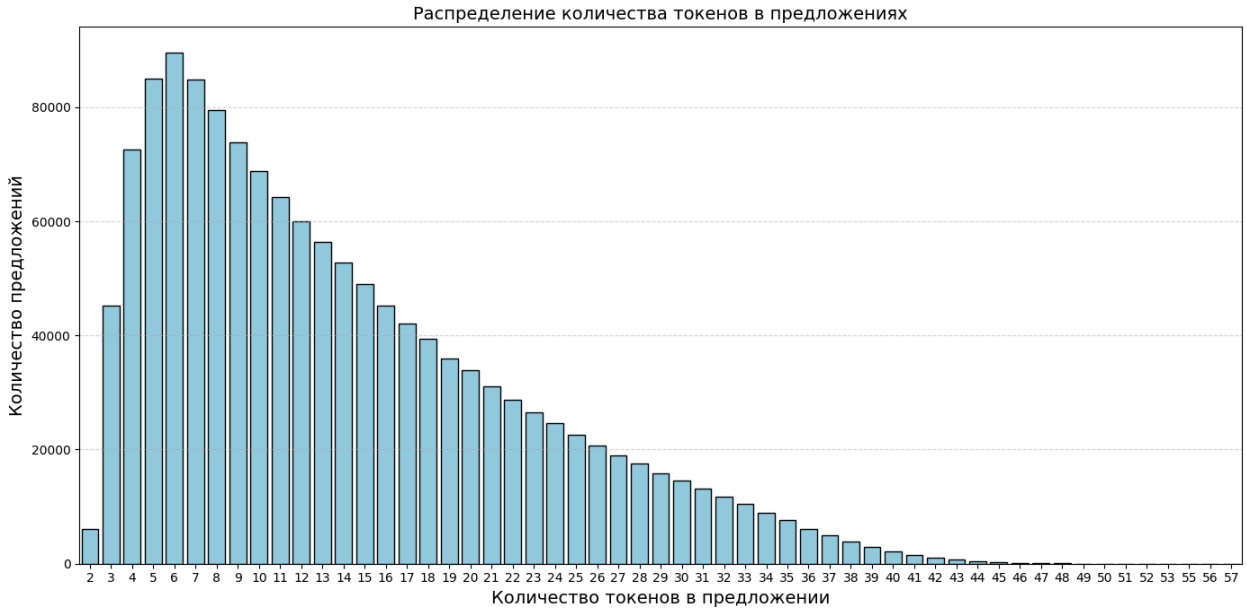

Рисунок 2 — Распределение количества токенов в предложениях

При таком ограничении отсекается очень малая часть предложений и остаётся самая значимая.

4.4 Токенизация и векторизация предложений

Сначала были получены токены для всех предложений художественного стиля, количество токенов в которых меньше выбранного ранее ограничения:

```
lit_tokens = []
for sentence in df.lit_text:
```

```
    tokens = word_tokenize(sentence)
    if len(tokens) > max_vector_length:
        continue
    else:
        lit_tokens.append(tokens)
```

Далее была произведена векторизация предложений по токенам. Размерность векторного представления любого предложения должна быть постоянна, независимо от количества токенов в нём, в нашем случае это `max_vector_length * target_dim`, где `max_vector_length` — количество токенов в предложении, а `target_dim` — длина вектора каждого токена. Если токенов в предложении было меньше максимально возможного количества, то начало вектора заполнялось нулями следующим образом:

```
padding_for = max_vector_length - len(tokens)
vector = [np.zeros(target_dim)] * padding_for
```

В Navec есть соответствующий вектор, доступный по ключу “<pad>”, но во время сжатия размерности векторов Navec он перестал быть нулевым, что может создать ненужные шумы при обучении модели, поэтому было решено заполнять начало векторов именно таким образом.

После этого, итерируясь по каждому токену предложения, сначала происходила попытка добавить сжатый вектор Navec, если для этого токена в словаре нет вектора, происходила попытка добавить вектор из словаря с векторами для знаков пунктуации, если и там не было вектора для этого токена, то добавлялся специальный сжатый вектор “<unk>”, который создан для векторизации всех неизвестных токенов. Далее создавался массив Numpy из списка векторов токенов предложения, и изменялась его размерность, так как они должны быть одномерными:

```
vector = np.array(vector, dtype=np.float16).reshape(max_vector_length
* target_dim)
```

Те же самые действия были произведены и для предложений разговорного стиля. Так как разговорном и художественном стиле было преобразовано в вектора разное количество предложений из-за ограничения максимального количества токенов, нужно было обрезать получившиеся вектора, содержащие все векторизованные предложения, до минимальной длины среди них:

```
length = min(len(lit_vectors), len(tg_vectors))
lit_vectors_np = np.array(lit_vectors[:length])
tg_vectors_np = np.array(tg_vectors[:length])
```

После этого они были сохранены в формате npy, а их вес составил около 1 ГБ. Подробнее с кодом, который использовался для векторизации данных, можно ознакомиться в приложении В.

ЗАКЛЮЧЕНИЕ

Изучив подходы к созданию модели, решающих проблему преобразования изображений и текстов между заданными доменами, можно выделить следующее:

1. CycleGAN использовалась для перевода изображений между доменами, сохраняя их исходные характеристики благодаря циклическим потерям.
2. Стилевой перенос текстов в одной из статей проводился с использованием encoder-decoder и обучения с подкреплением для изменения стиля при сохранении содержания.
3. Имеет место адаптация CycleGAN для работы с текстами на непараллельных корпусах для преодоления разрыва между стилем и содержанием.
4. Метод паттернов переноса стилей и контрастное обучение позволяют улучшить представления текстов и их производительность.

Был определён формат данных, подходящих для обучения: непараллельный датасет, состоящий из предложений как художественного, так и разговорного стиля. Предложения для разговорного стиля были взяты из сообщений 177 наиболее популярных чатов Telegram. Предложения художественного стиля были взяты из литературных произведений, а именно классической прозы. Из всех предложений были убраны все символы, кроме кириллицы и некоторых знаков пунктуации. Слишком короткие и слишком длинные предложения также не вошли в итоговый датасет. Все предложения были токенизированы. В качестве токенов выступали слова и знаки пунктуации. После этого все предложения были векторизованы: каждый токен — вектор из 10 вещественных чисел 16 бит, максимум токенов в каждом предложении — 40, если их было меньше, каждый вектор заполнялся нулями, начиная с начала, ровно на количество недостающих чисел.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Predictions for journalism in 2023: AI and tech // Journalism.co.uk. URL: https://www.journalism.co.uk/news/predictions-for-journalism-in-2023-ai-and-tech/s2/a991916/ (дата обращения: 05.12.2024)
2. Zhu J.-Y., Park T., Isola P., & Efros A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks. Proceedings of the IEEE International Conference on Computer Vision (ICCV), 2223–2232. https://doi.org/10.1109/ICCV.2017.244 (дата обращения: 24.09.2024)
3. Luo, F., Li, P., Zhang, W., Wang, J., & Zhang, M. (2019). Reinforcement learning based text style transfer without parallel training corpus. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers), 3737-3747. https://aclanthology.org/N19-1320.pdf (дата обращения: 07.10.2024)
4. Michela Lorandi, Maram A.Mohamed, and Kevin McGuinness (2023). Adapting the CycleGAN Architecture for Text Style Transfer. Dublin City University. https://dcu-nlg.github.io/publications/adapting-the-cyclegan-architecture-for-text-style-transfer/ (дата обращения: 16.10.2024)
5. Jingxuan Han, Quan Wang, Licheng Zhang, Weidong Chen, Yan Song, Zhendong Mao (2023). Text Style Transfer with Contrastive Transfer Pattern Mining. Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (ACL 2023), 6302–6312. https://aclanthology.org/2023.acl-long.439/ (дата обращения: 03.11.2024)
6. Dolfik. Russian Telegram Chats History [Электронный ресурс] // Kaggle. 2018. URL: https://www.kaggle.com/datasets/dolfik/russian-telegram-chats-history (дата обращения: 04.11.2024)
7. Creative Commons. CC0 1.0 Universal: Public Domain Dedication [Электронный ресурс]. URL:

https://creativecommons.org/public-domain/cc0/ (дата обращения: 08.12.2024)

8. ЛитЛайф. Классическая проза [Электронный ресурс] // Litlife.club. URL: https://litlife.club/genres/56-klassicheskaya-proza (дата обращения: 05.11.2024)
9. Национальный корпус русского языка. Словарь векторных представлений слов [Электронный ресурс]. URL: https://natasha.github.io/navec/ (дата обращения: 12.11.2024)

Приложение А

Сбор данных

```
!pip install selenium
!pip install webdriver_manager


from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import time
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import requests
from bs4 import BeautifulSoup
import os
import zipfile
import re


def get_pages_with_books(url):
    response = requests.get(url)
    soup = BeautifulSoup(response.content, 'html.parser')
    books_links = soup.find_all('a', class_='text-decoration-none')
    books_urls = [link['href'] for link in books_links if ('href' in
link.attrs and re.match(r"^https://litlife\.club/books/\d+$",
link['href']))]
    return books_urls


def download_fb2_file(page_url, download_dir):
    options = webdriver.ChromeOptions()
    prefs = {"download.default_directory": download_dir}
    options.add_experimental_option("prefs", prefs)
    service = Service(ChromeDriverManager().install())
    driver = webdriver.Chrome(service=service, options=options)
    try:
        driver.get(page_url)
        wait = WebDriverWait(driver, 10)
        download_button =
wait.until(EC.element_to_be_clickable((By.XPATH, "//a[contains(@href,
'.fb2.zip')]")))
```

```
        download_button.click()
        time.sleep(15)
        print(f"Файл должен быть загружен в: {download_dir}")
    except Exception as e:
        print(f"Произошла ошибка: {e}")
    finally:
        driver.quit()


download_dir = "data"
count_of_pages = 20
for x in range(count_of_pages):
    main_page_url =
f"https://litlife.club/genres/56-klassicheskaya-proza?page={x}"
    books_url = get_pages_with_books(main_page_url)
    for book in books_url:
        download_fb2_file(book, download_dir)


all_files = os.listdir("data")
for file_zip in all_files:
    with zipfile.ZipFile(f"data/{file_zip}", 'r') as zip_ref:
        for file in zip_ref.namelist():
            if file.endswith(".fb2"):
                try:
                    fb2_file = zip_ref.open(file)
                    fb2_content = fb2_file.read()
                    text_content = ''
                    soup = BeautifulSoup(fb2_content, 'lxml-xml')
                    for para in soup.find_all(['p', 'section']):
                        text_content += para.get_text(separator='\n',
strip=True) + '\n'

                    with open(f"books_txt/{file}.txt", 'w',
encoding='utf-8') as txt_file:
                        txt_file.write(text_content)
                except FileNotFoundError:
                    ...
```

Приложение Б

Небольшая предобработка данных и формирование датасета

```
import json
import numpy
import os
import shutil
import zipfile
from google.colab import drive
import pandas as pd
import re
import nltk
from nltk.tokenize import sent_tokenize


nltk.download('punkt')


from google.colab import drive
drive.mount('/content/drive')


books_path =
"/content/drive/MyDrive/CycleGAN_for_TST_problem/books_txt.zip"
with zipfile.ZipFile(books_path, "r") as zip_ref:
    for filename in zip_ref.namelist():
        zip_ref.extract(filename, "/content/books")


tg_messages_path =
"/content/drive/MyDrive/CycleGAN_for_TST_problem/telegram_comments.zip
"
with zipfile.ZipFile(tg_messages_path, "r") as zip_ref:
    for filename in zip_ref.namelist():
        if "telegram/telegram/" not in filename:
            zip_ref.extract(filename, "/content/tg")


low_chars_border = 8
high_chars_border = 200


lit_data = set()
```

```
all_files = os.listdir("/content/books/books_txt/")
for file in all_files:
    file_txt = open(f"/content/books/books_txt/{file}")
    data_txt = file_txt.read()
    sentences = sent_tokenize(data_txt)
    for sentence in sentences:
        sentence = sentence.strip()
        sentence = re.sub("[^а-яА-Я--–,.!?:«» ]", "", sentence)
        while len(sentence) > 0 and sentence[0] in "--–,.!?:«» ":
            sentence = sentence[1:]
        if low_chars_border < len(sentence) < high_chars_border:
            lit_data.add(sentence)


lit = pd.Series(list(lit_data))


comnts = set()
allFiles = os.listdir("/content/tg/telegram/")
for filename in allFiles:
    if len(comnts) > len(lit) - 1:
                break
    with open(f"/content/tg/telegram/{filename}", 'r') as f:
            groupedComments = json.load(f)
            for comment in groupedComments:
                row = comment["text"]
                sentences = sent_tokenize(row)
                for sentence in sentences:
                    sentence = sentence.strip()
                    sentence = re.sub("[^а-яА-Я--–,.!?:«» ]", "",
sentence)
                    while len(sentence) > 0 and sentence[0] in
"--–,.!?:«» ":
                        sentence = sentence[1:]
                    if low_chars_border < len(sentence) <
high_chars_border:
                        comnts.add(sentence)
                        if len(comnts) > len(lit) - 1:
                            break
                if len(comnts) > len(lit) - 1:
                            break
```

```
com = pd.Series(list(comnts))


data = pd.DataFrame({"lit_text": lit, "tg_text": com})


data.to_csv("/content/drive/MyDrive/CycleGAN_for_TST_problem/data.csv"
, index=False)
```

Приложение В

Векторизация данных

```
!pip install navec


!wget
https://storage.yandexcloud.net/natasha-navec/packs/navec_hudlit_v1_12
B_500K_300d_100q.tar


import pandas as pd
import numpy as np
import torch
from navec import Navec
import nltk
nltk.download('punkt_tab')
from nltk.tokenize import word_tokenize
from sklearn.decomposition import PCA
from collections import Counter
import seaborn as sb
import matplotlib.pyplot as plt


from google.colab import drive
drive.mount('/content/drive')


df =
pd.read_csv("/content/drive/MyDrive/CycleGAN_for_TST_problem/data.csv"
)


path = 'navec_hudlit_v1_12B_500K_300d_100q.tar'
navec = Navec.load(path)


embeddings = list()
for word in navec.vocab.words:
        embeddings.append(navec[word])
embeddings = np.array(embeddings)
```

```
target_dim = 10
pca = PCA(n_components=target_dim)
reduced_embeddings = pca.fit_transform(embeddings)


reduced_navec = {word: np.array(reduced_embeddings[i],
dtype=np.float16) for i, word in enumerate(navec.vocab.words)}


punkt_vectors = {
    ".": np.array([0.0] * (target_dim - 1) + [1.0], dtype=np.float16),
    "!": np.array([0.0] * (target_dim - 1) + [0.9], dtype=np.float16),
    "?": np.array([0.0] * (target_dim - 1) + [0.8], dtype=np.float16),
    ",": np.array([0.0] * (target_dim - 1) + [0.5], dtype=np.float16),
    ":": np.array([0.0] * (target_dim - 1) + [0.6], dtype=np.float16),
    "–": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "-": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "—": np.array([0.0] * (target_dim - 1) + [0.4], dtype=np.float16),
    "«": np.array([0.0] * (target_dim - 1) + [0.2], dtype=np.float16),
    "»": np.array([0.0] * (target_dim - 1) + [0.3], dtype=np.float16),
}


vector_lengths = Counter(len(word_tokenize(line)) for line in
df.lit_text)


lengths = sorted(vector_lengths.keys())
counts = [vector_lengths[length] for length in lengths]
plt.figure(figsize=(14, 7))
data = pd.DataFrame({'Length': lengths, 'Count': counts})
sb.barplot(data=data, x='Length', y='Count', color='skyblue',
edgecolor='black')
plt.xlabel('Количество токенов в предложении', fontsize=14)
plt.ylabel('Количество предложений', fontsize=14)
plt.title('Распределение количества токенов в предложениях',
fontsize=14)
plt.grid(axis='y', linestyle='--', alpha=0.5)
plt.tight_layout()


max_vector_length = 40
```

```
lit_vectors = list()
for i, tokens in enumerate(lit_tokens):
    padding_for = max_vector_length - len(tokens)
    vector = [np.zeros(target_dim)] * padding_for
    for word in tokens:
        try:
            vector.append(reduced_navec[word])
        except KeyError:
            try:
                vector.append(punkt_vectors[word])
            except KeyError:
                vector.append(reduced_navec["<unk>"])
    vector = np.array(vector,
dtype=np.float16).reshape(max_vector_length * target_dim)
    lit_vectors.append(vector)


tg_tokens = []
for sentence in df.tg_text:
    tokens = word_tokenize(sentence)
    if len(tokens) > max_vector_length:
        continue
    else:
        tg_tokens.append(tokens)


tg_vectors = list()
for i, tokens in enumerate(tg_tokens):
    padding_for = max_vector_length - len(tokens)
    vector = [np.zeros(target_dim)] * padding_for
    for word in tokens:
        try:
            vector.append(reduced_navec[word])
        except KeyError:
            try:
                vector.append(punkt_vectors[word])
            except KeyError:
                vector.append(reduced_navec["<unk>"])
    vector = np.array(vector,
dtype=np.float16).reshape(max_vector_length * target_dim)
    tg_vectors.append(vector)
```

```
length = min(len(lit_vectors), len(tg_vectors))


lit_vectors_np = np.array(lit_vectors[:length])
tg_vectors_np = np.array(tg_vectors[:length])


np.save("/content/drive/MyDrive/CycleGAN_for_TST_problem/vectorized_li
t.npy", lit_vectors_np)
np.save("/content/drive/MyDrive/CycleGAN_for_TST_problem/vectorized_tg
.npy", tg_vectors_np)
```