2

# СОДЕРЖАНИЕ

**ВВЕДЕНИЕ 3**

**Глава 1. Погружение в проект 0**

1.1 Выбор темы практики 0

1.2 Описание проекта 0

1.3 Задание на практику 0

**Глава 2. Реализация приложения 0**

2.1 Реализация1 0

2.2 РеализацияN 0

**Глава 3. Описание результатов работы 0**

3.1 Результат1 0

3.2 РезультатN 0

**Глава 4. Описание командной работы 0**

**ЗАКЛЮЧЕНИЕ 0**

**СПИСОК ЛИТЕРАТУРЫ 0**

**ПРИЛОЖЕНИЕ 0**

3.1 Презентация проекта на студенческой научной конференции 0

3.3 Код методов 0

|  |  |  |
| --- | --- | --- |
| Ответственный за соответствие содержания и оформления текста отчёта предъявляемым требованиям | (подпись) | Ступников Андрей Анатольевич |

3

# ВВЕДЕНИЕ

Тип практики: технологическая (проектно-технологическая)

Дата начала: 26.02.2024

Дата окончания: 26.06.2024

Цель практики – получение первичных профессиональных умений и навыков, в том числе первичных умений и навыков научно-исследовательской деятельности, обучающихся по направлению подготовки 02.03.03 Математическое обеспечение и администрирование информационных систем (уровень бакалавриата).

Задачи учебной практики:

* Закрепление теоретических и практических знаний, умений и навыков, полученных на первом и втором курсах обучения;
* Выполнение заданий, предусмотренных программой практики и назначенных руководителем;
* Освоение отдельных компьютерных программ и информационных систем, используемых в профессиональной деятельности;
* Получение навыков работы с периодическими, реферативными и справочными информационными изданиями по прикладной математике и информационным системам;
* Подготовка к осознанному и углубленному изучению дисциплин старших курсов;
* Подготовка и защита в установленный срок отчета по практике.

В результате выполнения технологической (проектно-технологической) учебной практики студент должен обладать следующими компетенциями:

* Анализ и обработка больших данных
* Machine learning
* Создание веб-приложения с помощью фреймворка Flask

4

**Глава 1. Погружение в проект**

**1.1 Выбор темы практики**

Участники нашей команды ранее были знакомы и выполняли совместную работу в рамках хакатона. Нам была интересна сфера машинного и, в частности, глубокого обучения.

Проект основывается на работе с текстом (NLP) – данная тема является одной из наиболее актуальных в сфере машинного обучения. Задача проекта была ясна. Мы владели навыками алгоритмического и объектно-ориентированного программирования, управления проектами, однако мы не владели компетенциями машинного обучения. В ходе практики мы стремились на освоить машинное обучение, веб-разработку, анализ данных.

**1.2. Описание проекта**

**1.2.1 Общее описание проекта**

Связность текста – это признак текста, при котором следующее предложение строится на базе предыдущего с помощью языковых средств [1]. Связность также является важным содержательным критерием, от которого зависит качество его реферирования [2] и сложность восприятия читателем [3]. Инструменты для оценки данного критерия могут найти применение в области обработки естественного языка, в частности при анализе сложности восприятия текста читателем.

Существующие сервисы для английского языка используют различные подходы, включающие семантические сети (WordNet), дистрибутивную семантику (LSA, word2vec), эвристические подходы подсчета повторяющихся слов [4-5]. При этом сервисы для русского языка с открытым описанием их принципа работы используют только последний подход, не учитывая словосочетания производных слов, родовидовые и ссылочные отношения, которые также являются средствами связности [6].

Цель работы – разработать веб-приложение для оценки локальной связности перевода на русский язык с помощью модели машинного обучения, учитывающей синтаксические, лексические и морфологические средства связи.

Задачи проекта:

* Сформировать и разметить датасет.
* Провести сравнительный анализ регрессионных моделей машинного обучения
* Создать приложение, предоставляющее пользователю оценку связности введённого им текста.
* Внедрить модель для оценки локальной связности текста.

**1.2.2. Предметная область**

В этом параграфе надо описать предметную область, т.е. сферу деятельности, процессы, явления, систему, взаимосвязи в условиях чего выполняется проект

…

…

**1.2.3. Постановка задачи**

Дана выборка , элементы которой представляют пары предложений , где – два неповторяющихся предложения и бинарные оценки . D разделяется на На входе модель для обучения получает набор векторных представлений для , dim() = m, , m и набор оценок . На выходе модель выдает набор вещественных оценок , определяющих характер взаимосвязи между значениями целевых переменных и векторных представлений , в котором каждая оценка принадлежит отрезку от 0 до 1. Необходимо построить модель машинного обучения, находящую неизвестное отображение .

**1.2.4. Описание технологий**

Для проведения сравнительного анализа рассматривались модели машинного обучения на базе архитектуры transformer BERT [11] (rubert-base-cased [12] и sbert\_large\_nlu\_ru), градиентного спуска (CatBoostRegressor), байесовского подхода (Bayessian Ridge), метода LASSO (LassoLarsCV) и сетей с долгой краткосрочной памятью LSTM.

Мы освоили использование этих моделей. Среди них были модели (CatBoostRegressor, LassoLarsCV), архитектура которых изучалась в рамках дисциплины АДиОМО. Также встречались модели незнакомой нам архитектуры BERT, LSTM, принципы работы которых достаточно сложны и их изучение выходит за рамки практики.

Для векторизации данных обучающей и тестовой выборки применялись разные виды представлений, такие как: эмбеддинги sentence transformers исходной модели rubert-base-cased, tf-idf, усредненные по эмбеддингам слов векторы elmo, fasttext и word2vec.

Ембеддинг слова (word embedding) – общее название для векторного представления слова из естественного языка. Самый простой способ векторизации – one-hot encoding. Вектор некоторого слова i из словаря S – вектор длинны , где на i-том месте стоит 1, а на остальных позициях 0. Недостаток данного способа заключается в том, что полученные векторы обладают большой размерностью, также вектор слова никак не связан со смыслом слова. Мы не использовали этот способ векторизации напрямую, но он применяется в других методах.

Способ векторизации word2vec использует нейронные сети и основывается на контексте слова в предложении. Для обучения word2vec используются предобработанный текст (или набор текстов). С помощью скользящего окна размера 2n+1 слов, для каждого слова в тексте определяются другие слова, стоящие к нему на расстоянии не более n. Такие слова для исходного слова считаются близкими по контексту и расстояние между их векторами будет невелико. На основании такой разметки обучается нейронная сеть.

S – множество слов из текста (текстов). – множество слов, находящихся в тексте от слова i на расстоянии не более n, где . На вход нейросети word2vec подаётся one-hot encoding вектор слова i, а также правильный ответ в виде one-hot encoding вектора слова . Нейросеть word2vec устроена следующим образом:

Где A – матрица размера B – матрица размера ; j – искомый вектор, состоящий из вещественных чисел в диапазоне [0, 1], отражающий степень контекстной близости слов. Во время обучения . Задача обучения сводится к нахождению коэффициентов матриц A и B. Для каждого эта операция производится раз.

Недостатком word2vec является то, что с его помощью не могут быть представлены слова, не встречающиеся в обучающей выборке. Fasttext построен на схожих идеях, но призван решить проблемы word2vec.

В fasttext слова делятся на n-граммы по буквам скользящим окном, которые поступают на вход нейросети. Вектором слова в таком случае является сумма векторов его n-грамм. Однако такой подход требует больше вычислительных ресурсов.

Word2vec и fasttext не учитывают порядок слов в предложении, в попытке это исправить к задаче векторизации применили идею рекуррентных нейронных сетей. Elmo – модель для векторизации, основанная на архитектуре LSTM. Elmo учитывает не только контекст слова, но и его место в предложении, что помогает разделить векторы многозначных слов.

Более продвинутая технология transformer, применяемая в моделях архитектуры BERT, состоит из пары кодировщик – декодировщик, в каждом из которых используется механизм внимания – метод поиска связи между входными и выходными данными.

TF-IDF – метод векторизации, основывающийся на выделении ключевых

слов в тексте. TF (Частота термина) обозначает, насколько часто определенное слово появляется в данном тексте. IDF (Обратная частота документа) измеряет, насколько уникально слово является по всей коллекции текстов.

- Частота термина для слова "t" в тексте "d".

- Обратная частота текста (документа) для слова "t".

Подход TF-IDF учитывает важность слова в конкретном тексте, относительно других текстов, а также устраняет шум – слова, встречаемые очень часто во всех текстах. Однако такой метод не учитывает семантической информации, контекста и чувствителен к размерам текста.

Для предобработки теста использовались библиотеки NLTK, ruwordnet. NLTK обладает большим функционалом для обработки естественного языка. Мы применяли эту библиотеку для токенизации – выделения предложений в тексте. Библиотека ruwordnet содержит в себе базу синонимов, гипонимов гиперонимов, а также производных слов русского языка.

Для взаимодействия с пользователем мы решили использовать web-приложение. Чтобы его реализовать, мы использовали HTML, CSS, JavaScript с технологией ajax, Python c библиотекой Flask.

Flask – микрофреймворк для создания веб-приложений, предоставляющий базовые возможности: сервер и отладчик, поддержка cookie и сессий, RESTful routing.

HTML – язык разметки гипертекста, предназначенный для описания структуры документа.

CSS – формальный язык для описания внешнего вида документа, предназначен для гибкого и красивого представления структуры документа, а также для разделения логической структуры страницы и ее внешнего вида.

JavaScript – мультипарадигменный язык программирования. В основном используется в браузерах для того, чтобы пользователь мог комфортно взаимодействовать со страницей. Технология ajax используется для фонового обмена данными между пользователем и сервером, что позволяет клиентам отправлять и получать данные без обновления страниц.

**1.3. Задание на практику**

Сформировать и разметить датасет на основе 3503 текстов публицистической и информационной направленности корпуса несовершенных переводов [1]. Тексты представляют собой студенческие переводы с английского на русский язык со средней длиной в предложениях 26,02, в словах – 437,30. Требуется разбить тексты на предожения, перемешать предложения из разных текстов и разбить предложения на пары. Для каждой пары предложений вычислить значение целевой переменной.

Провести сравнительный анализ: обучить регрессионные модели, вычислить метрики MSE, MAE, выявить модель с лучшими метриками. Использовались регрессионные моделей машинного обучения: rubert-base-cased, sbert\_large\_nlu\_ru, CatBoostRegressor, Bayessian Ridge, LassoLarsCV, LSTM. При проведении сравнительного анализа требуется использовать различные способы представлений: эмбеддинги sentence transformers, tf-idf, elmo, fasttext, word2vec. Найти оптимальные гиперпараметры модели.

Создать приложение, предоставляющее пользователю оценку связности введённого им текста. Требуется описать структуру и стили документа, разработать beackend и script для отправки формы.

Внедрить модель оценки локальной связности текста, для последующего использования в приложении. Требуется интегрировать обученную модель в веб-приложение.

**Глава 2. Разработка**

**2.1. Архитектура приложения**

Файловая структура приложения:

* static
  + css
    - app.css
    - bootstrap.min.css
  + js
    - app.js
    - jquery-3.4.1.min.js
* templates
  + index.html
* app.py
* config.py
* process\_sent\_pairs.py
* best\_model

app.css – файл с описанием используемых стилей для index.html.

bootstrap.min.css – библиотека стилей и изображений.

app.js – файл, реализующий технологию ajax для отправки формы с предложениями (документа, текста), без обновления страницы.

jquery-3.4.1.min.js – библиотека ajax.

index.html – файл отвечающий за структуру основного документа.

app.py – основной файл откуда запускается web-приложение, здесь описаны get запросы, для получения пользователем основной страницы и скачивания csv файла с ответом на запрос пользователя. Также здесь описаны post запрос, для обработки формы с предложениями, отправленных пользователем.

config.py – файл с описанием параметров приложения.

prosecc\_sent\_pairs.py – файл с функциями оценки пар предложений по критериям связности.

best\_model – файл, содержащий данные обученной модели *rubert\_base\_cased .*

**2.2. Формирование датасета**

Нами был сформирован датасет на основе 3503 текстов публицистической и информационной направленности корпуса несовершенных переводов [7]. Исходные тексты с помощью токенизатора библиотеки NLTK [8] были разбиты на предложения, затем предложения были перемешаны в случайном порядке. Такой подход нужен: во-первых, потому что мы собираемся вычислять связность для пар предложений; во-вторых, потому что в исходном тексте пары предложений будут связаны всегда, тогда как для обучения моделей требуются и не связанные предложения тоже. На основе полученного набора предложений был составлен список пар соседних предложений объемом 91169 элементов.

Для получения столбца целевой переменной были реализованы 5 методов поиска отношений связности между словами внутри одной пары предложений на языке Python. Каждый метод возвращает 1 при успешном поиске и 0 в случае неуспешного. Признаками наличия отношений [6] являются случаи, когда одно из слов для другого является:

1. гипонимом () — понятие, выражающее частную сущность по отношению к другому, более общему понятию;
2. гиперонимом () — слово с более широким значением, выражающее общее, родовое понятие;
3. производным () — слово, образованное, произведённое от какого-либо другого слова или словосочетания;
4. формой слова ();
5. отсылкой (ссылкой) для другого ().

Общим этапом в работе реализованных методов является разбиение предложений на токены.

Методы для нахождения признаков 1–3 используют библиотеку ruwordnet [9], предоставляющую интерфейс для работы с одноименным тезаурусом. Он хранит информацию о синсетах – группах синонимичных слов, бинарно соответствующих другим группам посредством разметки: «гипонимы–гиперонимы», «производные–производящие» и т. д. Для поиска синсетов токены были приведены к нормальным формам, исключены знаки пунктуации. Условием окончания работы методов являлось наличие нормальной формы конкретного токена в плоском списке синсетов соседнего предложения.

Признаки 4 – 5 рассчитываются с использованием модуля stem библиотеки NLTK и морфологического анализатора pymorphy2 [10]. Для токенов соседних предложений создаются 2 списка стемов (основ слов) . Если , то метод возвращает 1, в противном случае – 0. Метод поиска возможной анафорической связи с помощью экземпляра морфологического анализатора MorphAnalyzer находит тэги частей речи, числа и рода для токенов без знаков пунктуации. Если в списке токенов второго предложения есть местоимение, совпадающее по роду и числу с существительным из первого списка, метод возвращает 1, в противном случае 0.

На основе признаков по формуле: формируется значение целевой переменной связности. Два предложения считаются связными, если целевая переменная равна 1, и не являются связными, если целевая переменная равна 0.

Итоговый набор содержит 13002 пары предложений ввиду того, что при отборе учитывались только пары, в которых представлен только один из признаков или ни одного. Таким образом, в датасете 6501 пара имеет хотя бы одно средство связности, другая половина – нет. Данные обучающей и тестовой выборок делятся в пропорции 80:20, для избежания предвзятости модели в пользу одного из классов, классы в выборках были сбалансированы.

Для обучения моделей требовалось перевести пары предложений в векторы. Эмбеддинги sentence transformers, tf-idf предоставляли готовый функционал для векторизации предложений, тогда как модели elmo, fasttext, word2vec предоставляли готовый функционал только для векторизации слов из предложения. Здесь перед нами возникла задача: требуется из эмбеддингов каждого слова в предложении получить эибеддинг для предложения. Эмбеддинг предложения должен отражать смысл предложения. Есть два подхода к решению этой проблемы: суммировать эмбеддинги отдельных слов, усреднять эмбеддинги отдельных слов. Мы реализовали оба этих способа и заметили, что суммирование хуже работает для предложений разной длинны. Таким образом модуль вектора длинного предложения будет в разы больше, чем вектор короткого предложения. Требуется, чтобы векторы предложений с одним смыслом были близки к друг другу (для определения дистанции используется косинусное расстояние). В следствии чего был выбран метод усреднения эмбеддингов отдельных слов для получения эмбеддингов предложений. При таком подходе векторы схожих по смыслу предложений будут близки, независимо от длинны этих предложений.

Нужно что-то сюда дописать. Пока слишком мало.

**Глава 3. Результаты работы**

**3.1. Полученные метриики**

Наилучший результат по метрикам показала rubert\_base\_cased со следующими гиперпараметрами: число эпох – 1, скорость обучения – , размер батча – 8. Дообучение происходило всего на 1 эпохе, т. к. на последующих эпохах качество снижалось из-за переобучения.

*Таблица 2*

**Сравнение качества обучения моделей на тестовой выборке**

|  |  |  |
| --- | --- | --- |
| Название модели | *mse* | *mae* |
| *rubert\_base\_cased* | **0.167** | 0.335 |
| ai-forever/sbert\_large\_nlu\_ru | 0.171 | **0.333** |
| BayesianRidge на эмбеддингах elmo длиной 1024 | 0.216 | 0.431 |
| LassoLarsCV на эмбеддингих elmo длиной 1024 | 0.219 | 0.435 |

*Окончание таблицы 2*

|  |  |  |
| --- | --- | --- |
| LassoLarsCV на эмбеддингах sentence transformers модели rubert\_base\_cased | 0.219 | 0.435 |
| BayesianRidge на эмбеддингах sentence transformers модели rubert\_base\_cased | 0.217 | 0.435 |
| CatBoostRegressor на эмбеддингах tf-idf | 0.227 | 0.439 |
| CatBoostRegressor на эмбеддингах fasttext | 0.22 | 0.43 |
| LSTM на эмбеддингах word2vec | 0.23 | 0.455 |

**3.2. Веб-приложение**

Для демонстрации работы модели было разработано веб-приложение с использованием фреймворка Flask.

Фронтенд написан с использованием HTML, CSS и JS. Модель интегрируется в приложение с помощью библиотеки simpletransformers [13] Взаимодействие с пользователем осуществляется через точки API посредством HTTP GET и POST запросов. Для получения предсказаний пользователь выбирает текстовый файл посредством диалогового окна, по кнопке “Загрузить файл” файл сохраняется в хранилище для его дальнейшего использования моделью. После нажатия “Получить предсказание” начинается процесс обработки данных моделью, после завершения на экран также выводятся средние значения признаков. Пользователь также может скачать отчет в формате csv. Интерфейс приложения представлен на рис. 3.

A screenshot of a computer

Description automatically generated

*Рис. 3.* Интерфейс приложения

**3.3. Тестирование**

Во время тестирования мы обнаружили, что модель *rubert\_base\_cased* при оценки связности не учитывает значение многозначных слов. Так предложения «Как приготовить автомобиль к поездке?» и «Нужно его пожарить.» модель считает связными, не учитывая другое значение слова «приготовить»