Министерство науки и высшего образования РФ
Федеральное государственное автономное
образовательное учреждение
высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт космических и информационных технологий
институт
Информатика
кафедра

УТВЕРЖДАЮ
Заведующий кафедрой
_____ А.С. Кузнецов
подпись
«____» _____ 2023г.

БАКАЛАВРСКАЯ РАБОТА

09.03.04 – Программная инженерия
код – наименование направления

Разработка программного модуля для формирования и классификации типовых замечаний на выпускаемую продукцию
тема

Руководитель	_____	доцент, кандидат технических наук	К.В. Богданов
	подпись, дата	должность, ученая степень	инициалы, фамилия
Выпускник	_____		А.К. Никитин
	подпись, дата		инициалы, фамилия
Нормоконтроллер	_____	доцент, кандидат технических наук	К.В. Богданов
	подпись, дата	должность, ученая степень	инициалы, фамилия

Красноярск 2023

РЕФЕРАТ

Выпускная квалификационная работа по теме «Разработка программного модуля для формирования и классификации типовых замечаний на выпускаемую продукцию» содержит 65 страниц текстового документа, 16 использованных источников, 27 иллюстраций, 8 формул, 14 таблиц, 1 листинг.

NLP, КЛАСТЕРИЗАЦИЯ ТЕКСТОВ, КЛАССИФИКАЦИЯ ТЕКСТОВ, РЕКУРРЕНТНЫЕ НЕЙРОННЫЕ СЕТИ, DBSCAN, FASTTEXT.

Цель работы – разработать автоматизированный модуль, проводящий кластеризацию текста для формирования типовых замечаний и классификацию текста для присвоения меток замечаниям.

В соответствии с целью ставятся следующие задачи:

- изучить литературу по обработке естественного языка, решению задачи кластеризации и классификации;

- провести анализ предметной области и входных данных;

- выбрать модель векторизации замечаний;

- выбрать и подобрать набор оптимальных параметров алгоритма кластеризации замечаний;

- разработать и обучить модель классификации замечаний;

- разработать автоматизированный модуль анализа замечаний.

В результате выпускной квалификационной работы был создан и передан заказчику модуль, осуществляющий интеллектуальный анализ замечаний.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

ООО «РН-КрасноярскНИПИнефть» разрабатывает проектно-сметную документацию (ПСД) к своим проектам. Для удобства составления и хранения документация компанией была разработана информационная система «Система учета замечаний».

Замечание представляет собой текстовый комментарий, оставленный в процессе анализа ПСД работником организации, и набор описывающих его параметров.

В процессе эксплуатации системы накопилось огромное количество замечаний, что сделало затруднительным процесс анализа и работы с ПСД. Существующая функциональность системы не способна адекватно обработать весь массив замечаний, что требует доработки системы.

Итак, целью выпускной квалификационной работы является усовершенствование существующей информационной системы компании ООО «РН-КрасноярскНИПИнефть» с использованием методов машинного обучения посредством внедрения интеллектуальных систем в процесс работы и анализа ПСД.

Поставленную цель можно разбить на две задачи:

1) разработать модели машинного обучения для классификации и кластеризации замечаний;

2) спроектировать и разработать программный модуль, осуществляющий обработку новых замечаний разработанными моделями.

Успешно внедренный программный модуль существенно сократит время, затрачиваемое проектировщиками организации на процесс анализа и работы с замечаниями при составлении ПСД.

1 Анализ предметной области

1.1 Описание «Системы учета замечаний»

Система учета замечания (СУЗ) – модуль информационной системы (разработки ООО «РН-КрасноярскНИПИнефть»), предназначенный для организации процесса устранения замечаний на выпускаемую продукцию, а также для накопления и обмена знаниями внутри и между КНИПИ.

СУЗ предназначен для автоматизации процессов:

- организации отработки замечаний к выпускаемой проектной продукции;
- контроля исполнения поручений в процессе устранения замечаний;
- автоматической загрузки / выгрузки листа коллективной проверки (ЛКП);
- формирования аналитических срезов с возможностью выборки и сортировки замечаний по заданным критериям;
- формирования и отработки чек-листов внутренней проверки ПСД.

Замечание – это компонент, входящий в состав системы учета замечаний, фиксирующий ошибку в составлении ПСД в виде текстового комментария. Типовое замечание – группа замечаний, фиксирующая одну ошибку.

Чек-лист – это компонент, входящий в состав системы учета замечаний, предназначенный для самоконтроля и контроля выполнения работниками необходимых действий и проверок при разработке разделов ПСД.

Чек-лист состоит из критериев чек-листов, которые формулируются на основе типовых замечаний. Таким образом, предполагается, что при выполнении всех критериев чек-листов из них будут обработаны все соответствующие типовые замечания.

1.2 Задача обработки естественного языка

1.2.1 Постановка задачи

Обработка естественного языка (NLP) – направление машинного обучения и компьютерной лингвистики, направленное на изучение проблемы синтеза

естественных языков и компьютерного анализа. Основными направлениями обработки естественного языка являются: распознавание речи, генерация естественного языка и понимание естественного языка [1]. В контексте выпускной квалификационной работы первостепенной является задача понимания естественного языка, делающая возможной использование классических моделей и алгоритмов машинного и глубокого обучения на текстовых данных.

Обработка естественного языка, как правило, предполагает выполнение следующих этапов:

а) токенизация;

б) нормализация;

в) фильтрация;

г) векторизация.

Ниже представлено подробное описание каждого этапа.

1. Токенизация – это процесс разделения текста на компоненты – набор токенов (tokens). В качестве таких компонент могут выступать предложения, слова или отдельные символы [1]. Чаще всего разбиение текста на токены происходит по символу-разделителю или знакам пунктуации, однако в ряде языков (таких как японский и китайский) такой подход не может быть применен, и для проведения токенизации используются иные методы.

2. Нормализация. Процесс нормализации предназначен для того, чтобы убрать из исходного текста грамматическую информацию (падежи, числа, глагольные виды и времена, залоги причастий, род и т.п.), оставляя семантическую составляющую [1].

Процесс нормализации может осуществляться двумя алгоритмами – стемминг и лемматизация.

Стемминг – упрощенный алгоритм морфологического разбора слова, оптимизированный под максимально быстрое нахождение префикса, общего для всех грамматических форм заданного слова. Обычно получаемая при стемминге основа включает в себя морфологический корень, вместе с приставкой. У

стеммера всегда есть некоторый процент ошибок, возникающих из особенностей словоизменения естественного языка.

Лемматизация – более комплексный процесс, использующий словарь и морфологический анализ, целью которого является привести слово к его канонической форме – лемме.

3. Фильтрация. При анализе текста алгоритмами машинного обучения его лучше перевести в нижний регистр (за исключением собственных имен и аббревиатур), удалить знаки препинания, удалить неинформативные слова (стоп-слова).

Стоп-слова – слова и фразы, которые можно удалить без потери смысла предложения. Стоп-слова применительно к алгоритмам машинного обучения могут добавлять много шума, поэтому появляется необходимость избавляться от нерелевантных слов.

4. Векторизация и извлечение признаков

Алгоритмы машинного обучения не могут работать напрямую с текстом, поэтому необходима векторизация, т.е. преобразование текста в массив чисел, и извлечение признаков [1]. Векторное представление слова и предложения в литературе часто называется эмбеддингом.

В представленных ниже главах рассмотрены основные техники векторизации слов.

1.2.2 Методы, основанные на частотности слова

1.2.2.1 Bag-of-words

При использовании данного подхода каждому тексту сопоставляется вектор размерности словаря корпуса текстов, что описывается формулой (1). Под словарём подразумевается множество всех слов, входящих в корпус текстов.

$$\mathrm{d} \rightarrow \mathrm{v} \in \mathbb{R}\mathrm{m}: \mathrm{vi} = \mathrm{nwi}, \mathrm{wi} \in \mathrm{W}, \mathrm{i} = 1 \ldots m, \quad (1)$$

где d – текст,

$v \in \mathbb{R}$ – последовательность векторов размерности m,

m – размер словаря,

w – слово из словаря W,

n_w – количество вхождений слова w в текст d.

При всей простоте реализации данный подход имеет ряд недостатков:

- для большого корпуса текстов размерность словаря, а, следовательно, и размерность вектора, представляющего текст, может исчисляться сотнями тысяч, а иногда и миллионами параметров;

- не учитывается контекст слова в тексте [2].

Bag-of-ngrams – модификация метода Bag-of-words, подсчитывающая N-граммы – последовательности из N слов.

1.2.2.2 TF-IDF

TF-IDF (Term Frequency – Inverse Document Frequency) – статистическая мера, используемая для оценки важности слова в контексте. Идея данного подхода заключается в том, что словам, которые чаще встречаются в одном тексте и реже в других текстах, следует придавать большее значение, поскольку они могут быть более значимыми для анализа. Важность увеличивается пропорционально тому, сколько раз слово появляется в документе, но компенсируется частотой появления слова в корпусе.

Метрика TD-IDF вычисляется по формуле (2).

$$TF - IDF(w, d, D) = TF(w, d) \cdot IDF(w, D), \quad (2)$$

где w – слово из словаря,

d – текст из множества текстов D,

TF – частота слова, оценивает важность слова w в пределах отдельного текста согласно формуле (3),

IDF – обратная частота текста, уменьшает вес широко употребляемых слов, что показывает формула (4).

$$TF(w,d) = \frac{n_w}{\sum_{i=1}^{m} n_{wi}}, \quad (3)$$

где n_w – количество вхождений слова w в текст,

d, m – размер словаря.

$$IDF(w,D) = \log\frac{|D|}{|\{d \in D \mid w \in d\}|} \quad (4)$$

где $|D|$ – количество текстов в коллекции,

$|\{d \in D \mid w \in d\}|$ – число текстов из коллекции D, в которых встречается слово w [2].

1.2.3 Методы, основанные на глубоком обучении

1.2.3.1 Модель векторизации BERT

BERT (Bidirectional Encoder Representations from Transformers) – это модель глубокого обучения, которая используется для решения задач обработки естественного языка. Она была разработана и опубликована в 2018 году командой исследователей из Google.

Схематичное отображение архитектуры BERT представлено на рисунке 1.

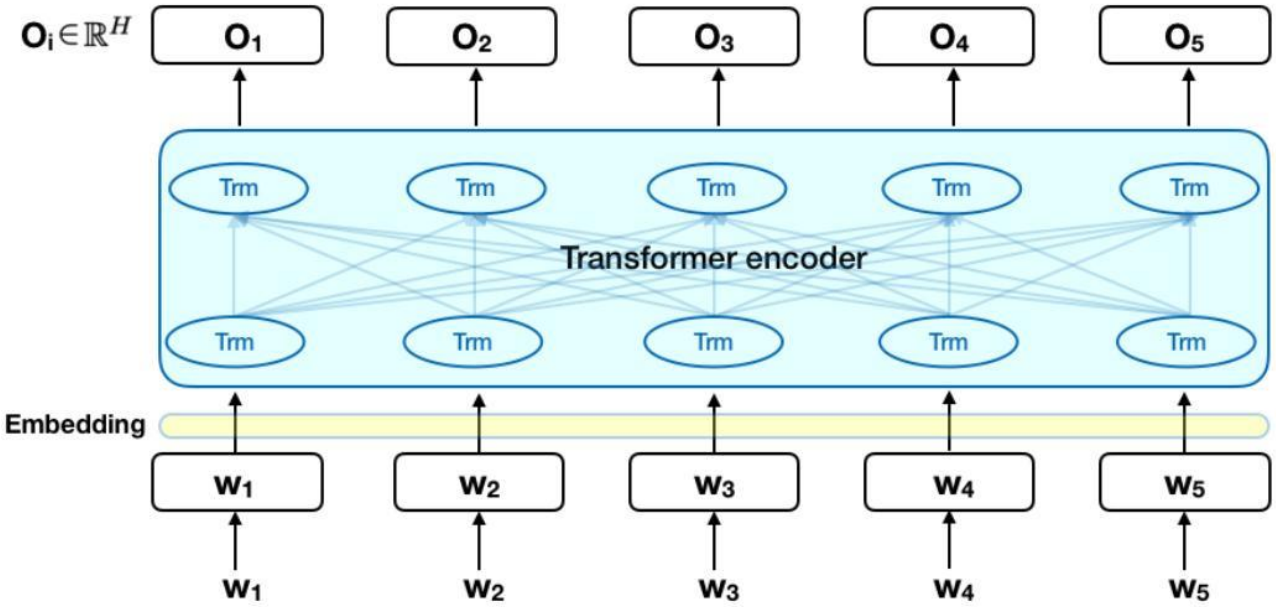

Рисунок 1 – Устройство модели BERT

BERT использует технологию трансформеров (Transformers) для обработки текста. Трансформеры – это архитектура нейронных сетей, которая позволяет моделировать длинные зависимости между элементами последовательности, такими как слова в предложении.

С точки зрения структуры и методов обучения трансформер выглядит аналогично энкодер-декодеру на основе рекуррентных сетей. На вход энкодера подаётся последовательность слов на исходном языке. Векторы этих слов, пройдя через последовательность слоёв самовнимания (Self-Attention), меняются с учётом контекста всего предложения [3].

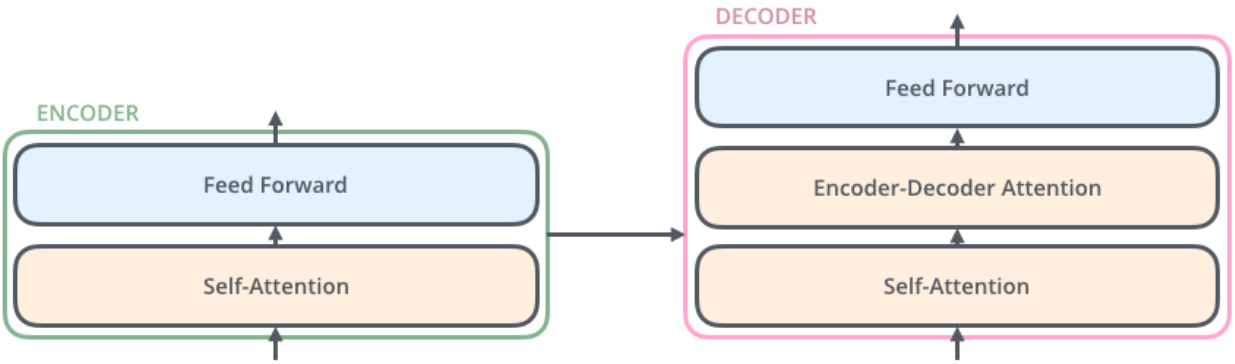

Рисунок 2 – Архитектура модели Transformers

На рисунке 3 представлено описание структуры энкодера трансформера. На его вход подаётся тензор формы (N, B, E), где N – число слов во входной последовательности, B - число одновременно обрабатываемых примеров (батч) и E – размерность их векторов эмбединга. Этот тензор пропускается через функцию Self-Attention. Результат складывается с исходным тензором и нормируется. Получившийся тензор поступает в полносвязный слой (Feed Forward) с двумя линейными преобразованиями ReLU.

Выход этого слоя снова суммируется с его входом и нормируется. Подобные вычисления повторяются несколько раз. На выходе последнего блока получается тензор исходной формы (N, B, E), который описывает слова последовательности с учётом контекста всего текста [3].

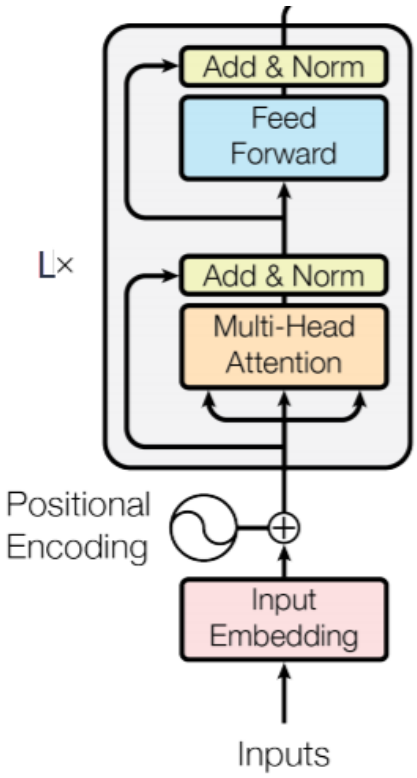

Рисунок 3 – Архитектура энкодера модели Tranformer

На рисунке 4 представлено описание структуры декодера трансформера. На вход декодера подаются слова целевого предложения. Эти слова векторизуются с отличным от энкодера эмбедингом, и к ним добавляются векторы номера позиции слова (positional encoding).

Затем векторы, аналогично энкодеру, проходят блок Self-Attention, для уточнения контекстного смысла векторов с наложением маски на эмбеддинг (Masked Multi-Head Attention). Выход блока Self-Attention суммируется с его входом и нормируется.

После этого включается механизм Self-Attention на словах предложения исходного языка вслед за их обработкой энкодером. При этом запросами являются слова декодера, а в качестве ключей и значений выступают векторы энкодера (*K*, *V* на рисунке 4). Выход снова суммируется со входом и нормируется.

Завершает блок декодера полносвязная сеть (Feed Forward) из двух слоёв аналогично структуре энкодера. Декодер имеет несколько последовательных блоков Feed Forward [3].

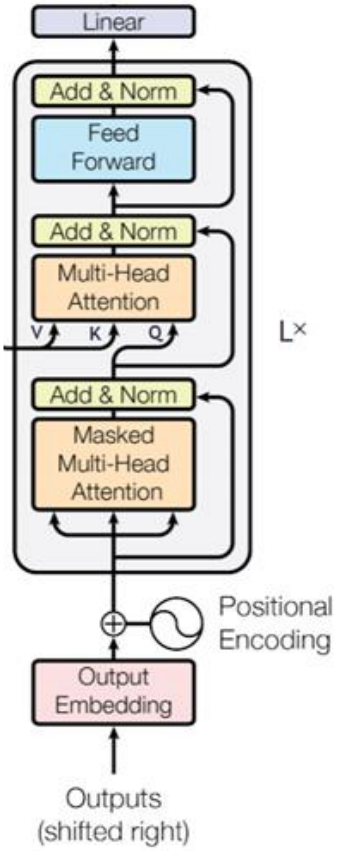

Рисунок 4 – Архитектура декодера модели Transformer

На выходе декодера находится полносвязный слой с числом нейронов равным размеру словаря.

1.2.3.2 Модель векторизации FastText

Модель FastText является модификацией модели word2vec, придуманной в 2014 году компанией Facebook, чтобы решить проблему обработки незнакомых (out-of-vocabulary) слов.

FastText и Word2Vec имеют схожую функциональность, однако существует одно ключевое отличие, связанное с обработкой отсутствующих слов в словаре. В случае отсутствия слова в словаре, FastText разбивает его на символьные n-граммы. Для каждой n-граммы извлекается соответствующий эмбеддинг из матрицы эмбеддингов n-грамм, и затем эти эмбеддинги усредняются, образуя вектор [8].

Таким образом, FastText дает возможность получить векторное представление для слов, которых нет в словаре. Это позволяет нам вычислять семантическую близость даже для незнакомых слов.

Однако, существуют некоторые недостатки в использовании FastText, представленные ниже.

1. FastText не учитывает весь контекст текста при вычислении его векторного представления. Для получения общего вектора текста необходимо использовать дополнительные методы, такие как усреднение или усреднение со взвешиванием по весам TF-IDF.

2. Векторное представление для отдельного слова в FastText остается одним и тем же независимо от контекста. Это означает, что для слов с множественными значениями, такими как "язык", будет использоваться один и тот же вектор вне зависимости от его различных контекстов [8].

1.3 Задача классификации эмбеддингов

1.3.1 Постановка задачи

Классификация текста позволяет разделить текстовые документы на заранее определенные категории или классы, что облегчает их дальнейшую обработку и понимание. Этапу классификации предшествует преобразование слов или текстов в плотные числовые векторы фиксированной размерности – эмбеддинги.

В списке ниже представлены основные подходы к классификации эмбеддингов.

1. Статистические методы. Этот подход основан на анализе статистических свойств эмбеддингов. В качестве метрики для классификации используется сходство или расстояние между эмбеддингами. При анализе расстояний эмбеддингов с известными классами могут использоваться такие методы, как евклидово расстояние, косинусное расстояние или махаланобисово расстояние. При анализе сходства эмбеддингов одного класса может использоваться метод

ближайшего соседа (Nearest Neighbor) и метод к-ближайших соседей (k-Nearest Neighbors, k-NN).

2. Методы машинного обучения. Этот подход предполагает использование методов машинного обучения для классификации. Для обучения модели классификации на основе данных эмбеддингов могут быть использованы все классические методы классификации. Примерами таких методов являются метод опорных векторов (Support Vector Machines, SVM), наивный Байесовский классификатор (Naive Bayes Classifier), случайный лес (Random Forest) и др.

3. Нейронные сети. В связи с зачастую большим объемом датасета и пространства признаков данных эмбеддингов для классификации эмбеддингов принято использовать модели глубокого обучения. Для этих целей могут применяться многослойные персептроны (Multilayered Perceptron), сверточные нейронные сети (Convolutional Neural Networks, CNN), и, в частности, рекуррентные нейронные сети (Recurrent Neural Networks, RNN). Глубокое обучение позволяет извлекать контекстуальные и семантические особенности эмбеддингов для более точной классификации.

В представленных ниже главах описываются модели многослойного персептрона и рекуррентных нейронных сетей, выбранные для анализа в рамках ВКР.

1.3.2 Многослойный персептрон

Многослойная нейронная сеть (персептрон) — это нейронная сеть, состоящая из входного, выходного и расположенных между ними одного (или нескольких) скрытых слоев нейронов [4].

Многослойный персептрон характеризуется следующими параметрами:

а) размерность входного вектора n вещественных чисел;

б) вид функции активации нейронов;

в) количество скрытых слоев и нейронов в каждом скрытом слое;

г) начальные значения весов *w*.

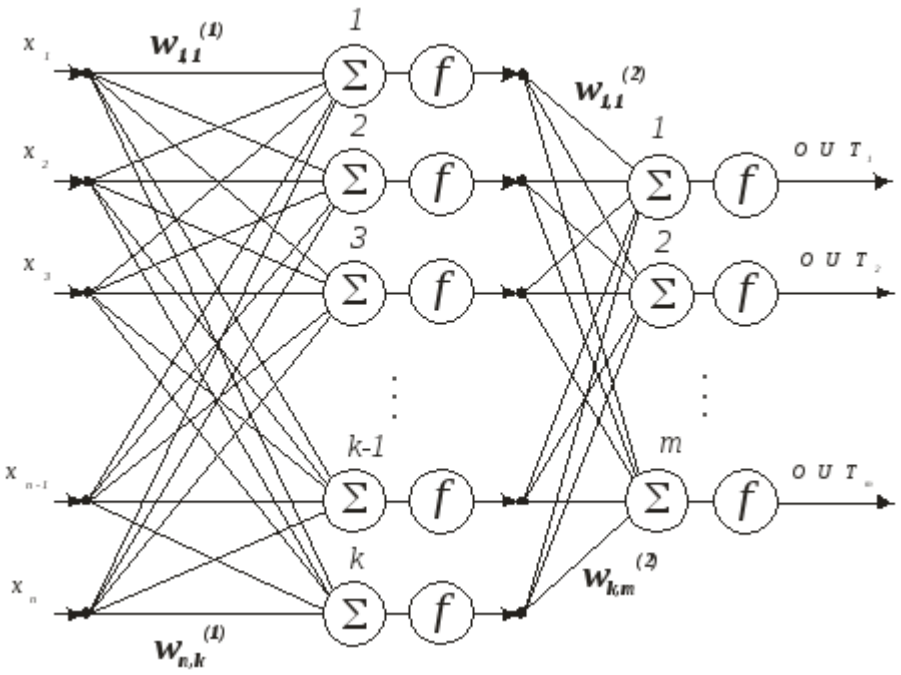

Рисунок 5 – Модель многослойного персептрона

В связи с тем, что входной слой многослойного персептрона является линейным массивом вещественных чисел, его невозможно использовать для классификации предложений, представленных массивом эмбеддингов слов; таким образом, многослойный персептрон способен классифицировать только эмбеддинги предложений.

Ключевой особенностью многослойного персептрона является использование алгоритма обратного распространения ошибки (Backpropagation), состоящего из этапов прямого и обратного прохода.

Вначале нейронная сеть проходит прямой проход, где входные данные подаются на входной слой сети и передаются через последовательные слои с учетом весов и функций активации. Затем сравнивается полученный выход с ожидаемым значением и вычисляется ошибка прогнозирования.

Далее, алгоритм переходит к обратному распространению ошибки. Ошибка передается обратно через слои сети, начиная с последнего слоя и двигаясь к первому. На каждом слое вычисляются градиенты функции ошибки по активациям и по весам.

Вычисленные градиенты используются для обновления весов нейронов сети с помощью градиентного спуска. Алгоритм повторяется на каждой итерации обучения, пока ошибка прогнозирования не достигнет заданного порога или пока не будет достигнуто определенное количество эпох обучения.

1.3.3 Рекуррентные нейронные сети

Рекуррентные нейронные сети (Recurrent neural network, RNN) – вид нейронных сетей, где связи между элементами образуют направленную последовательность. Благодаря этому появляется возможность обрабатывать цепочки данных.

Рекуррентные сети могут использовать свою внутреннюю память для обработки последовательностей произвольной длины, поэтому сети RNN применимы в таких задачах, где нечто целостное разбито на сегменты [4]. Это свойство рекуррентных сетей выгодно выделяет их относительно модели многослойного персептрона, способного к анализу лишь простых данных. Таким образом, сети RNN способны обойти ограничение многослойного персептрона, не позволявшего использование массива эмбеддингов слов для обучения модели.

На рисунке 6 представлена архитектура базовой рекуррентной нейронной сети.

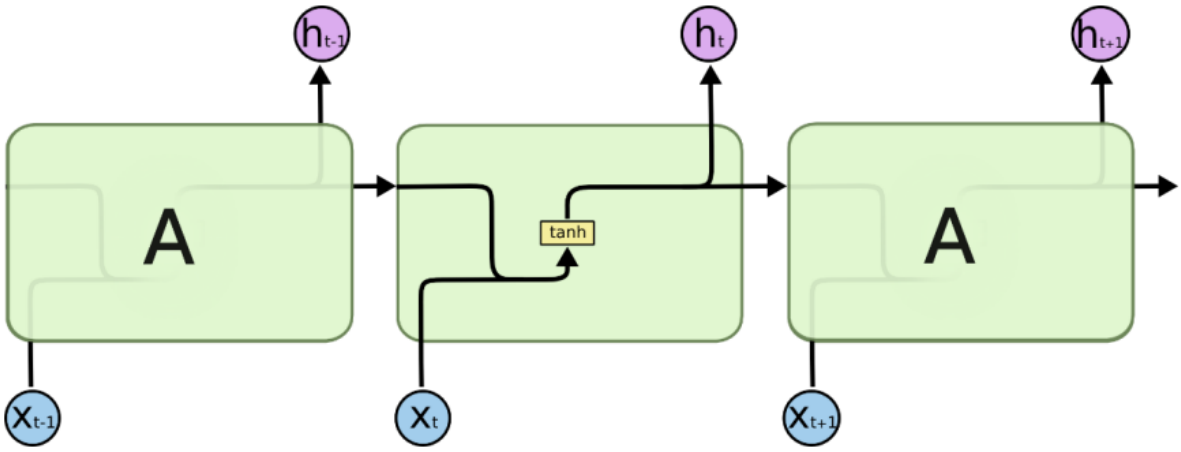

Рисунок 6 – Структура RNN

На рисунке выше участок нейронной сети A получает некие данные X на вход и подает на выход некоторое значение H. Циклическая связь позволяет передавать информацию от текущего шага сети к следующему.

Было предложено много различных архитектурных решений для рекуррентных сетей. Наибольшее распространение получили сеть с долго-краткосрочной памятью и управляемый рекуррентный блок.

Сеть долго-краткосрочной памяти (Long Short Term Memory, LSTM) - особый вид рекуррентной сети, способный к обучению долгосрочным зависимостям [4]. Структура сети LSTM отражена на рисунке 7.

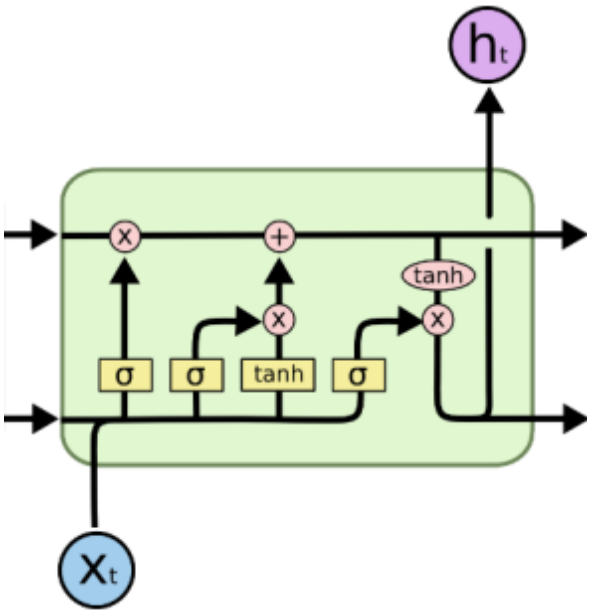

Рисунок 7 – Структура ячейки LSTM

Каждая ячейка памяти LSTM состоит из трех основных компонентов: входного вентиля (input gate), забывающего вентиля (forget gate) и выходного вентиля (output gate). Входной вентиль определяет, какую информацию следует обновить и добавить в ячейку памяти. Забывающий вентиль контролирует, какую информацию следует удалить из ячейки памяти, чтобы избежать накопления старых данных. Выходной вентиль регулирует, какую информацию следует отправить на следующий временной шаг.

Управляемый рекуррентный блок (Gated Recurrent Unit, GRU) – это модификация сети LSTM, предлагает более простую архитектуру, но сравнимую производительность. Она совмещает забывающие и входные вентили в один обновляющий вентиль (update gate) и вносит изменения в их работу. Структура сети GRU представлена на рисунке 8.

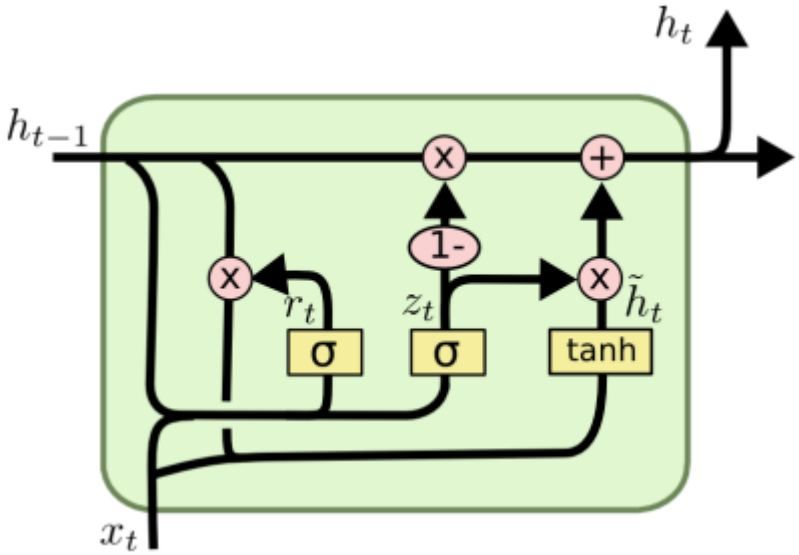

Рисунок 8 – Структура ячейки GRU

1.4 Задача кластеризации эмбеддингов с неизвестным номером кластеров

1.4.1 Постановка задачи

Кластеризация текстовых данных (эмбеддингов) позволяет группировать семантически близкие тексты по конечным наборам непересекающихся групп – кластерам. Зачастую в реальных текстовых данных количество таких кластеров неизвестно заранее либо динамически изменяется в процессе накопления новых данных. В таком случае необходимо применение более узконаправленных алгоритмов кластеризации – кластеризации с неизвестным номером кластеров, в число которых входят:

а) иерархическая кластеризация;

б) DBSCAN;

в) OPTICS;

г) Mean Shift.

В представленных ниже главах описывается семейство иерархических алгоритмов и алгоритмы DBSCAN и OPTICS, выбранные для анализа в рамках ВКР.

1.4.2 Иерархическая кластеризация

Основная особенность алгоритма иерархической кластеризации заключается в построении иерархической структуры кластеров. Этот метод позволяет организовать данные в виде дерева (или дендрограммы), где каждый узел представляет собой кластер, а расстояние между узлами отражает степень их схожести. Алгоритм не требует заранее заданного числа кластеров для своей работы [7].

Существуют два основных подхода в иерархической кластеризации: агломеративный и дивизивный. Их схематичная работа отражена на рисунке 9.

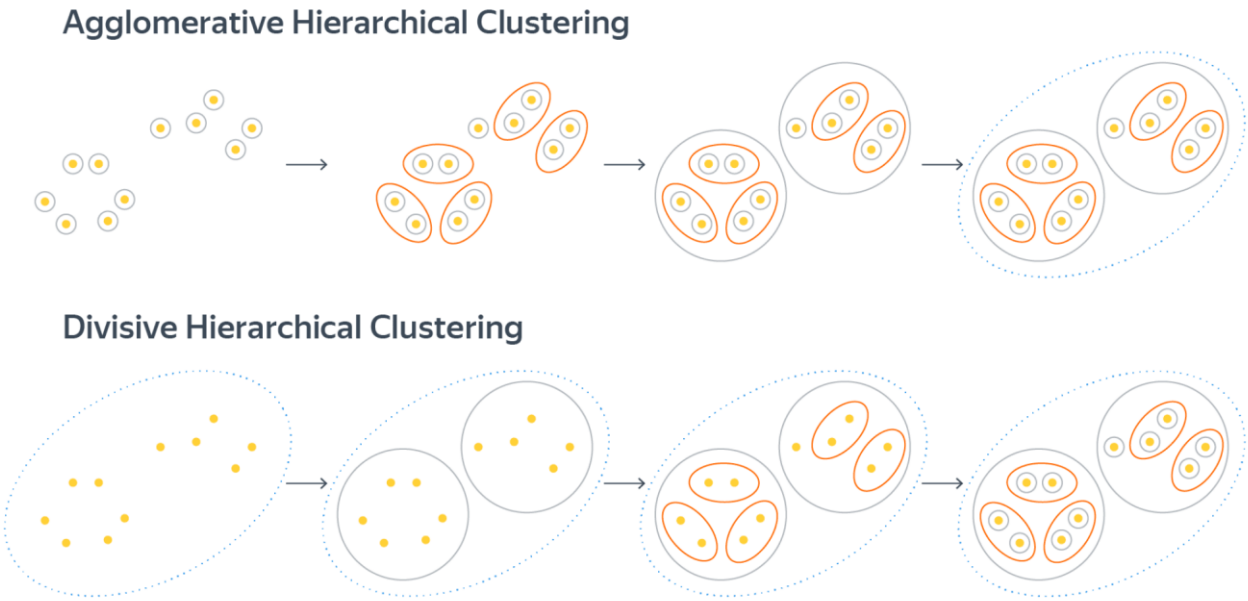

Рисунок 9 – Принцип работы иерархической кластеризации

Агломеративный метод начинает работу с каждого элемента данных в отдельном кластере и затем последовательно объединяет ближайшие точки,

чтобы создать более крупные кластеры. На каждом шаге агломеративной кластеризации находятся два ближайших кластера и объединяются в один новый кластер. Процесс продолжается, пока все элементы не объединятся в один кластер-корень дерева. Условие объединения кластеров представлено формулой 5:

$$D = \min(dist(a, b)), \quad (5)$$

где *dist* – функция поиска расстояния между двумя точками,

a, b – точки в пространстве признаков данных [5].

Дивизивный метод, наоборот, начинает работу с одного общего кластера и последовательно разделяет его на более мелкие составляющие. На каждом шаге дивизивной кластеризации выбирается кластер для разделения, и элементы внутри него распределяются между двумя новыми кластерами. Процесс продолжается до тех пор, пока каждый элемент данных не станет отдельным кластером. Условие разделения кластера представлено формулой 6:

$$D = max(dist(a, b)). \quad (6)$$

1.4.3 DBSCAN и OPTICS

DBSCAN (Density-Based Spatial Clustering Application with Noise) принадлежит к семейству алгоритмов кластеризации на основе плотности, где более плотные области рассматриваются как кластеры, а области низкой плотности называются шумом. DBSCAN не требует, чтобы число кластеров было задано в качестве параметра из-за способа формирования кластеров на основе возможности соединения векторов данных друг с другом [6].

Алгоритм использует два основных параметра: радиус окрестности (ε) и минимальное число точек в окрестности (*MinPts*).

Пусть *D* - множество всех точек данных, p – одна из точек данных в *D*, q – любая другая точка в *D*. Тогда плотностная окрестность точки p определяется следующим образом:

$$\mathrm{N}_{\varepsilon}(p) = \{q \in D \mid dist(p,q) \leq \varepsilon\} \quad (7)$$

где *dist*(*p*, *q*) – функция расстояния между точками *p* и *q*.

Точка *p* считается ядром (core point), если число точек в ее плотностной окрестности не меньше *MinPts*. Если точка p является ядром, то DBSCAN расширяет кластер, добавляя все точки, принадлежавшие множеству $N_{\varepsilon}(p)$ в него. Затем алгоритм рекурсивно рассматривает каждую точку в кластере, проверяя их плотностные окрестности. Если точка является ядром, то ее плотностная окрестность также добавляется в кластер. Процесс продолжается до тех пор, пока не будут рассмотрены все связанные точки данных.

Точки, которые не являются ядрами и недостаточно близки к плотным областям, считаются шумом и не принадлежат ни одному кластеру.

На рисунке 10 продемонстрирован принцип работы алгоритма DBSCAN.

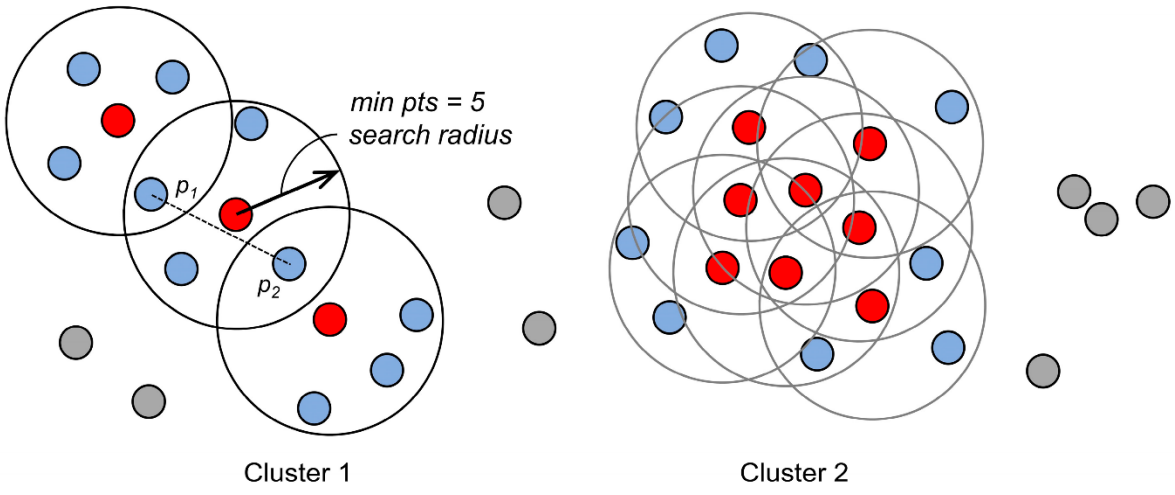

Рисунок 10 – Алгоритм DBSCAN

OPTICS (Ordering Points To Identify the Clustering Structure) является расширением алгоритма DBSCAN, добавляя возможность построения иерархической структуры кластеров.

В отличие от DBSCAN OPTICS в вычислениях оперирует "дистанциями достижимости" (reachability distance) между точками данных. Дистанция достижимости между точками *p* и q определяется следующей формулой:

$$reachability - distance(p,q) = \max(core - distance(q), dist(p,q)) \quad (8)$$

где *core-distance*(q) является корневым показателем плотности точки q,

dist(p, q) - расстояние между точками p и q.

На рисунке 11 представлен принцип работы алгоритма OPTICS.

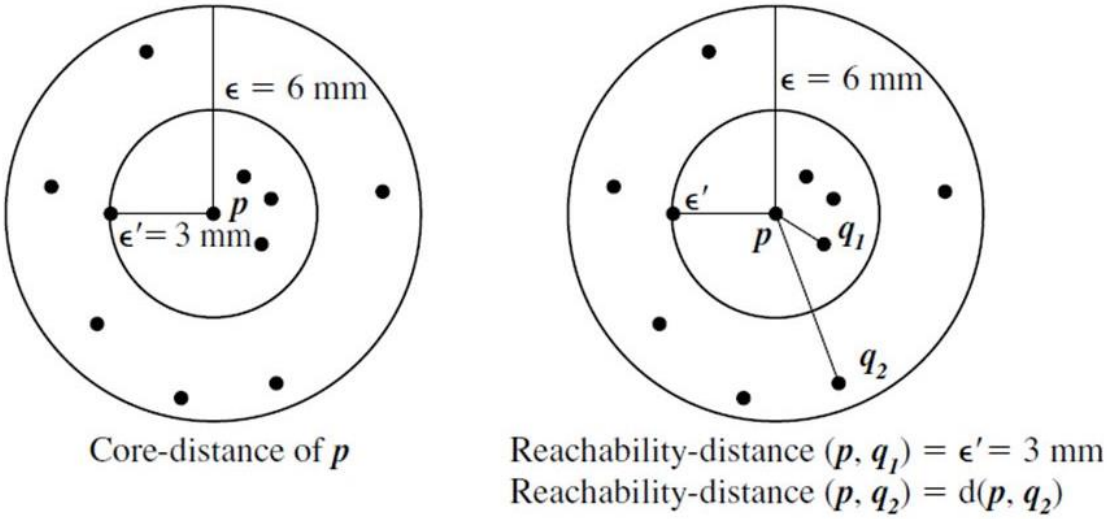

Рисунок 11 – Алгоритм OPTICS

На основе дистанций достижимости OPTICS строит упорядоченный список точек данных, называемый "OPTICS-граф". OPTICS-граф отражает связи между точками данных и их относительные уровни плотности. Более высокие значения дистанции достижимости указывают на точки данных, которые находятся в менее плотных областях.

1.5 Выводы по главе

В главе анализа предметной области затронуто функционирование и основная терминология системы СУЗ компании ООО «РН КрасноярскНИПИнефть».

Был представлен теоретический материал касательно архитектуры и принципов работы, используемых в рамках ВКР алгоритмов и моделей машинного обучения, а именно:

а) алгоритмов и моделей векторизации текстовой информации;

б) архитектур моделей нейронных сетей, способных к классификации эмбеддингов;

в) алгоритмов кластеризации, способных кластеризовать набор эмбеддингов с неизвестным количеством кластеров.

2 Назначение и общая характеристика автоматизированного модуля обработки замечаний

2.1 Проблематика

В «Системе учета замечаний» накопилось большое количество повторяющихся (типовых) замечаний, и их количество продолжает увеличиваться.

Процесс анализа замечаний усложнился. Даже после применения фильтрации остается значительное количество замечаний, который затруднительно прочитать, чтобы сделать выводы и разработать эффективные корректирующие мероприятия.

Из-за этого разработчикам ПСД не хватает времени на анализ замечаний и выявление значимых/типовых замечаний для актуализации критериев чек-листов.

В системе уже реализован алгоритм автоматизированного поиска типовых замечаний. Этот алгоритм очень помогает проектировщикам, но не решает проблему полностью. Недостатки действующего отчета по типовым замечаниям:

1) ориентация на текст замечания, отчего часть аналогичных по смыслу замечаний не попадают в отчет;

2) с накоплением данных отчет становится очень объемным (в ТНИПИ на текущий момент 12 тыс. кластеров замечаний), что сделало проблематичным проводимый в дальнейшем анализ;

3) в отчете есть дубли – похожие по смыслу замечания, но с разными формулировками.

Дополнительная проблема - пользователи при кодировании замечаний делают методологические ошибки и очень часто кодируют неправильно. В одном кластере одинаковые замечания имеют различные коды и значимость.

Для разработки эффективных корректирующих мероприятий требуется кластеризация замечаний, формирование рейтинга кластеров по объему замечаний/ частоте повторений.

Также необходимо построить модель классификации типовых замечаний на конечный набор кластеров-классов.

2.2 Назначение программного модуля

Как было отмечено ранее, основная цель разработки данного программного продукта – повышение эффективности разработки проектно-сметной документации путем оптимизации работы проектировщиков ПСД. В рамках проекта реализуются две бизнес-цели:

1) улучшение поиска типовых замечаний;

2) разработка алгоритма группировки типовых замечаний.

Ниже представлено содержание каждой из упомянутых целей.

«Улучшение поиска типовых замечаний» предполагает замену существующего алгоритма поиска на другой алгоритм, который был бы способен в дополнение к возможностям старого алгоритма учитывать семантический смысл замечания, что позволит улучшить формирование кластеров типовых замечаний.

«Разработка алгоритма группировки типовых замечаний» представляет собой обучение модели классификации, обученной на выборке, состоящей из пар «Замечание – Метка класса».

Разработанный модуль будет интегрирован в СУЗ, обеспечив доступ к результатам работы моделей в виде SQL-таблиц.

2.3 Необходимые к реализации задачи

Основные задачи, необходимые к реализации в программном модуле, представлены в списке ниже.

1. Разработка системы преобразования текстовых данных замечаний в формат, пригодный для задачи поиска схожих предложений;

2. Подбор и настройка модели кластеризации для кластеризации типовых замечаний;

3. Подбор и настройка модели классификации для разбиения множества типовых замечаний на подмножества, объединенные общей темой;

4. Разработка автоматизированного программного модуля, запускающего вышеописанные алгоритмы.

2.4 Функциональные требования

2.4.1 Автономный запуск модуля

2.4.1.1 Описание

Программный модуль должен иметь два режима работы:

а) немедленный запуск алгоритмов;

б) ежедневный запуск алгоритмов.

Ниже представлено подробное описание каждогона данных из режимов.

Под «немедленным запуском алгоритмов» понимается запуск алгоритмов непосредственно после запуска модуля. При такой конфигурации модуль завершит свое выполнение по истечении работы всех алгоритмов.

Под «ежедневным запуском алгоритмов» понимается запуск алгоритмов ежедневно в определенное время, заданное пользователем, один или более раз за день. Время запуска при этом динамическое и задается пользователем при запуске модуля.

2.4.1.2 Требования к реализации

В таблицах 1-2 представлены функциональные требования запуска модуля.

Таблица 1 – Функциональное требование «Немедленный запуск алгоритмов»

Название	Немедленный запуск алгоритмов
Входные данные	Флаг типа запуска алгоритмов.
Описание	Непосредственный запуск алгоритмов в обычном режиме в главном потоке.

Окончание таблицы 1

Выходные данные	—

Таблица 2 – Функциональное требование «Ежедневный запуск алгоритмов»

Название	Ежедневный запуск алгоритмов
Входные данные	Флаг типа запуска алгоритмов, список со временем запуска.
Описание	Запуск алгоритмов в режиме ожидания указанного пользователем времени дня. Пользователь может передать сразу несколько строк со времени для запуска. Для работы алгоритмов создается отдельный поток.
Выходные данные	—

2.4.2 Обработка замечаний и критериев чек-листа

2.4.2.1 Описание

Преобразует текст замечаний и критериев чек-листа в эмбеддинги (вектора) и записывает результат в SQL-таблицу. Данный функционал системы необходим для ускорения последующего процесса анализа, позволяя заранее рассчитывать и в последствии хранить обработанные сущности системы (замечания и критерии чек-листа), требующие значительных временных затрат на вычисление.

2.4.2.2 Требования к реализации

В таблицах 3-5 представлены функциональные требования обработки замечаний и критериев чек-листа.

Таблица 3 – Функциональное требование «Получение замечаний и критериев чек-листа без эмбеддингов из базы данных»

Название	Получение замечаний и критериев чек-листа из базы данных
Входные данные	SQL-таблицы замечаний и критериев чек-листа, SQL-таблицы эмбеддингов замечаний и критериев чек-листа.

Окончание таблицы 3

Описание	Отправляет запрос к базе данных, получающий необработанный модулем массив замечаний и критериев чек-листа.
Выходные данные	Массив замечаний и критериев чек-листа.

Таблица 4 – Функциональное требование «Расчет эмбеддингов замечаний и критериев чек-листа»

Название	Расчет эмбеддингов замечаний и критериев чек-листа
Входные данные	Массив замечаний и критериев чек-листа.
Описание	Рассчитывает эмбеддинги для каждого замечания и критерия чек-листа.
Выходные данные	Массив эмбеддингов замечаний и критериев чек-листа.

Таблица 5 – Функциональное требование «Запись эмбеддингов в базу данных»

Название	Запись эмбеддингов в базу данных
Входные данные	Массив эмбеддингов замечаний и критериев чек-листа.
Описание	Записывает сведения об эмбеддингах замечаний и критериев чек-листа в соответствующие таблицы эмбеддингов.
Выходные данные	SQL-таблицы эмбеддингов замечаний и критериев чек-листа.

2.4.3 Формирование типовых замечаний

2.4.3.1 Описание

Разбивает множество всех замечаний на подмножества типовых замечаний. Данная операция представляет собой процесс кластеризации множества эмбеддингов замечаний, полученных в результате выполнения функционального требования «Расчет эмбеддингов замечаний и критериев чек-листа».

2.4.3.2 Требования к реализации

В таблицах 6-8 представлены функциональные требования формирования типовых замечаний.

Таблица 6 – Функциональное требование «Получение всех эмбеддингов замечаний из базы данных»

Название	Получение эмбеддингов замечаний из базы данных
Входные данные	SQL-таблица эмбеддингов замечаний.
Описание	Отправляет запрос к базе данных, получающий массив замечаний, для которых были построены эмбеддинги.
Выходные данные	Массив эмбеддингов замечаний.

Таблица 7 – Функциональное требование «Формирование типовых замечаний»

Название	Формирование типовых замечаний
Входные данные	Список эмбеддингов замечаний.
Описание	Объединяет замечания в кластеры типовых замечаний из данных эмбеддингов.
Выходные данные	Массив кластеров типовых замечаний.

Таблица 8 – Функциональное требование «Запись типовых замечаний в базу данных»

Название	Запись типовых замечаний в базу данных
Входные данные	Массив кластеров типовых замечаний.
Описание	Записывает сведения о типовых замечаниях в таблицу типовых замечаний.
Выходные данные	SQL-таблица типовых замечаний.

2.4.4 Классификация замечаний

2.4.4.1 Описание

Разбивает множество замечаний на крупные группы, объединенные общей темой. Данная операция представляет собой процесс классификации множества новых эмбеддингов замечаний, полученных в результате выполнения функционального требования «Расчет эмбеддингов замечаний и критериев чек-листа».

2.4.4.2 Требования к реализации

В таблицах 9-11 представлены функциональные требования классификации замечаний.

Таблица 9 – Функциональное требование «Получение эмбеддингов замечаний без класса из базы данных»

Название	Получение эмбеддингов замечаний без класса из базы данных
Входные данные	SQL-таблица эмбеддингов замечаний, SQL-таблица классов замечаний.
Описание	Отправляет запрос к базе данных, получающий массив замечаний, для которых были построены эмбеддинги и не было спрогнозировано классов.
Выходные данные	Массив эмбеддингов замечаний.

Таблица 10 – Функциональное требование «Прогнозирование классов замечаний»

Название	Прогнозирование классов замечаний
Входные данные	Массив эмбеддингов замечаний.
Описание	Прогнозирует класс замечаний на основе данных эмбеддингов.
Выходные данные	Массив классов замечаний.

Таблица 11 – Функциональное требование «Запись типовых замечаний в базу данных»

Название	Запись классов замечаний в базу данных
Входные данные	Массив классов замечаний.
Описание	Записывает сведения о классах замечаний в таблицу типовых замечаний.
Выходные данные	SQL-таблица классов замечаний.

2.5 Нефункциональные требования

2.5.1 Требования к производительности

Суммарное время работы алгоритмов должно занимать не более 4 часов на сервере заказчика с учетом обработки новых замечаний.

2.5.2 Требования к качеству программного обеспечения

Программный продукт предназначен для использования сервером заказчика, поэтому он должен удовлетворять конфигурации этого сервера.

2.5.3 Требования к безопасности системы

Программному продукту запрещено каким-либо образом пересылать данные компании ООО «РН КрасноярскНИПИнефть» либо использовать их не по непосредственному назначению.

Функционирование программного продукта не должно нести угрозы технической и информационной сети вышеуказанной компании, тем или иным образом препятствовать ее работе или нести прочий злой умысел.

2.5.4 Требования к сохранности данных

Программный продукт не должен считывать и изменять не затрагиваемые в рамках проекта таблицы базы данных.

2.6 Выводы по главе

Цель разработки программного модуля – повышение эффективности анализа проектно-сметной документации экспертами компании ООО «РН КрасноярскНИПИнефть».

Выполнение поставленной цели заключается в выполнении трех задач:

а) разработка алгоритма поиска типовых замечаний методами кластеризации;

б) разработка модели классификации замечаний;

в) разработка автоматизированного программного модуля.

По указанным задачам были составлены функциональные и нефункциональные требования к разрабатываемой системе.

3 Анализ и описание моделей для решения задач кластеризации и классификации замечаний

3.1 Описание исходных данных

3.1.1 База данных СУЗ

Сотрудниками компании ООО «РН КрасноярскНИПИнефть» был сформирован и направлен backup-файл, содержащий обезличенные данные базы данных «Системы учета замечаний», общим размером 1122 МБ.

База данных состоит из 51 SQL-таблицы, из которых для анализа использовались 5 таблиц: «Department», «Remark», «CheckListCriteria», «RemarkRemarks_DepartmentDepartments» и «DepartmentDepartments_CheckListCriteriaCriteria» – где в таблицах «Remark» и «CheckListCriteria» хранится информация о замечания и критериях чек-листа, в таблице «Department» – информация об отделах; остальные таблицы реализуют связь «многие-ко-многим».

Дополнительно стоит отметить, что количество записей таблицы «CheckListCriteria» составляет 3625 единиц, таблицы «Remark» – 221203 единиц, таблицы «Department» – 1230 единиц.

3.1.2 Обучающая выборка

Обучающая выборка представляет собой excel-файл, состоящий из 9 листов, где каждый лист имеет название класса замечания и хранит в себе список соответствующих замечаний.

Размер обучающей выборки – 6418 единиц. Статистика распределения замечаний исходных данных по классам представлена на рисунке 12.

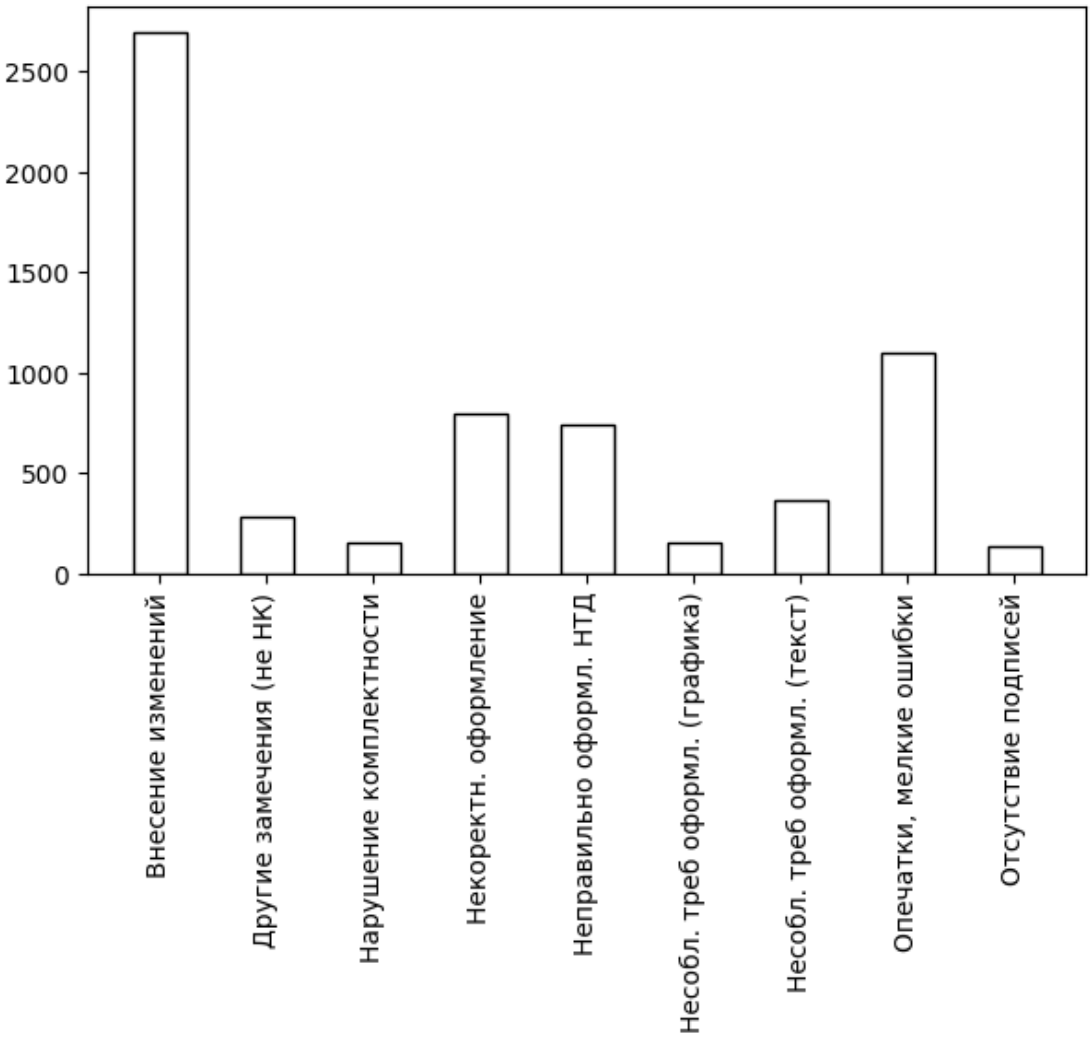

Рисунок 12 – Диаграмма распределения классов

Из диаграммы становится очевиден дисбаланс классов обучающей выборки. Класс «Внесение изменений» составляет приблизительно 42% всей выборки, в то время как классы «Нарушение комплектности», «Несоблюдение требований оформлений (графика)» и «Отсутствие подписей» занимают не более 2.5% набора данных.

В связи со спецификой текстовых данных, выборку невозможно дополнить искусственно сгенерированными данными, а поскольку размер набора данных мал, попытка ликвидации дисбаланса классов посредством удаления наиболее распространенных классов замечаний приведет лишь к снижению точности модели. Таким образом, модель предположительно может склоняться к предсказанию преобладающего класса, игнорируя редкие классы.

3.2 Обработка замечаний

Важнейшим шагом в обработке естественного языка является трансформация токена (слова) в векторное представление – эмбеддинг. В рамках работы было рассмотрено две модели такой трансформации: SBERT с предобученными компанией ПАО «Сбербанк» весами и FastText, обученный на датасете замечаний и критериев чек-листов, – а также алгоритм TF-IDF.

В качестве метрик для сравнения качества моделей использовались время преобразования текста в вектор и точность классификации с использованием наивного байесовского классификатора для TF-IDF и модель многослойного персептрона для SBERT и FastText.

Ниже представлены результаты исследований моделей.

TF-IDF. Точность классификации: 81%. Размерность выходного вектора: 10000. Время обработки 100 замечаний – 0.008 секунды.

FastText. Точность классификации: 85%. Размерность выходного вектора: 128. Время обработки 100 замечаний – 0.2 секунды.

SBERT. Точность классификации: 88%. Размерность выходного вектора: 1024. Время обработки 100 замечаний – 58 секунд.

В связи с тем, что точность алгоритма TF-IDF является слишком низкой, а время работы SBERT существенно превышает допустимый компанией ООО «РН КрасноярскНИПИнефть» предел, в качестве модели построения эмбеддингов был выбран FastText.

3.3 Классификация замечаний

3.3.1 Классификация с помощью многослойного персептрона

Многослойный персептрон является классической моделью для классификации в сфере глубокого обучения, применимый в том числе и для классификации текстов.

Предварительным шагом перед обучением модели многослойного персептрона на корпусе текста является конвертация эмбеддингов слов в

эмбеддинги предложений. Данная операция была реализована получением среднего значения по оси признаков слов, позволив изменить размерность входных данных с (*32*, *128*) до (*128*,) – размерность входного слоя персептрона.

Архитектура получившегося многослойного персептрона представлена на рисунке 13.

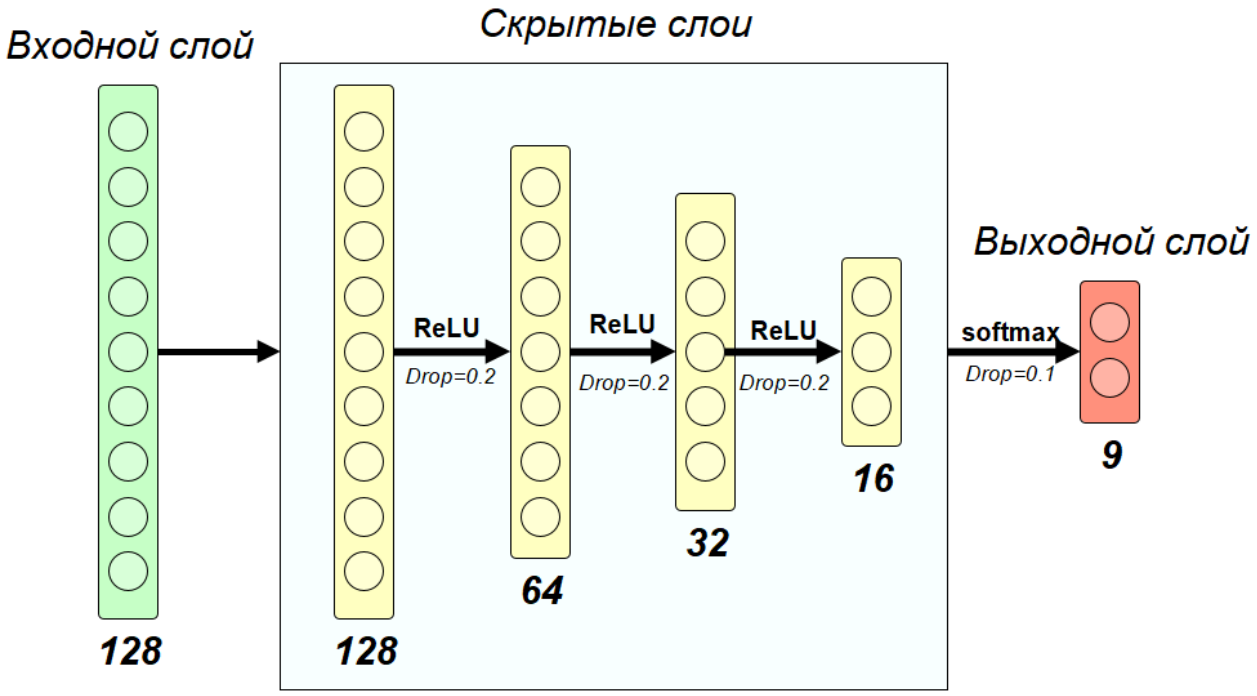

Рисунок 13 – Архитектура обученного многослойного персептрона

Модель включает в себя 6 последовательно расположенных слоев:

а) входной слой со 128 нейронами;

б) линейный слой со 128 нейронами, функцией активации ReLU и коэффициентом Dropout 0.2;

в) линейный слой с 64 нейронами, функцией активации ReLU и коэффициентом Dropout 0.2;

г) линейный слой с 32 нейронами, функцией активации ReLU и коэффициентом Dropout 0.2;

д) линейный слой с 16 нейронами, функцией активации Softmax и коэффициентом Dropout 0.2;

е) выходной слой с 9 нейронами.

Обучение проводилось на 60 эпохах с динамическим коэффициентом скорости обучения. В качестве функции потерь была выбрана категориальная кроссэнтропия. Функцией оптимизации является Adam с начальным коэффициентом скорости обучения, равным 0.02.

На рисунке 14 представлен график изменения коэффициента скорости обучения в процессе обучения, на рисунке 15 – изменения точности на обучающей и валидационной выборках.

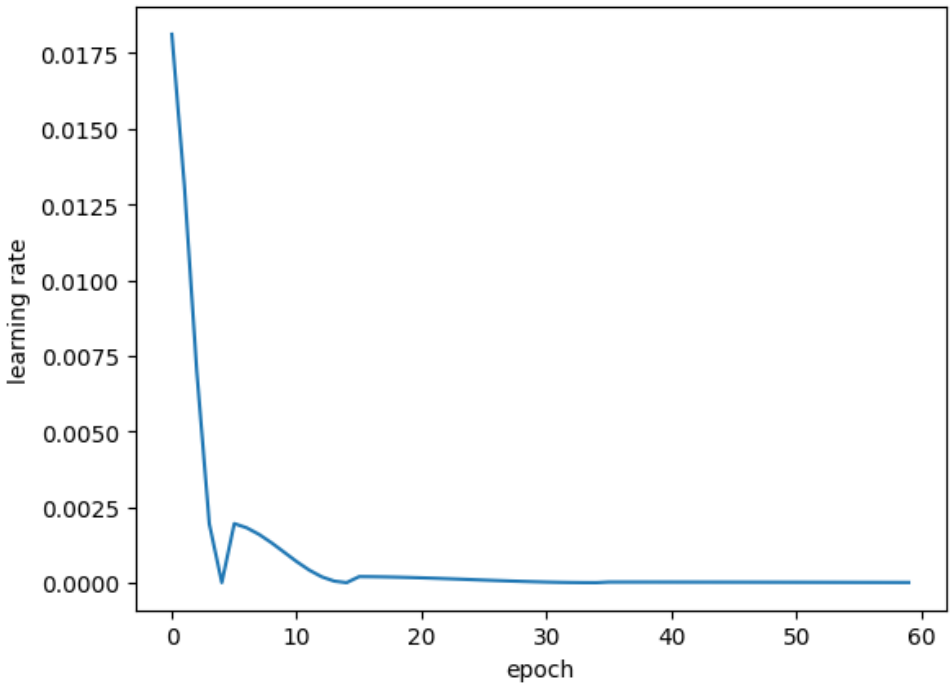

Рисунок 14 – Изменение learning rate при обучении многослойного персептрона

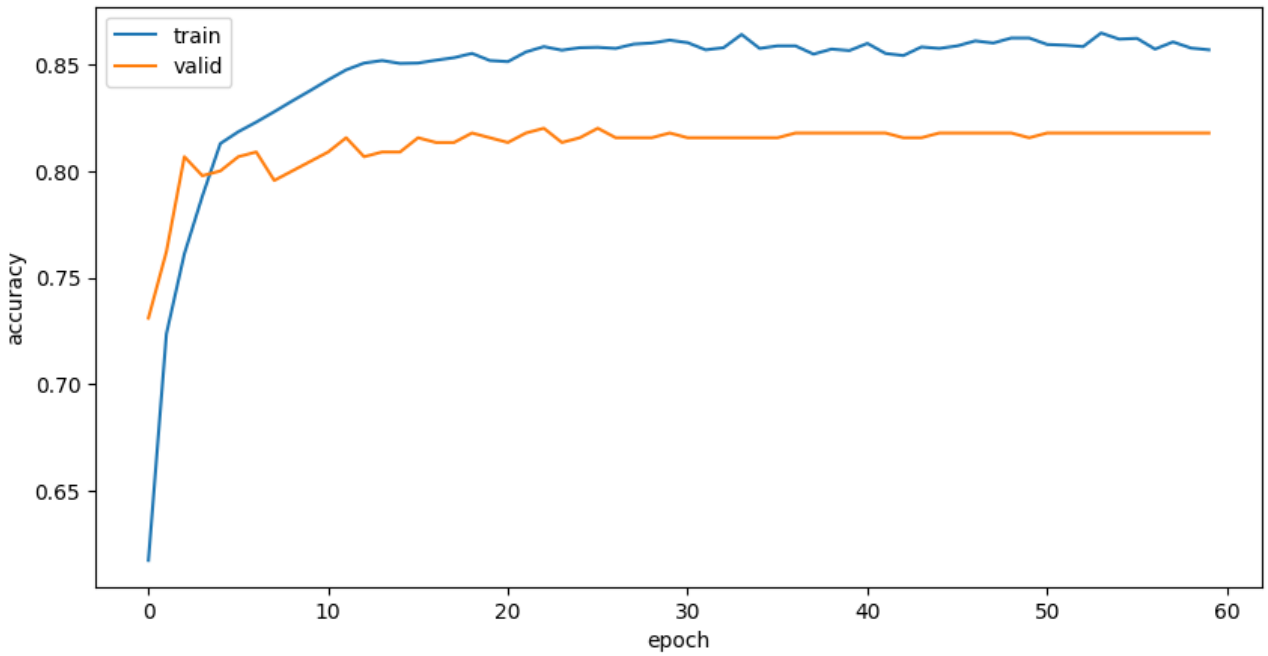

Рисунок 15 – Изменение точности обучающей и тестовой выборки при обучении многослойного персептрона

Разница между точностью классификации на тренировочной и валидационной выборках, равная 7%, указывает на переобучение модели. Попытки избавиться от переобучения путем поднятия коэффициентов Dropout и введением регуляризации при вычислении весов линейных слоев привели лишь к снижению точности валидационной выборки.

Финальная точность модели на валидационной выборке – 88%. На рисунке 16 представлена матрица ошибок с рассчитанными precision (последняя строка), и recall (последний столбец).

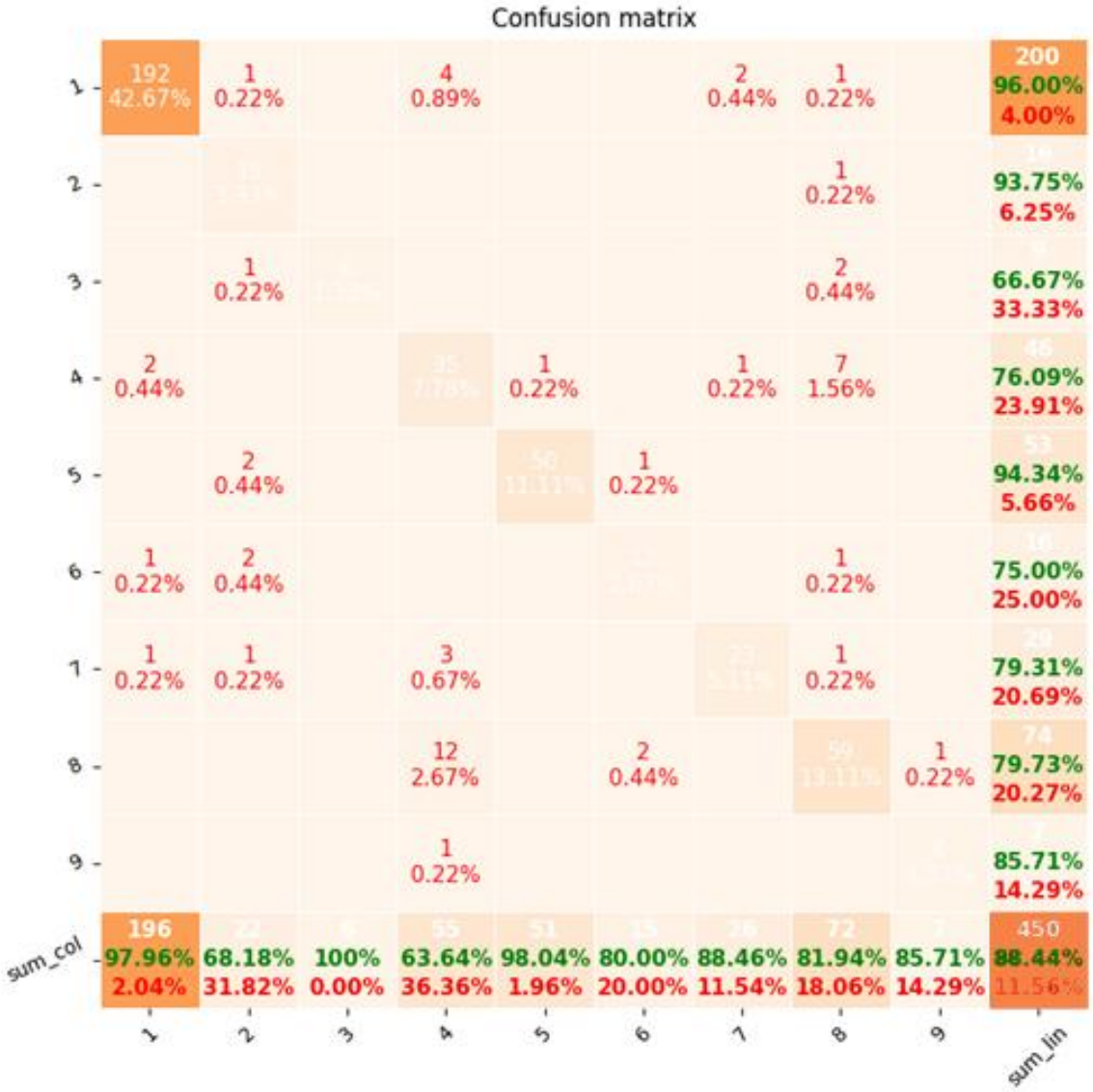

Рисунок 16 – Матрица ошибок классификации многослойного персептрона

Из матрицы ошибок очевидно, что модель персептрона умеет хорошо предсказывать класс «Внесение изменений» (строка 1 на рисунке), который является самым несбалансированным в обучающей выборке. Однако остальные классы модель может предсказать с куда меньшей точностью. В частности, низкой точностью классификации обладает класс «Некорректное оформление» (цифра 4 на рисунке), имеющий precision, равный 64%, и recall, равный 76%, при общем количестве замечаний, равном 35.

Исходя из этих фактов, было принято решение перейти на более сложную архитектуру нейронных сетей – рекуррентные нейронные сети.

3.3.2 Классификация с помощью рекуррентных нейронных сетей

3.3.2.1 Выбор архитектуры рекуррентной нейронной сети

Рекуррентные нейронные сети (RNN), в отличие от модели многослойного персептрона, умеют классифицировать последовательности данных, которой является в том числе и текст. Это позволяет рассматривать предложение не как вектор абстрактных признаков, а как набор взаимосвязанных векторов слов. Модели RNN получают преимущество надо моделями многослойных персептронов в задаче анализа текстовой информации именно по причине способности распознавать и учитывать связи между словами в процессе классификации.

Существует две общепризнанные архитектуры рекуррентных нейронных сетей, которые расширяют базовую структуру RNN: LSTM и GRU. Было проведено исследование, целью которого является нахождение оптимальной архитектуры и параметров модели. Ниже представлен список этих параметров.

1. Архитектура модели. Объектами анализа стала классическая структура RNN (Simple RNN), LSTM и GRU.

2. Количество нейронов в рекуррентном слое.

3. Количество нейронов в слое Time Distributed.

4. Количество нейронов в линейном слое, находящемся непосредственно после рекуррентного.

5. Коэффициент Dropout.

В качестве метрик качества, определяющих качество модели, были использованы точность на валидационной выборке и время обучения (производительность) модели.

Суммарно было обучено 43 модели. Результаты исследований представлены на рисунках 17, 18, 19 и 20.

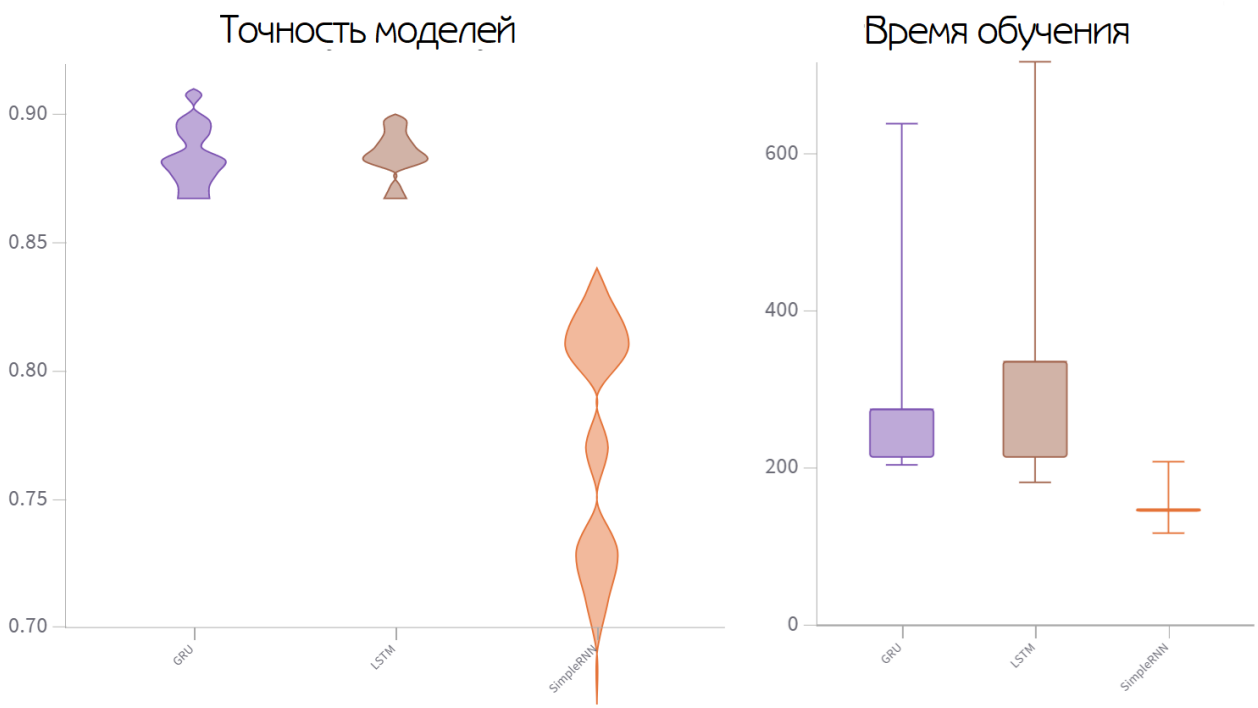

Рисунок 17 – Корреляция архитектуры рекуррентного слоя с точностью и временем обучения

Судя по диаграмме, можно сделать вывод, что использование простого рекуррентного слоя дает небольшой выигрыш во времени работы, но существенный проигрыш в точности всей модели. С другой стороны, архитектуры GRU и LSTM отличаются друг от друга слабо: GRU имеет точность и время работы слегка выше, чем LSTM.

Таким образом, была выбрана архитектура GRU.

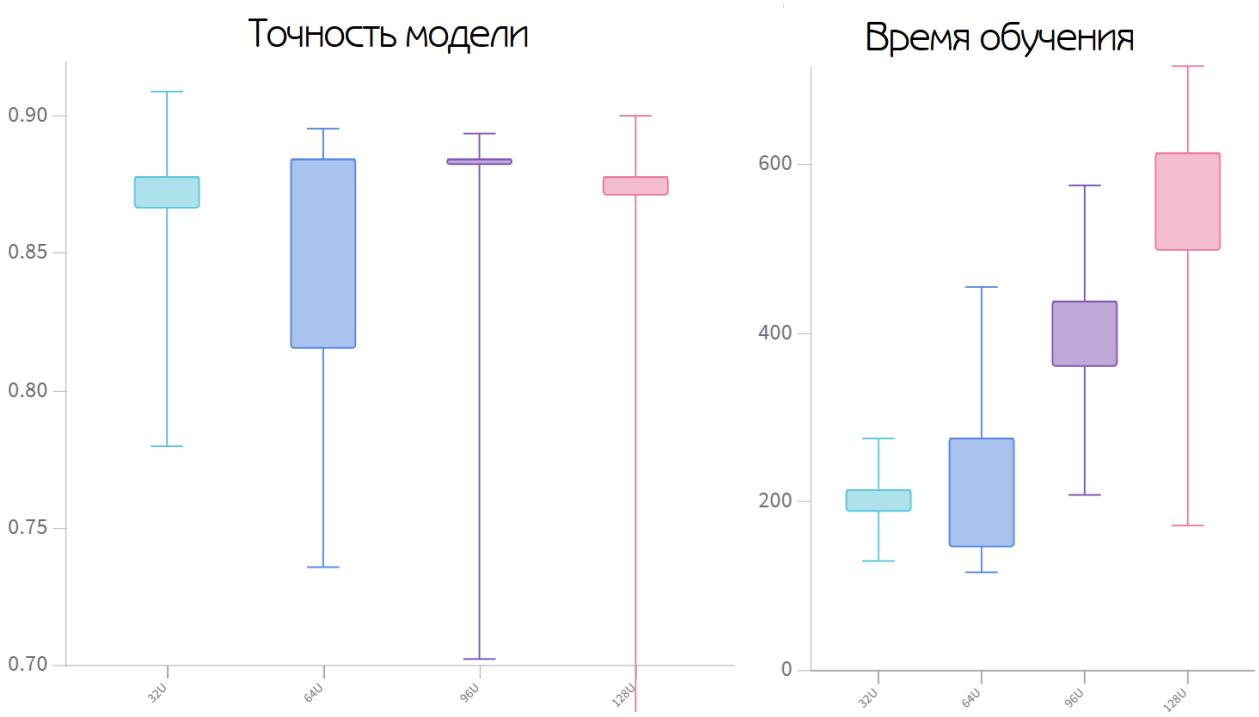

Рисунок 18 – Корреляция количества нейронов в рекуррентном слое с точностью и временем обучения

Из диаграммы точности можно заключить, что самые эффективные конфигурации имеют незначительные различия, разница между которыми не превышает 1%. Тем не менее, статистическая корреляция между ними все же присутствует.

С другой стороны, по диаграмме времени обучения можно явно наблюдать полиномиальный рост при увеличении числа нейронов рекуррентного слоя. Таким образом, 32 нейрона является самым оптимальным показателем для рекуррентного слоя.

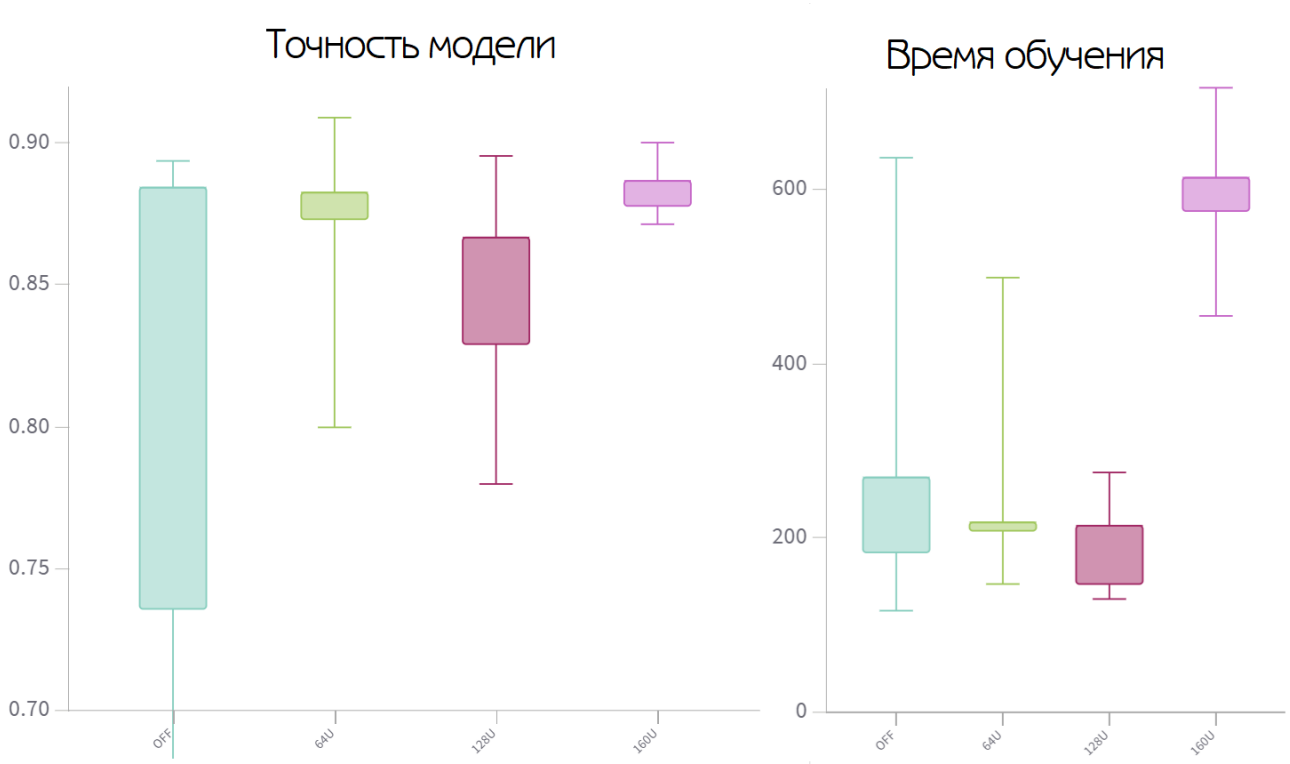

Рисунок 19 – Корреляция количества нейронов в слое Time Distributed с точностью и временем обучения

Согласно диаграммам, наличие или отсутствие слоя, а также его размер положительно коррелируют с точностью и не оказывают влияния на время обучения при малом количестве нейронов.

Таким образом, было принято решение использовать «Time Distributed»-слой с 64 нейронами.

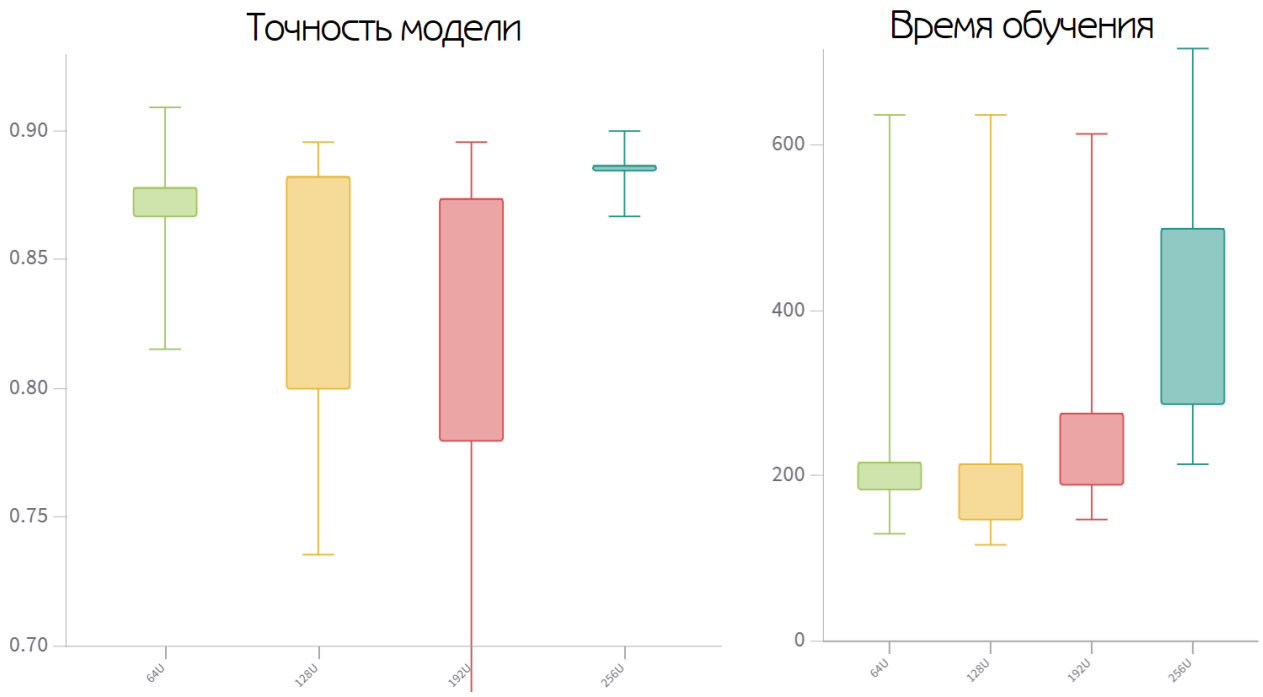

Рисунок 20 – Корреляция количества нейронов в линейном слое с точностью и временем обучения

На основе диаграммы можно сделать вывод, что время обучения модели зависит от количества нейронов экспоненциально, в то время как положительное влияние на точность модели присутствует, но является слабым.

Таким образом, решение использовать линейный слой с 64 нейронами после рекуррентного слоя будет оптимальным.

3.3.2.2 Обучение и анализ лучшей модели

Архитектура получившейся в результате перебора модели RNN представлена на рисунке 21.

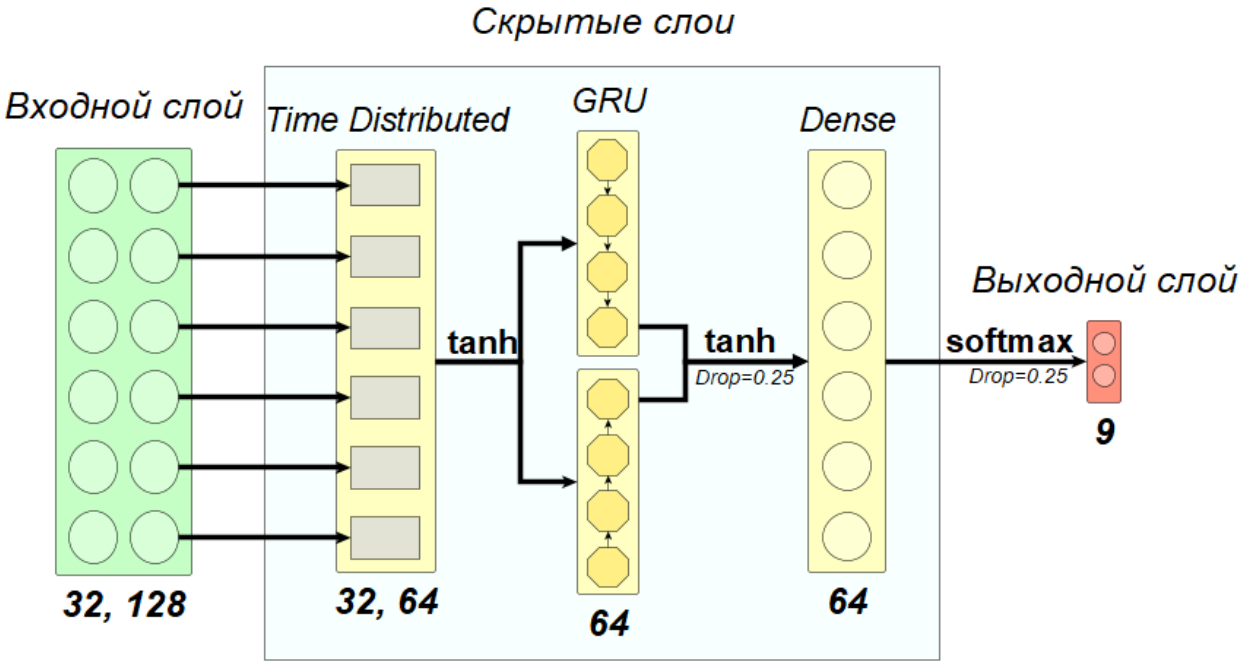

Рисунок 21 – Архитектура обученной рекуррентной нейронной сети

Модель включает в себя 6 последовательно расположенных слоев:

а) входной слой с 32 (количество слов) * 128 (размерность вектора слова) нейронами;

б) «Time Distributed»-слой с 32 * 64 нейронами и гиперболическим тангенсом в качестве функции активации;

в) двунаправленный GRU-слой с 32 (в сумме 64) нейронами, гиперболическим тангенсом в качестве функции активации и коэффициентом Dropout 0.25;

г) линейный слой с 64 нейронами, функцией активации Softmax и коэффициентом Dropout 0.25;

д) выходной слой с 9 нейронами.

Обучение проводилось на 50 эпохах с динамическим коэффициентом скорости обучения. В качестве функции потерь была выбрана категориальная кроссэнтропия. Функцией оптимизации является Adam с начальным коэффициентом скорости обучения, равным 0.02.

На рисунке 22 представлен график изменения коэффициента скорости обучения в процессе обучения, на рисунке 23 – изменения точности на обучающей и валидационной выборках.

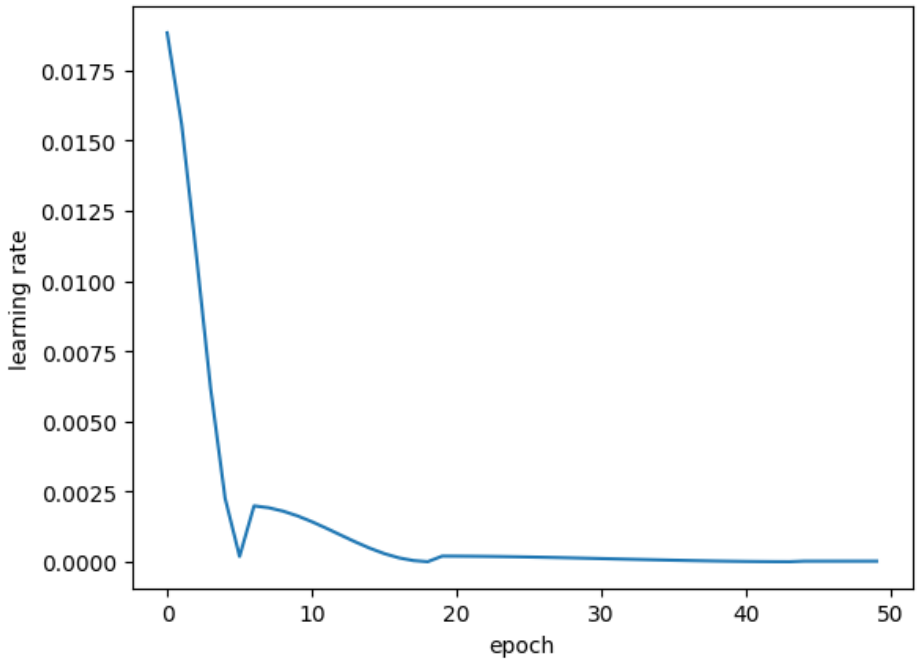

Рисунок 22 – Изменение learning rate при обучении рекуррентной нейронной сети

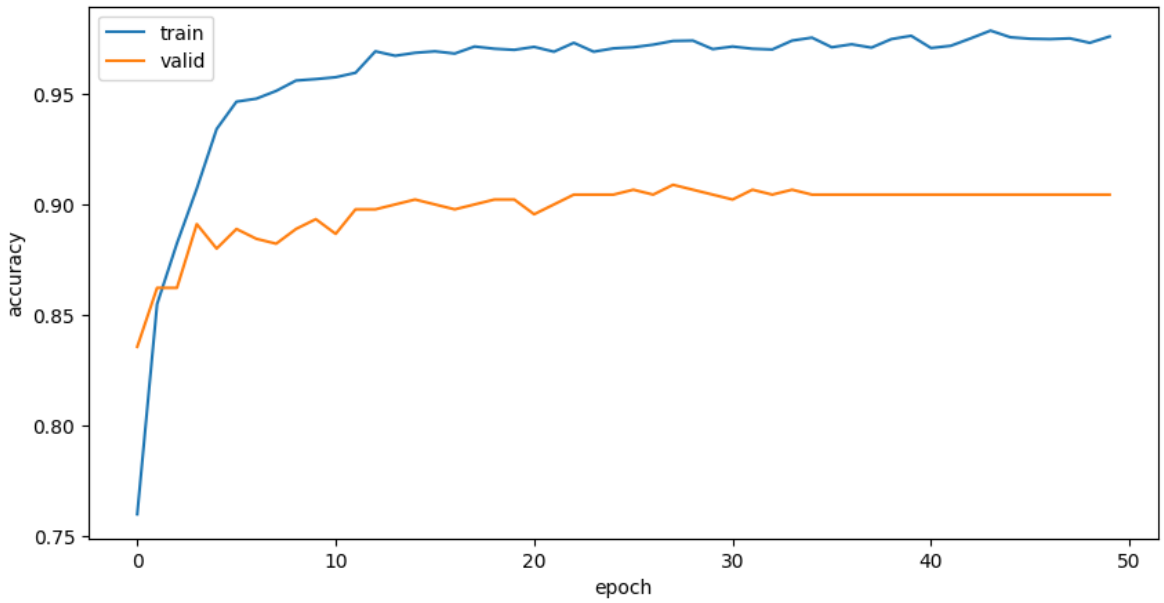

Рисунок 23 – Изменение точности обучающей и тестовой выборки при обучении рекуррентной нейронной сети

Как можно заметить из рисунка 23, модель, как и в случае с многослойным персептроном, подверглась переобучению.

Финальная точность модели на валидационной выборке – 91%, что на 3% выше точности модели многослойного персептрона. На рисунке 24 представлена матрица ошибок с рассчитанными precision (последняя строка), и recall (последний столбец).

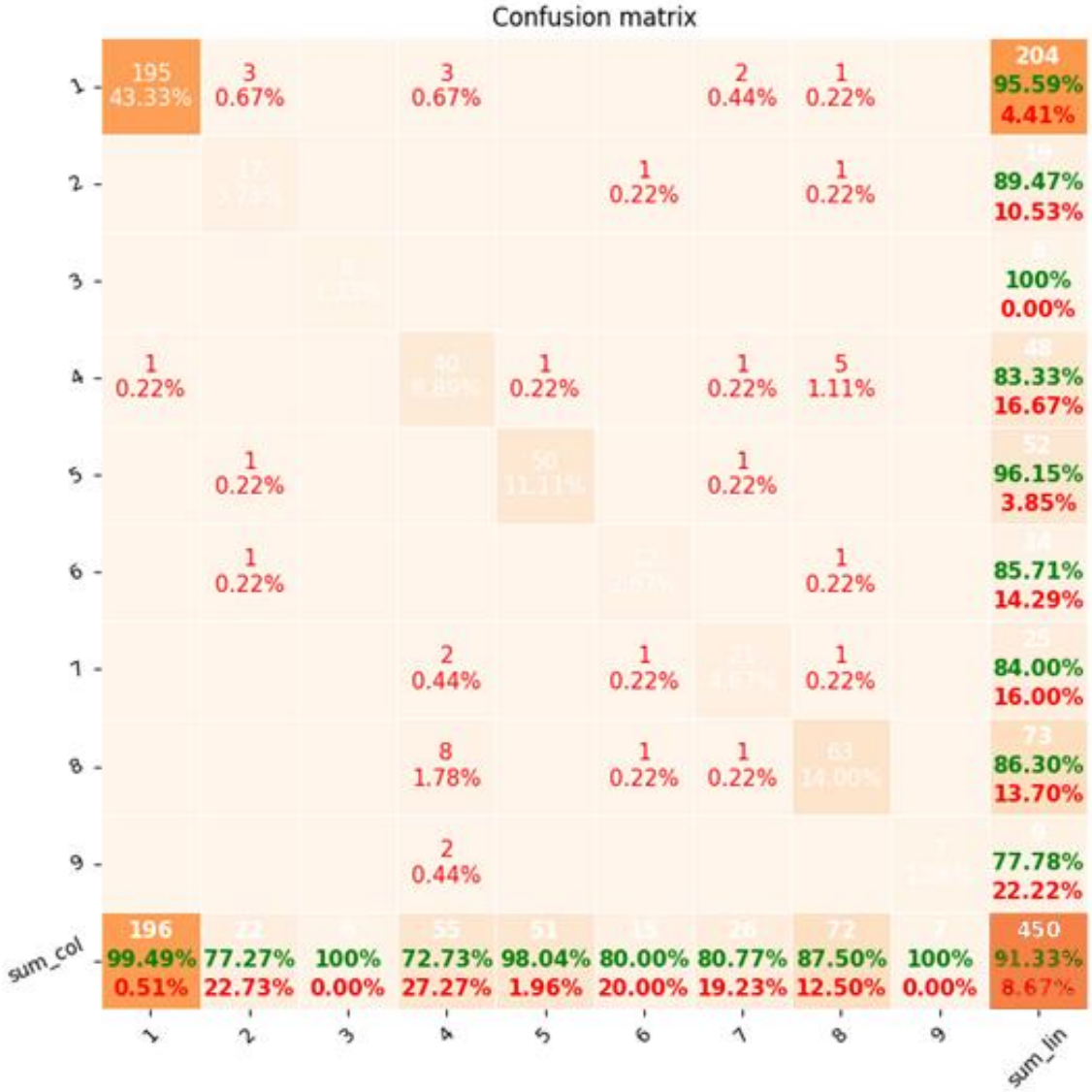

Рисунок 24 – Матрица ошибок классификации рекуррентной нейронной сети

Как видно из рисунка 24, модель классификации на основе рекуррентных сетей почти по всем показателям либо осталась на том же уровне, либо превзошла точность классификации многослойного персептрона. При этом эффект переобучения и дисбаланса классов стал выражен еще сильнее – оценка

precision для класса «Внесение изменений» (цифра 1 на рисунке) стала равной 99.5% на 195 замечаниях.

3.4 Кластеризация замечаний

3.4.1 Иерархическая кластеризация

В качестве тестовых данных использовалась специально составленная выборка с кластерами типовых замечаний, имеющая 100 записей.

Ниже указан список исследуемых параметров алгоритма иерархической кластеризации.

1. Affinity. Функция расчета расстояний между точками. Диапазон значений: '*euclidean*', '*l1*', '*l2*', '*cosine*', '*manhattan*'.

2. Distance threshold. Максимальный диаметр кластера. Диапазон значений: 0.6…3.

3. Linkage. Способ расчета сходства между кластерами. Диапазон значений: '*ward*', '*complete*', '*average*', '*single*'.

Результаты исследования алгоритма иерархической кластеризации с лучшими комбинациями параметров представлены в таблице 12.

Таблица 12 – Исследование влияния параметров алгоритма иерархической кластеризации на точность и время исполнения

affinity	distance_threshold	linkage	score	time	time relative
l1	0,75	average	0,891892	0	-0,86207
l1	0,75	single	0,891892	0	-0,86207
l1	1,05	complete	0,891892	0	-0,86207
l1	1,05	average	0,891892	0	-0,86207
manhattan	0,75	complete	0,891892	0	-0,86207
manhattan	1,05	complete	0,891892	0	-0,86207
manhattan	1,5	average	0,891892	0	-0,86207
manhattan	1,5	single	0,891892	0	-0,86207
manhattan	0,75	single	0,891892	0,000997	1,14925
manhattan	1,5	complete	0,891892	0,000998	1,152617
l1	1,5	average	0,891892	0,000999	1,153098
l1	1,5	complete	0,891892	0,000999	1,15406
l1	1,5	single	0,891892	0,001	1,155022

Окончание таблицы 12

manhattan	0,75	average	0,891892	0,001	1,155022
l1	0,75	complete	0,891892	0,001	1,155984

По результатам исследования можно заключить, что наилучшее время (почти мгновенное проведение кластеризации) слабо зависит от выбранных метрик, отклонения имеют в большей степени случайный характер. Оптимальный размер кластера – 0.75…1.5.

Максимальная точность алгоритма иерархической кластеризации – 89%, минимальное время пренебрежимо мало на выборке малого размера.

3.4.2 DBSCAN

Набор тестовых данных совпадает с тестовыми данными из предыдущего пункта.

Ниже указан список исследуемых параметров алгоритма DBSCAN.

1. Metric. Функция расчета расстояний между точками. Диапазон значений: '*euclidean*', '*l1*', '*l2*', '*minkowski*', '*manhattan*', '*cityblock*'.

2. Eps. Параметр ε алгоритма. Диапазон значений: 0.6…3.

3. Algorithm. Алгоритм поиска ближайшего соседа. Диапазон значений: '*ball_tree*', '*kd_tree*', '*brute*'.

Результаты исследования алгоритма DBSCAN с лучшими комбинациями параметров представлены в таблице 13.

Таблица 13 – Исследование влияния параметров алгоритма DBSCAN на точность и время исполнения

algorithm	**eps**	**metric**	**score**	**time**	**time relative**
brute	1,5	cityblock	0,891892	0,003001	-1,56929
brute	1,25	manhattan	0,891892	0,003001	-1,56923
brute	1,5	cityblock	0,891892	0,003001	-1,56923
brute	1,25	manhattan	0,891892	0,003002	-1,56897
brute	1,25	l1	0,891892	0,003004	-1,56839
brute	1,05	l1	0,891892	0,004	-1,29843
brute	1,25	cityblock	0,891892	0,004001	-1,29817
brute	1,5	manhattan	0,891892	0,004001	-1,29817

Окончание таблицы 13

brute	1,5	l1	0,891892	0,004002	-1,29791
kd_tree	1,5	l1	0,891892	0,006426	-0,64089
ball_tree	1,5	l1	0,891892	0,006535	-0,61129
kd_tree	1,15	cityblock	0,891892	0,006933	-0,50342
kd_tree	1,15	manhattan	0,891892	0,006974	-0,49218
kd_tree	1,05	cityblock	0,891892	0,006998	-0,48578

По результатам исследования можно заключить, что наилучшее время на малых данных наблюдается при значении algorithm, равным *brute*. Использование разных метрик почти не отражается на работе алгоритма. Оптимальное значение параметра ε лежит в диапазоне 1.05…1.5.

Максимальная точность алгоритма DBSCAN – 89%, минимальное время пренебрежимо мало на малой выборке.

3.4.3 OPTICS

Набор тестовых данных совпадает с тестовыми данными из предыдущего пункта.

Ниже указан список исследуемых параметров алгоритма DBSCAN.

4. Metric. Функция расчета расстояний между точками. Диапазон значений: '*euclidean*', '*l1*', '*l2*', '*minkowski*', '*manhattan*', '*cityblock*'.

5. Xi. Параметр ξ алгоритма. Диапазон значений: 0.01…0.3.

6. Algorithm. Алгоритм поиска ближайшего соседа. Диапазон значений: '*ball_tree*', '*kd_tree*', '*brute*'.

7. Predecessor correction. Корректировка кластера методами OPTICS. True или False.

Результаты исследования алгоритма OPTICS с лучшими комбинациями параметров представлены в таблице 14.

Таблица 14 – Исследование влияния параметров алгоритма OPTICS на точность и время исполнения

algorithm	metric	predecessor correction	xi	score	time	time_rel
brute	manhattan	-	0,11	0,911036	0,132001	-1,4798
brute	l1	-	0,11	0,911036	0,135	-1,44055
brute	l1	+	0,11	0,911036	0,137999	-1,4013
brute	manhattan	+	0,11	0,911036	0,139	-1,3882
brute	cityblock	+	0,11	0,911036	0,146007	-1,2965
brute	cityblock	+	0,11	0,911036	0,148004	-1,27037
ball_tree	manhattan	+	0,11	0,911036	0,220862	-0,31684
kd_tree	cityblock	+	0,11	0,911036	0,224078	-0,27475
kd_tree	l1	+	0,11	0,911036	0,225032	-0,26227
ball_tree	l1	-	0,11	0,911036	0,226248	-0,24636
ball_tree	l1	+	0,11	0,911036	0,227675	-0,22768
ball_tree	cityblock	-	0,11	0,911036	0,229164	-0,20819
ball_tree	cityblock	+	0,11	0,911036	0,233605	-0,15007
kd_tree	l1	-	0,11	0,911036	0,236747	-0,10895
ball_tree	manhattan	-	0,11	0,911036	0,244184	-0,01162

По результатам исследования можно заключить, что наилучшее время на малых данных наблюдается при значении algorithm, равным *brute*. Использование разных метрик и predecessor correction почти не отражается на работе алгоритма. Оптимальное значение параметра ξ лежит в диапазоне 0.05…0.15.

Максимальная точность алгоритма OPTICS – 91%, минимальное время при максимальной точности – 0.13 секунды на малой выборке.

3.4.4 Выбор лучшего алгоритма

Итак, было проанализировано три разных алгоритма кластеризации, примененных к эмбеддингам предложений. Результаты анализа:

а) иерархическая кластеризация – точность 89%, время работы – 7.2 секунды на 10000 замечаниях;

б) DBSCAN – точность 89%, время работы – 2.2 секунды на 10000 замечаниях;

в) OPTICS – точность 91%, время работы – 16 секунд на 10000 замечаниях.

Таким образом, алгоритм кластеризации DBSCAN является лидирующим для задачи быстрой кластеризации множества эмбеддингов предложений большого размера, несмотря на более высокую точность кластеризации алгоритма OPTICS.

Параметры алгоритма DBSCAN, показавшие наилучшие точность и время исполнения:

а) eps: 1.25;

б) metric: l1;

в) algorithm: brute.

3.5 Выводы по главе

По результатам анализа лучшей моделью составления эмбеддингов оказался FastText, выделяющийся наибольшей точностью классификации и допустимым временем обработки одного замечания.

Алгоритмом, проводящем кластеризацию на типовые замечания, был выбран DBSCAN, так как он выделяется самой быстрой скоростью работы относительно конкурентов, лишь незначительно проигрывая в точности кластеризации алгоритму OPTICS.

Исходные данные, предоставленные для задачи классификации, имеют проблему дисбаланса классов – самый крупный класс занимает 42% общей выборки, в то время как самый малый класс – всего 2%. Данная проблема в сочетании с малым размером выборки привели к переобучению модели классификатора на обучающих данных. Тем не менее, точность модели рекуррентной нейронной сети, показавшей наилучший результат, составила 91%.

4 Проектирование и разработка автоматизированного модуля

4.1 Общая информация о модуле

Модуль проводит анализ существующих и новых замечаний в системе, а также находит пары «Замечание – Критерий чек-листа» и записывает полученные результаты в базу данных компании Роснефть.

Модуль может быть запущен в автоматизированном режиме, активируя алгоритмы анализа замечаний по достижении временной отметки, заданной пользователем. Как только процесс обработки замечаний завершается, система вновь переходит в режим ожидания до наступления следующей временной метки.

В качестве основного языка, на котором написан программный модуль, был выбран Python 3.10 [9] по следующим причинам:

1) использование языка в сфере обучения и использования интеллектуальных систем машинного обучения;

2) относительно быстрая скорость разработки;

3) встроенная поддержка многопроцессорности (модуль multiprocessing).

Основные программные инструменты, использованные в системе:

а) Tensorflow-2.12.0 [10] – запуск нейросетевых моделей;

б) Scikit-learn-1.2.1 [11] – реализация моделей машинного обучения в контексте кластеризации;

в) SQLAlchemy-1.4.46 [12] – интеграция с БД Microsoft SQL Server;

г) FastText-0.9.2 [13] – реализация моделей NLP;

д) NLTK-3.8.1 [14] – предобработка текста;

е) Pandas-1.5.3 [15] – работа с таблицами данных.

ж) Pymorphy2-0.9.1 [16] – лемматизация слов.

4.2 Архитектура системы

Разрабатываемая программная система в общем случае представляет собой процесс-демон, запускающийся каждую ночь в отведенное время и проводящий

анализ необработанных системой замечаний, оставленных пользователями за прошедший день. Запуск планируется проводить в ночное время, чтобы не нагружать серверную аппаратуру Заказчика при дневной работе.

На рисунке 25 представлена диаграмма, иллюстрирующая схематичное представление структуры разработанной автоматизированной системы.

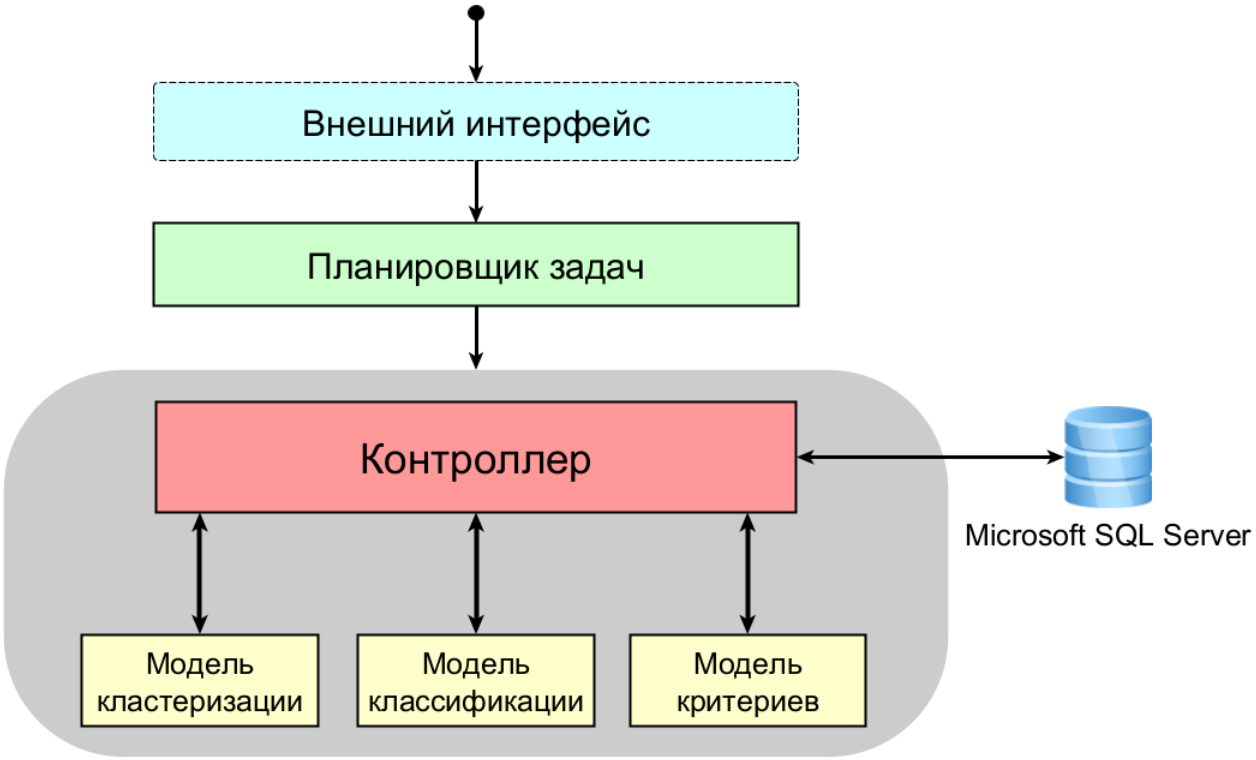

Рисунок 25 – Архитектура программной системы

Ниже представлено подробное объяснение каждой составной части диаграммы.

1. Внешний интерфейс. Часть системы, включающая в себя аргументы командной строки, конфигурирующие работу системы. Запускается в первую очередь до импорта основных библиотек.

2. Планировщик задач. Часть системы, ответственная за запуск «Контроллера», согласно расписанию, заданному пользователем при запуске программы. Является главным потоком.

3. Контроллер. Часть системы, запускающая себя в отдельном потоке. Отвечает за запуск и общение между собой элементов системы. Осуществляет

чтение и запись данных из БД посредством библиотеки SQLAlchemy и последовательный запуск моделей машинного обучения на полученных из БД и обработанных другими элементами системы данных.

4. Модель кластеризации. Часть системы, осуществляющая генерацию типовых замечаний из списка обычных замечаний. Реализует кластеризацию текста.

5. Модель классификации. Часть системы, осуществляющая распределение замечаний по глобальным классам. Реализует классификацию текста.

6. Модель критериев. Класс, осуществляющий поиск ближайших к замечаниям критериев чек-листа. Реализует анализ схожести текста.

Из диаграммы на рисунке 26 может быть неочевидна последовательность выполняемых программной системой операций. Чтобы решить эту проблему, была построена диаграмма последовательностей системы. Данная диаграмма отражена на рисунке 26.

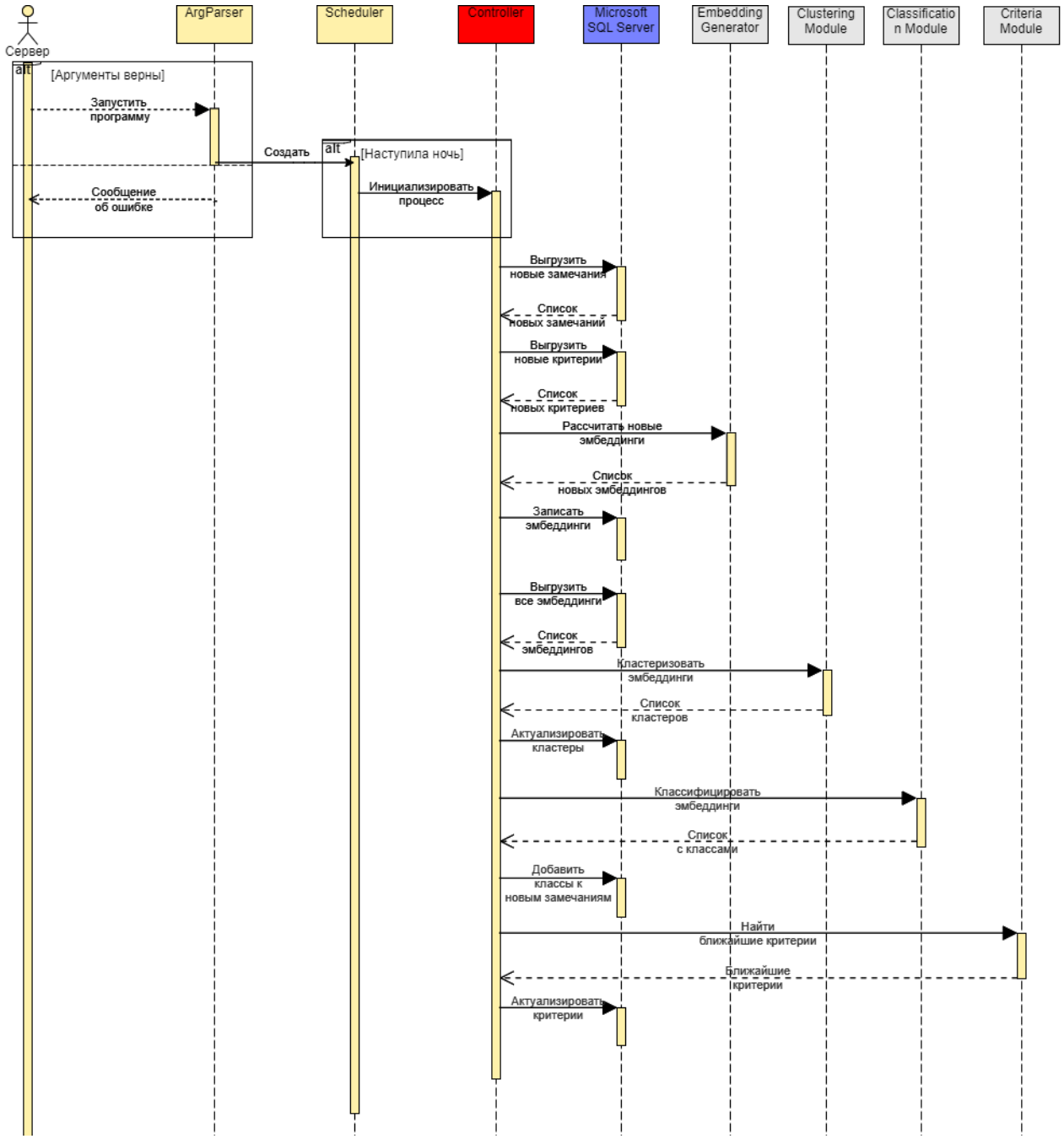

Рисунок 26 – Диаграмма последовательностей системы

4.3 Оболочка внешнего интерфейса

Внешний интерфейс по взаимодействию с ОС представляет собой консольное приложение, принимающее от пользователя набор параметров, конфигурирующих запуск и работу сервера.

Входные данные для запуска модуля задаются через аргументы командной строки. Список опциональны параметров можно разделить на три группы:

конфигурация запускаемых алгоритмов, конфигурация подключения к БД и тип подключения к БД.

Конфигурация запускаемых алгоритмов включает в себя параметр «**algorithm**» – перечень последовательно запускаемых алгоритмов (типизация замечаний, классификация замечаний, поиск ближайших критериев чек-листа). Для запуска программного модуля необходимо передать хотя бы один параметр из списка.

Конфигурация подключения к БД включает в себя параметры:

1. **-s** – IP или домен сервера базы данных;

2. **-d** – наименование базы данных.

При отсутствии хотя бы одного из этих параметров программная система не запустится.

Подключения к БД может осуществляется посредством передачи логина и пароля либо через систему Windows Authentication (в случае подключения к локальной базе данных Microsoft SQL Server на ОС Windows). В первом случае на вход принимаются следующие параметры:

1. **-u** – имя пользователя базы данных;

2. **-p** – пароль пользователя базы данных.

Во втором случае передается лишь один параметр «**--trusted**» – является ли подключение к БД доверенным. Для запуска программной системы необходимо выбрать один из этих способов.

Конфигурация способа запуска модуля включает в себя два параметра:

1. **-t** – время ежедневного запуска системы (параметр может принимать на вход несколько значений);

2. **-i** – независимый запуск.

На рисунке 27 представлен пример интерфейса командной строки во время работы программного модуля.

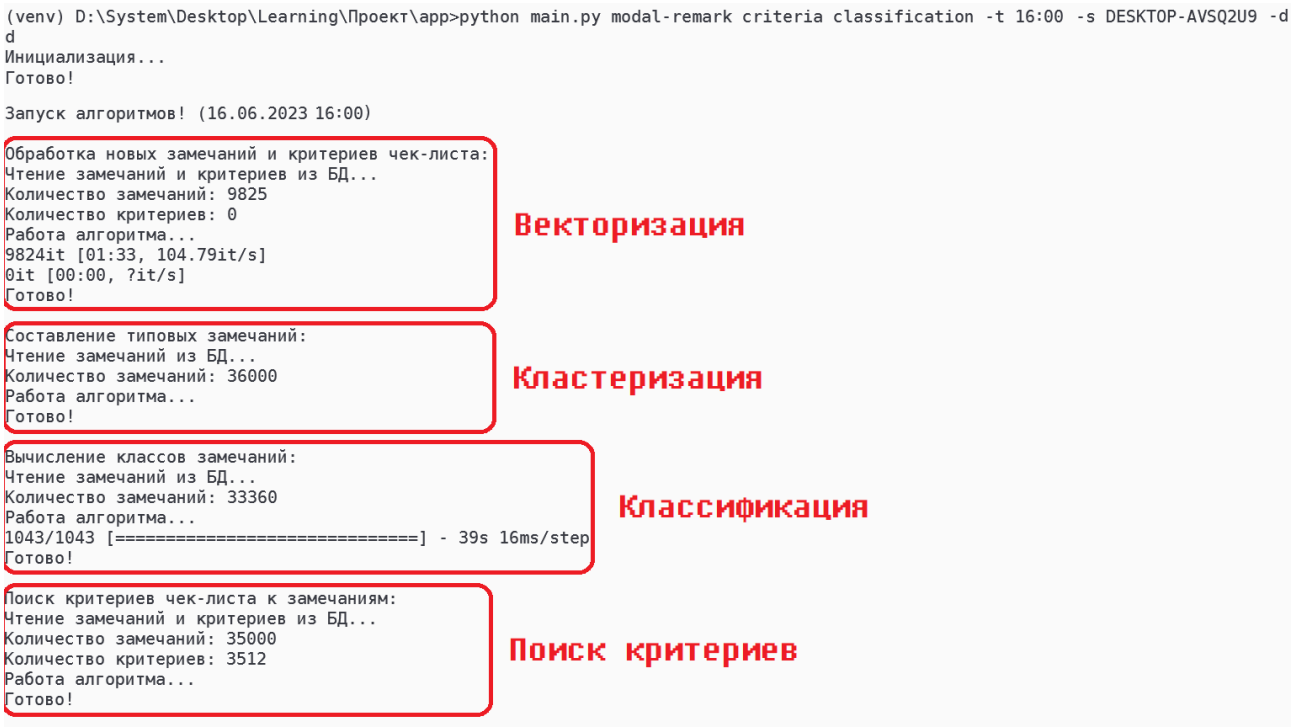

Рисунок 27 – Интерфейс командной строки

4.4 Коммуникация с БД

4.4.1 Подключение к базе данных

База данных на настоящий момент развернута и сконфигурирована на сервере ООО «РН КрасноярскНИПИнефть». Главная база данных для хранения пользовательской, системной и служебной информации – реляционная база данных Microsoft SQL Server.

Для подключения к реляционной базе данных Microsoft SQL Server было принято решение использовать программную библиотеку на языке Python с возможностью реализации ORM системы. Наиболее популярными библиотеками для этих целей являются SQLAlchemy и PeeWee. Несмотря на то, что PeeWee является лучшим решением для небольших систем, предпочтение было отдано SQLAlchemy-1.4.46 по причине большей популярности среди разработчиков, что позитивно скажется на читаемости и модифицируемости кода другими программистами.

Объект подключения к базе данных используется для простой конвертации таблиц из базы данных в объект pandas.DataFrame посредством метода pandas.read_sql_query().

Таким образом, запись в базу данных происходит методами SQLAlchemy, чтение из базы данных осуществляется методами pandas.

4.4.2 Сущности базы данных

На рисунке 27 отражена ER-диаграмма разработанной базы данных, расширяющей существующие таблицы Remark и CheckListCriteria.

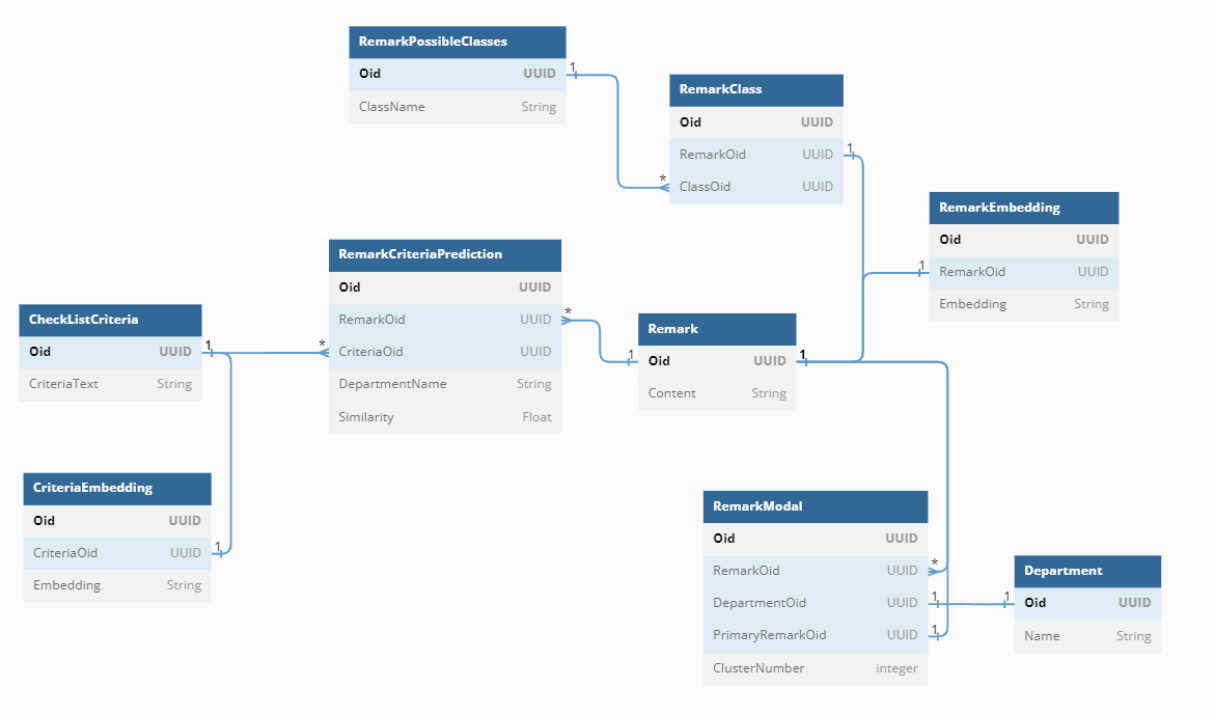

Рисунок 27 – ER-диаграмма базы данных

Ниже приведено подробное описание каждой сущности таблицы.

1. RemarkEmbedding – таблица, хранящая эмбеддинги замечаний.

а) Oid: UUID, Primary Key – уникальный идентификатор строки;

б) RemarkOid: UUID, Foreign Key – уникальный идентификатор замечания;

в) Embedding: String – массив чисел типа float, хранящихся в виде строки формата JSON, построенный согласно тексту соответствующего замечания.

2. CheckListCriteriaEmbedding – таблица, хранящая эмбеддинги критериев чек-листа.

а) Oid: UUID, Primary Key – уникальный идентификатор строки;

б) CriteriaOid: UUID, Foreign Key – уникальный идентификатор критерия чек-листа;

в) Embedding: String – массив чисел типа float, хранящихся в виде строки формата JSON, построенный согласно тексту соответствующего критерия чек-листа.

3. RemarkModal – таблица, хранящая информацию о кластерах типовых замечаний.

а) Oid: UUID, Primary Key – уникальный идентификатор строки;

б) RemarkOid: UUID, Foreign Key – идентификатор замечания;

в) DepartmentOid: UUID, Foreign Key – идентификатор департамента;

г) RemarkClusterNumber: Integer – номер кластера, к которому принадлежит типовое замечание;

д) PrimaryRemarkOid: String, Foreign Key – идентификатор замечания в кластере типовых замечаний, рассматриваемого в нем в качестве основного (центроид кластера).

4. RemarkPossibleClasses – таблица, хранящая информацию о типах возможных классов замечания.

а) Oid: Integer, Primary Key – уникальный идентификатор строки;

б) ClassName: String – наименование класса.

5. RemarkClass – таблица, хранящая информацию о возможных классах типовых замечаний.

а) Oid: UUID, Primary Key – уникальный идентификатор строки;

б) RemarkOid: UUID, Foreign Key – уникальный идентификатор замечания;

в) ClassOid: UUID, Foreign Key – идентификатор класса замечания.

6. RemarkCriteriaPrediction – таблица, хранящая не больше пяти самых близких критериев чек-листа для замечания.

а) Oid: UUID, Primary Key – уникальный идентификатор строки;

б) RemarkOid: UUID, Foreign Key – идентификатор замечания;

в) CriteriaOid: UUID, Foreign Key – идентификатор критерия чек-листа;

г) DepartmentName: String – название департамента, по которому составлена пара критерия и замечания;

д) Similarity: Float – степень близости критерия чек-листа и замечания, чем она выше, тем вероятнее связь между ними;

4.5 Развертывание программного модуля

Разработанный программный модуль планируется развернуть на сервере компании ООО «РН-КрасноярскНИПИнефть».

Предполагается два возможных способа развертывания проекта на устройствах с ОС Windows и Linux.

Первый способ заключается в установке интерпретатора Python 3.10, менеджера пакетов pip и самих пакетов из файла requirements.txt вручную. Такой подход удобен, если на устройстве уже установлен интерпретатор Python необходимой версии.

Вторым способом является создание и монтирование докер-контейнера. Содержимое докер-контейнера представлено в листинге 1.

Листинг 1 – Содержимое файла Dockerfile

```
FROM ubuntu:20.04
ARG DEBIAN_FRONTEND=noninteractive

RUN apt-get -y update && apt-get -y install software-properties-common curl \
&& add-apt-repository ppa:deadsnakes/ppa && apt-get -y install python3.10 python3-pip python3.10-distutils python3-apt

RUN curl -sS https://bootstrap.pypa.io/get-pip.py | python3.10
```

Окончание листинга 1

```
RUN python3.10 -m pip install --upgrade pip

RUN curl https://packages.microsoft.com/keys/microsoft.asc | apt-key add - \
&&    curl    https://packages.microsoft.com/config/ubuntu/20.04/prod.list    >
/etc/apt/sources.list.d/mssql-release.list \
&& apt-get update --allow-unauthenticated \
&& apt-get install -y unixodbc \
&& ACCEPT_EULA=Y apt-get install -y msodbcsql17 \
&& apt-get clean -y \
&& export PATH="$PATH:/opt/mssql-tools/bin"

WORKDIR /app
COPY requirements.txt ./

RUN python3.10 -m pip install -r requirements.txt

COPY . /app

ENTRYPOINT ["python3.10", "main.py"]
```

4.6 Выводы по главе

В рамках главы было представлено описание разработанного программного модуля. Оно представляет собой консольное приложение, конфигурирующее свою работу с применением аргументов командной строки. В качестве языка разработки был выбран язык Python.

По наступлении указанного пользователем времени приложение в отдельном потоке запускает алгоритмы и модели анализа замечаний, записывая результаты своей работы в таблицы базы данных SQL.

К программному модулю была составлена инструкция по установке и эксплуатации и создан удаленный репозиторий.

ЗАКЛЮЧЕНИЕ

По итогам выпускной квалификационной работы был разработан автоматизированный программный модуль, проводящий интеллектуальный анализ замечаний. Процесс анализа состоит из двух основных шагов: поиск типовых замечаний и расчет классов замечаний.

Поиск типовых замечаний – процесс нахождения дубликатов в базе данных замечаний, имеющих отличное синтаксическое, но схожее семантическое значение. Обнаружение дубликатов производится посредством кластеризации массива эмбеддингов замечаний алгоритмом DBSCAN. Точность кластеризации составила 89%.

Расчет классов замечаний – процесс нахождения классов замечаний. Предсказание классов осуществляется с использованием рекуррентной нейронной сети со слоем GRU. Точность классификации составила 91%.

Разработанный программный модуль может быть запущен в автоматическом режиме, активируя алгоритмы анализа замечаний согласно заданному пользователем времени.

Финальная версия программного продукта была передана заказчику вместе с инструкцией по запуску и эксплуатации модуля.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Обработка естественного языка / Ю.А. Золушкин, Т.А. Васяева // Материалы научно-практической конференции для молодых ученых «Young scientists' researches and achievements in science» посвященной 100-летнему юбилею Донецкого национального технического университета. – Донецк: ДонНТУ – 2021. – С. 71-79.

2. Анализ алгоритмов классификации текстов / Т.В. Логунова, Л.В. Щербакова, В.М. Васюков, В.В. Шимкун // Universum: технические науки. – 2023. – № 2 (107). – С. 4-20.

3. ML: Трансформер // Машинное обучение и нейронные сети : [сайт]. – URL: https://qudata.com/ml/ru/NN_Attention_Transformer.html (дата обращения 10.06.2023)

4. Гафаров Ф.М. Искусственные нейронные сети и приложения: учеб. пособие / Ф.М. Гафаров, А.Ф. Галимянов. – Казань: Изд-во Казан. ун-та, 2018. – С. 75-79.

5. Обзор основных методов классификации и кластеризации данных / Д.С. Черезов, Н.А. Тюкачев // Вестник ВГУ, серия: системный анализ и информационные технологии. – 2009. – № 2. – С. 25-29.

6. Современные тенденции методов интеллектуального анализа данных: метод кластеризации / Н. Махрусе // Московский экономический журнал. – 2019. – №6. – С. 359-377.

7. Кластеризация: Учебник по машинному обучению : [сайт]. – URL: https://academy.yandex.ru/handbook/ml/article/klasterizaciya (дата обращения 11.06.2023)

8. Deep Learning vs common sense: разрабатываем чат-бота // Новости информационных технологий : [сайт]. – URL: https://www.pvsm.ru/algoritmy/327761 (дата обращения 10.06.2023)

9. Python Programming Language : [сайт]. – 2023. – URL: https://www.python.org/ (дата обращения 20.05.2023).

10. TensorFlow documentation : [сайт]. – 2023. – URL: https://www.tensorflow.org/api_docs (дата обращения 20.05.2023).

11. Scikit-learn documentation : [сайт]. – 2023. – URL: https://scikit-learn.org/stable/user_guide.html (дата обращения 20.05.2023).

12. SQLAlchemy documentation : [сайт]. – 2023. – URL: https://docs.sqlalchemy.org/en/20/ (дата обращения 20.05.2023).

13. FastText documentation : [сайт]. – 2023. – URL: https://fasttext.cc/docs/en/python-module.html (дата обращения 20.05.2023).

14. Natural Language ToolKit documentation : [сайт]. – 2023. – URL: https://nltk.readthedocs.io/en/stable/ (дата обращения 20.05.2023).

15. Pandas documentation : [сайт]. – 2023. – URL: https://pandas.pydata.org/docs/ (дата обращения 20.05.2023).

16. Pymorphy2 documentation : [сайт]. – 2023. – URL: https://pymorphy2.readthedocs.io/en/stable/user/guide.html (дата обращения 20.05.2023).