МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение
высшего образования
«ТЮМЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ МАТЕМАТИКИ И КОМПЬЮТЕРНЫХ НАУК
Кафедра программного обеспечения

РЕКОМЕНДОВАНО К ЗАЩИТЕ В ГЭК
Заведующий кафедрой
К.т.н., доцент
_____ М. С. Воробьева
25.06 2022 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА
магистерская диссертация

РАЗРАБОТКА СЕРВИСА АВТОМАТИЧЕСКОЙ ОБРАБОТКИ ОБРАЩЕНИЙ НА ГОРЯЧУЮ ЛИНИЮ МКУ “ТЮМЕНЬГОРТРАНС”

02.04.03 Математическое обеспечение и администрирование информационных систем
Магистерская программа «Разработка технологий Интернета вещей и больших данных»

Выполнил работу
студент 2 курса
очной формы обучения
Леонов Никита Андреевич

Научный руководитель
Доцент, к. пед. н.
Плотоненко Юрий Анатольевич

Рецензент
Tech Lead группы Data Science,
Отдела разработки и исследований
ООО «Нейро»,
ведущий аналитик данных
Глазков Максим Юрьевич

Тюмень
2022

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

На сегодняшний день во многих компаниям существуют собственные горячие линии или call-центры для работы с жалобами, вопросами и обращениям вне зависимости от сферы их деятельности. При этом, все больше и больше операций по работе с вопросами звонящего автоматизируется за счет использования голосовых или чат ботов, что способствует увеличению скорости обработки обращений и снижению затрат на обучение и содержание персонала для их обработки.

В МКУ «Тюменьгортранс» уже много лет существует собственная горячая линия для работы с обращениями граждан города Тюмени по вопросам общественного транспорта, маршрутной сети и парковочного пространства. При этом зачастую горячая линия сильно загружена и звонящему приходится ждать большое количество времени в ожидании ответа освободившегося оператора. Длительное ожидание создает негативное впечатление о компании, а большое количество подряд идущих звонков создает дополнительную нагрузку на операторов горячей линии, что снижает их работоспособность.

В связи с перечисленными проблемами возникает идея использовать чат бот для перенаправления части обращений на него и последующей их автоматической обработки, то есть определение категории обращения и занесение его в систему “АСД”.

В настоящее время в области обработки естественного языка решается большое количество задач связанных с классификацией текстов и распознаванием речи. Основными подходами к решению данных задач являются: модели машинного обучения, статистические методы и нейронные сети. В результате были разработаны готовые модели, общие подходы и алгоритмы для их решения.

На текущий момент МКУ «Тюменьгортранс» обладает всеми необходимыми данными для разработки собственного чат бота, который

позволит производить обработку обращений от граждан города Тюмени в автоматическом режиме.

Цель выпускной квалификационной работы – разработать систему для автоматической обработки и классификации по категориям поступающих обращений на горячую линию МКУ «Тюменьгортранс».

Для осуществления обозначенной цели были поставлены следующие задачи:

1. сформировать исходные тексты обращений из имеющихся записей телефонных разговоров и сделать их привязку к готовым заявкам, хранящимся в системе “АСД”;
2. разработать классификатор для соотнесения обращений к существующим категориям;
3. создать Telegram бот для получения необходимых данных от пользователя (время, место, контактные данные обратившегося, маршрут на котором что-то произошло и суть обращения) и занесения обработанной заявки в систему “АСД”;
4. проанализировать полученные результаты.

ГЛАВА 1. КЛАССИФИКАЦИЯ ТЕКСТОВ

1.1 ПОСТАНОВКА ЗАДАЧИ

Горячая линия МКУ «Тюменьгортранс» предназначена для комплексного информационно-аналитического обеспечения процессов МКУ «Тюменьгортранс» в части исполнения следующих процессов:

- Принятие обращений от граждан города Тюмени по работе общественного транспорта и замечаниям к функционированию маршрутной сети.
- Обработка обращения и присвоение ему необходимой категории.
- Занесение обработанных данных в информационную систему “АСД”

Основными целями создания системы для автоматической обработки обращений горячей линии являются:

- Создание информационной системы, которая бы предоставляла возможность комплексного обеспечения процессов обработки обращений. Существующая модель обработки с использованием телефонной линии является морально устаревшей, не обеспечивает должной скорости обработки обращений и зачастую дает сбои при большом скоплении звонящих.
- Повышение эффективности исполнения вышеописанных процессов, путем сокращения непроизводительных и дублирующих операций, операций, выполняемых «вручную».
- Уменьшение среднего времени разговора через телефонную линию минимум на 1 минуту за счет перенаправления объемных заявок на Telegram бот.

Для реализации поставленных целей система должна решать следующие задачи:

- Заполнение данных обращения от пользователя.
- Автоматическое соотнесение обращения к нужной категории.
- Интегрироваться с существующей информационной системой “АСД”.

Формально задача классификации может быть записана следующим образом: существует множество документов D = {d1, …, d|D|} и множество возможных категорий (классов) C = {c1, …, c|C|}. Неизвестная целевая функция Ф: D × C → {0, 1} задается формулой

$$\Phi(d_j, c_i) = \begin{cases} 0, \text{если } d_j \notin c_i \\ 1, \text{если } d_j \in c_i \end{cases} \quad (1)$$

Необходимо построить классификатор Ф', максимально близкий к Ф.

Следует отметить, что в приведенной выше постановке задачи о категориях и документах есть только информация, которая может быть извлечена из самого документа [1].

Если классификатор выдает точный ответ:

Ф': D × C → {0, 1}, (2)

тогда классификация называется точной.

Если классификатор определяет степень сходства (CSV) документа:

CSV: D → [0, 1], (3)

тогда классификация называется пороговой.

Процесс обучения с учителем можно описать следующим образом. Имеется обучающая выборка L, с которой система еще не сталкивалась и не обладает шаблоном для определения исходных классов. На основе этой выборки происходит обучение классификатора, то есть определение значений параметров, для которых модель показывает наилучший результат. Данный процесс выполняется путем формирования решающих правил, с помощью которых выборка разбивается на заданные категории. Для проверки качества полученного разбиения используется тестовая выборка T. При этом необходимо удовлетворение условий [2]:

$L \cap T = \emptyset$, (4)

$\Omega = L \cup T \subset C \times D$. (5)

Для множества примеров Ω известны значения целевой функции Ф.

Решение задачи классификации состоит из четырех последовательных этапов:

1. предварительная обработка и индексация документов;
2. уменьшение размерности пространства признаков;
3. создание и обучение классификатора с использованием методов машинного обучения;
4. оценка качества классификации.

1.2 ПРЕДОБРАБОТКА ТЕКСТА

Целью предварительной обработки текста является удаление нерелевантных фрагментов текста, поскольку они могут скрывать значимые шаблоны и приводить к снижению качества классификации. Также в процессе предобработке текста производится токенизация и приведение слова к основной форме (стемминг или лемматизация).

Во время предварительной обработки сначала производится очистка от слов, которые, как правило, содержат мало информации, такие как артикли союзы и предлоги. Эти слова называются стоп-словами и для каждого языка этот список является уникальным. Когда регистр букв не имеет значения, рекомендуется преобразовать все заглавные буквы в строчные.

Далее производится токенизация отдельных слов. Токенизация также может производиться не только для слов, но и для знаков препинания, цифр, тегов и других символов (например, смайликов). Знаки препинания и цифры, если они не относятся к рассматриваемой задаче классификации, удаляются, хотя в некоторых случаях они могут быть информативными и, таким образом, сохраняться (восклицательные знаки или смайлики, например, могут указывать на настроение).

Затем производится нормализация текста, то есть приведение к основной форме слова. Обычно этот процесс выполняется путем стемминга (удаление суффиксов слова) или лемматизации (приведение слова к его канонической форме – лемме). Ключевым допущением при нормализации слов является то, что слова, имеющие сходные корни, идентичны по значению. Этот процесс

позволяет учитывать только значимые слова в тексте и значительно снижает размерность итогового пространства признаков.

Процесс создание числовой модели текста, с которой в дальнейшем будут работать алгоритмы классификации, называется индексацией документов.

Одной из самых распространенных моделей индексации текста является модель «мешка слов» (bag-of-words). Ключевым допущением в этом подходе является то, что текстовое предложение может быть выражено с использованием неупорядоченного набора частот выбранных слов. Другими словами, документ представляет собой вектор в многомерном пространстве, координаты которого соответствуют номерам слов, а значения координат – значениям весов [3].

Другая распространенная модель индексации – Word2vec. Она основана на представлении текста с использованием так называемых вложений слов, часто называемых word-to-vec. Данная модель представляет каждое слово в виде вектора, содержащего информацию о контекстных (сопутствующих) словах [4].

Аналогично модели bag-of-words, Word2vec игнорирует (по умолчанию) порядок слов. Можно принять во внимание n-граммы слов, которые формируются из последовательностей соседних символов и тем самым будет учтен локальный порядок слов.

1.3 УМЕНЬШЕНИЕ РАЗМЕРНОСТИ ПРОСТРАНСТВА ПРИЗНАКОВ

Так как слишком большая размерность исходного пространства признаков приводит к большой вычислительной сложности алгоритмов классификации, эту самую размерность стараются понизить за счет уменьшения количества признаков (терминов).

Одним из самых распространенных способов определения веса признаков документа является вычисление функции TF-IDF [5]. Его основная идея заключается в том, что слова с высокой частотой использования в определенном документе и с низкой частотой использования в других документах получали больший вес.

Частота термина TF (term frequency) – оценка важности слова в пределах одного документа d вычисляется по формуле

$$TF = n_{t,d} / n_d, \quad (6)$$

где $n_{t,d}$– количество употреблений слова t в документе d; n_d – общее количество слов в документе d.

Обратная частота документа IDF (inverse document frequency) – инверсия частоты, с которой слово встречается в документах коллекции. IDF уменьшает вес часто используемых слов по формуле

$$IDF = \log(|D| / D_t), \quad (7)$$

где $|D|$ – количество всех документов в выборке; D_t – количество всех документов, в которых встречается слово t.

Окончательный вес термина в документе относительно всего набора документов вычисляется по формуле

$$V_{t,d} = TF \times IDF. \quad (8)$$

Следует отметить, что формула (8) оценивает значимость термина только с точки зрения частоты его появления в документе, не принимая во внимание порядок терминов в документе и их лексическую сочетаемость.

Построение и обучение классификатора с использованием методов машинного обучения можно разделить на следующие категории:

- вероятностные;
- метрические;
- логические;
- линейные;
- методы на основе искусственных нейронных сетей.

1.4 МЕТОД ОПОРНЫХ ВЕКТОРОВ

Метод опорных векторов (Support Vector Machine, SVM) – это метод линейной классификации. Данный метод широко применяется в таких разнообразных областях, как классификация текста, распознавание изображений и распознавание рукописных цифр. Популярность метода SVM обусловлена

следующими факторами. Во-первых, метод имеет прочную теоретическую основу. Также алгоритм SVM может быть применен в различных областях за счет своей гибкости и стабильности. Гибкость в данном алгоритме достигается за счет параметризации с помощью различных функций ядра. В свою очередь алгоритм имеет хорошую масштабируемость, благодаря чему способен работать с большими наборами данных. И наконец, одной из самых важных причин популярности SVM является его точность.

Для понимания работы метода SVM перейдем к пространству размерности |D|, точками которого являются исходные документы для классификации. Группа точек называется линейно разделимой в том случае, когда точки разных классов могут быть разделены гиперплоскостью (прямой линией в случае двухмерного пространства). И тогда решение достигается путем построения прямой таким образом, чтобы точки разных классов находились по разные стороны от этой прямой. После чего для классификации новых точек достаточно посмотреть с какой стороны прямой они будут расположены.

Исходя из вышеописанного условия, можно провести бесконечное количество прямых (гиперплоскостей), но для более точного результата классификации необходимо построить прямую так, чтобы она располагалась как можно дальше от всех известных точек. В данном случае расстояние от множества точек до прямой рассчитывается, как расстояние между прямой и ближайшей к ней точкой из множества. Ниже на рисунке 1 представлена разделяющая гиперплоскость L. Гиперплоскость называется разделяющей, если она максимизирует расстояние до двух параллельных гиперплоскостей. На рисунке 1 пунктирные линии проходят через опорные вектора (точки наиболее близкие к параллельным гиперплоскостям). Так как максимизация расстояния между классами позволяет добиться более уверенной классификации, то алгоритм старается увеличить расстояние между этими параллельными гиперплоскостями и за счет этого уменьшается средняя ошибка классификатора [2].

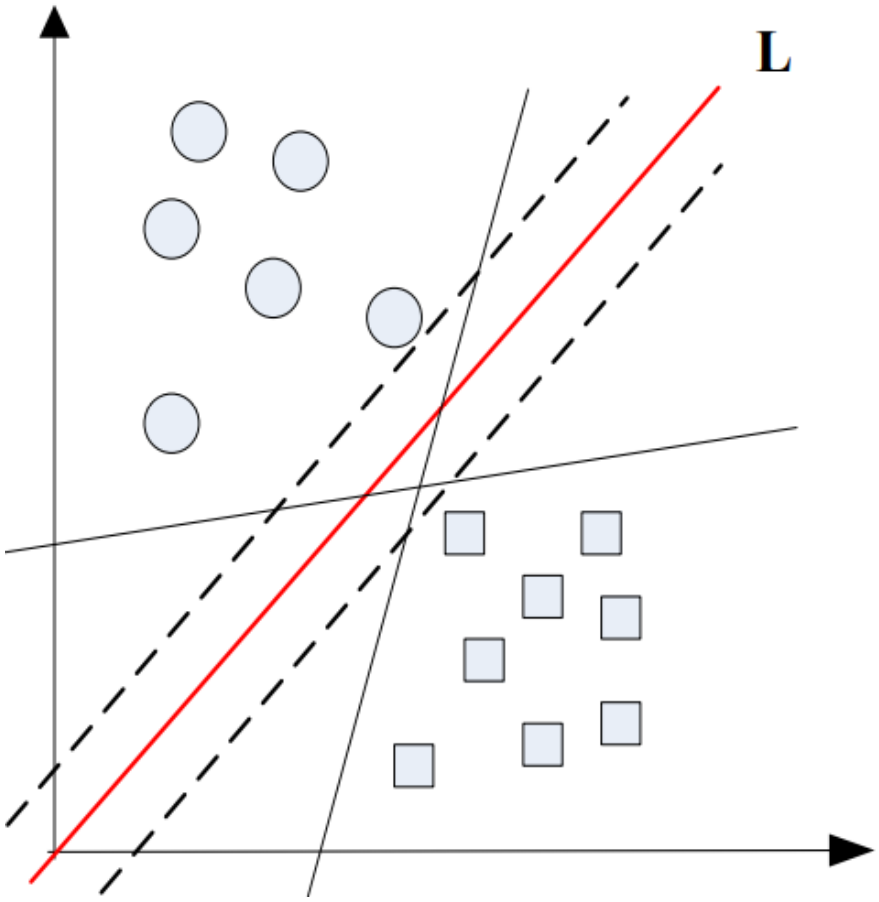

Рис. 1. Разделяющая гиперплоскость в методе опорных векторов

Но не всегда удается разделить гиперплоскостью точки одного класса и в таком случае выборка называется линейно неразделимой. Для решения этой проблемы требуется осуществить переход к новому пространству признаков за счет замены каждого скалярного произведения определенной функцией. Данная функция должна отвечать некоторым требованиям. Одним из таких требований, например, может быть наложение штрафа за каждый неверно определенный класс документа. Такую функцию еще называют ядром.

Преимущества метода:

- один из наиболее качественных методов;
- возможность работы с небольшим набором данных для обучения;
- сводимость к задаче выпуклой оптимизации, имеющей единственное решение.

Недостатки метода:

- сложная интерпретируемость параметров алгоритма;

- неустойчивость по отношению к выбросам в исходных данных.

1.5 МЕТОДЫ НА ОСНОВЕ ИСКУССТВЕННЫХ НЕЙРОННЫХ СЕТЕЙ

Существует большое количество видов нейронных сетей, основными из которых являются сети прямого распространения, рекуррентные сети, радиально-базисные функции и самоорганизующиеся карты.

В классических нейронных сетях прямого распространения сигнал последовательно проходит от входного слоя нейронов через промежуточные слои к выходному. Примером такой структуры является многослойный перцептрон.

Для классификации с помощью нейронной сети прямого распространения, веса признаков документа подаются на соответствующие входы сети. Активация распространяется по сети, а значения, получившиеся на выходах, и являются результатом классификации. Обычно обучение подобной сети происходит методом обратного распространения ошибки, суть которого заключается в следующем: если на одном из выходов получен неверный ответ для обучающего документа, то происходит распространение ошибки в обратном направлении сети и веса ребер изменяется с целью уменьшения ошибки.

Сверточная нейронная сеть (CNN) – это архитектура глубокого обучения, которая обычно используется для иерархической классификации документов. Хотя изначально CNN была создана для обработки изображений, она также эффективно используется для классификации текста. В базовом CNN для обработки изображений тензор изображения производит операцию свертки с помощью набора ядер размером d × d. Для того чтобы уменьшить вычислительную сложность, CNN использует операция подвыборки (pooling) для уменьшения размера выходных данных от одного уровня к следующему в сети.

Рекуррентная нейронная сеть (RNN) является естественным обобщением нейронных сетей прямого распространения для последовательностей данных. Она получается путем введения обратных связей из многослойного перцептрона.

Одной из широко распространенных разновидностей рекуррентных нейронных сетей является сеть Элмана – представлена на рисунке 2 [6]. В ней обратные связи идут не от выхода сети, а от выходов внутренних нейронов, что позволяет учесть предысторию наблюдаемых процессов.

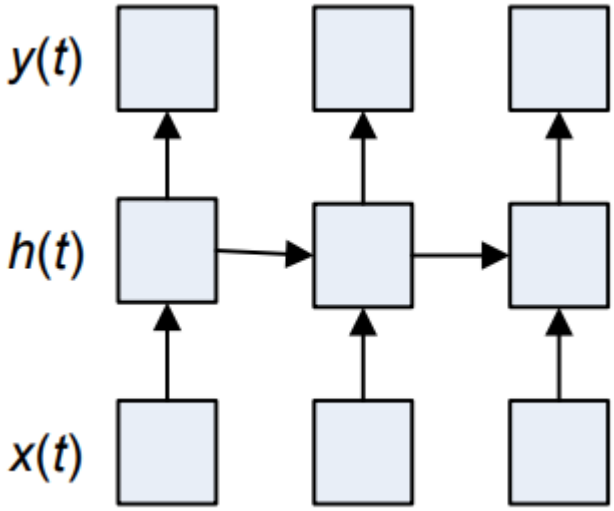

Рис. 2. Нейронная сеть Элмана (разновидность рекуррентной сети)

Рекуррентные нейронные сети добавляют память в искусственные нейронные сети, но данная память получается короткой. На каждом шаге обучения информация в памяти смешивается с новой и через несколько итераций полностью перезаписывается. Эта проблема называется проблемой долговременных зависимостей или проблемой исчезновения градиента. Сеть LSTM – это особый тип RNN, которая решает эти проблемы путем сохранения долгосрочной зависимости более эффективным способом по сравнению с базовой RNN. LSTM особенно эффективна в плане решения проблемы исчезающего градиента [7].

Модуль LSTM-сети представляет собой не один слой как в обычной модели рекуррентной сети, а уже четыре слоя, которые взаимодействуют особым образом (рисунок 3). Желтым прямоугольником на рисунке обозначен слой нейронной сети, розовым кружком – поточечная операция, стрелками – поток передачи целого вектора, сходящимися стрелками – конкатенация векторов [8].

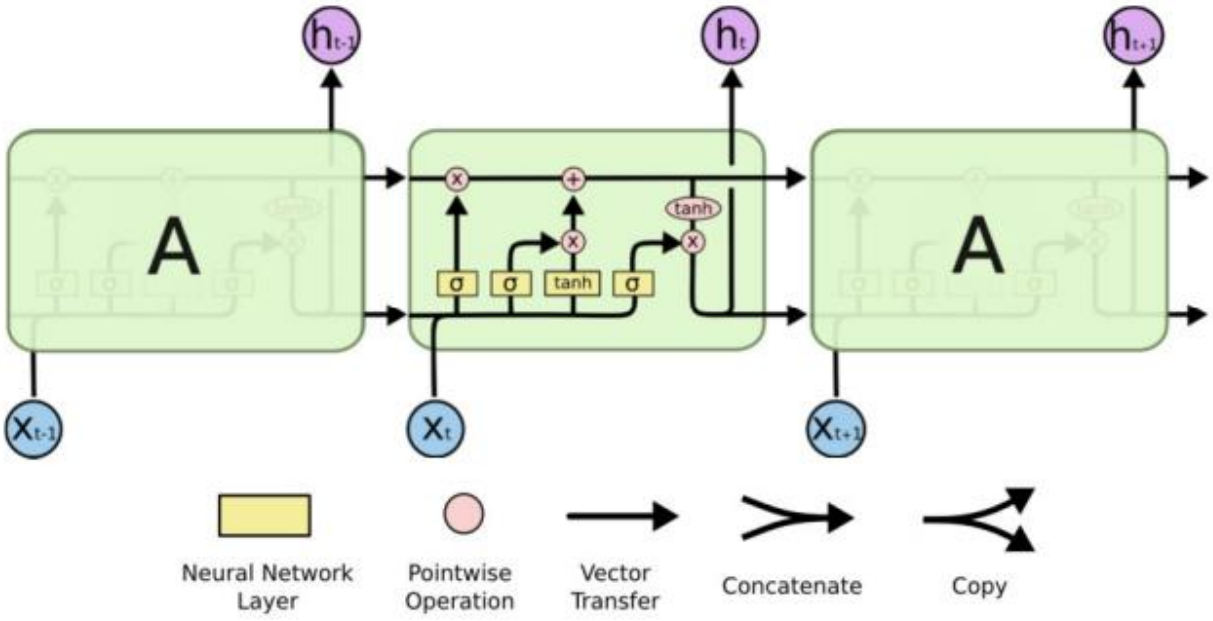

Рис. 3. Общая архитектура LSTM-сети

Ключевым компонентом модуля LSTM-сети является вектор состояния (рисунок 4). Вектор состояния регулируется специальными фильтрами, они контролируют обновление и удаление содержащейся в нем информации. Сигмоидальный слой в фильтре определяет, какую часть информации следует перезаписать в векторе состояния (0 – не записывать ничего, 1 –записать все).

Условные обозначения здесь:

- x_t - входной вектор в момент времени t,
- h_t - выходной вектор в момент времени t,
- C_t - вектор состояний в момент времени t,
- W_k - матрица параметров слоя k, то есть веса связей,
- b_k - вектор смещений выходов слоя k,
- f_t - вектор фильтра забывания в момент времени t,
- i_t - вектор входного фильтра в момент времени t,
- o_t - вектор выходного фильтра в момент времени t.

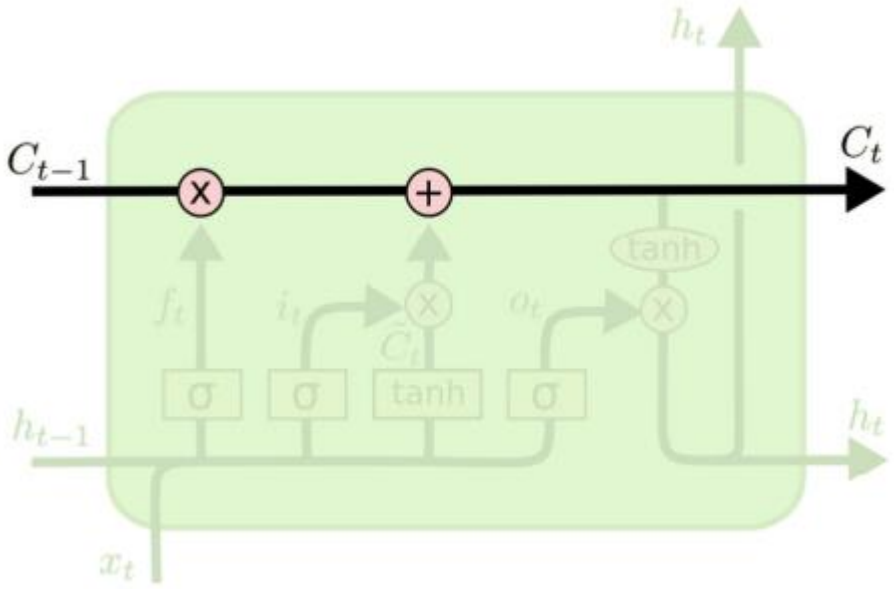

Рис. 4. Состояние модуля LSTM-сети

Принцип работы модуля LSTM [9]:

1. Слой фильтра забывания определяет, какую информацию можно удалить из вектора состояния (Рисунок 5a). Для каждого числа из состояния C_{t-1} возвращается число от 0 до 1 (где 0 означает, что значение должно быть забыто).

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \quad (9)$$

2. Сигмоидальный слой входного фильтра определяет, какие значения должны быть обновлены, а тангенс-слой строит вектор новых значений ε_t, которые могут быть добавлены в состояние C_t (Рисунок 5b).

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \quad (10)$$

$$\varepsilon_t = tanh(W_c \cdot [h_{t-1}, x_t] + b_c) \quad (11)$$

3. Для обновления состояния необходимо умножить старое состояние на f_t и прибавить к нему $i_t * \varepsilon_t$ (Рисунок 5c).

4. Выход LSTM-сети представляет собой состояние C_t с примененной к нему функцией активации, при этом выходной фильтр определяет, какие именно элементы состояния выводить (Рисунок 5d).

$$h_t = o_t * \tanh(C_t) \quad (12)$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (13)$$

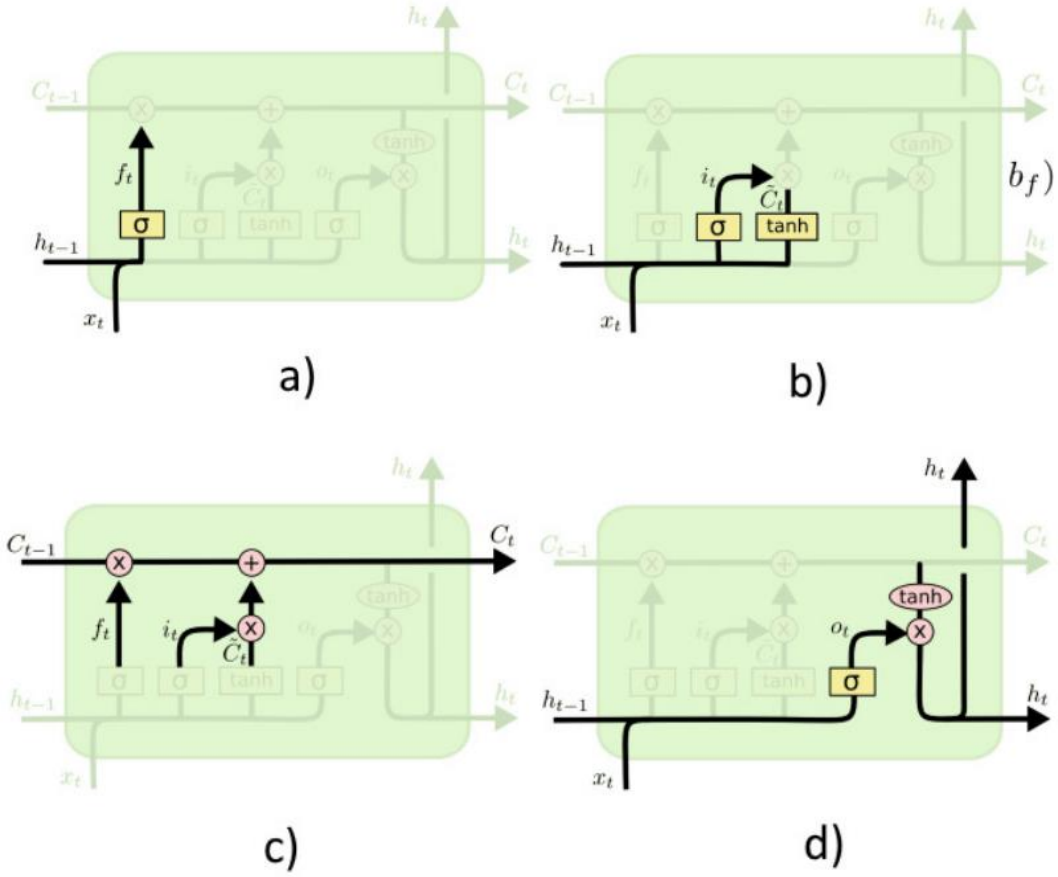

Рис. 5. Принцип работы LSTM-сети

Преимущества нейронных сетей:

- при удачном выборе параметров имеют очень высокую точность;
- являются универсальным аппроксиматором непрерывных функций;
- поддерживают инкрементное обучение.

Недостатки нейронных сетей:

- поскольку для настройки сети используются градиентные методы, то существует вероятность возможной расходимости или медленной сходимости;
- для достижения высокой точности требуется очень большой объем данных;
- низкая скорость обучения;
- сложная интерпретируемость параметров алгоритма.

1.6 ОЦЕНКА КАЧЕСТВА КЛАССИФИКАЦИИ

Основной проблемой при оценке методов классификации текстов является отсутствие стандартных протоколов сбора данных. Даже если бы существовал общий метод сбора данных, простой выбор разных обучающих и тестовых выборок может привести к несоответствиям в эффективности модели.

Качество работы классификатора оценивается на тестовой выборке. В то же время, оценкой работы системы занимается эксперт. Оценки качества работы классификатора приведены в таблице 1.

Таблица 1

Оценка качества работы классификатора

Класс c_i		Экспертная оценка	
		Положительная	Отрицательная
Оценка системы	Положительная	TP	FP
	Отрицательная	FN	TN

В таблице приняты следующие условные обозначения: TP – истинно положительное решение; TN – истинно отрицательное решение; FP – ложно положительное решение; FN – ложно отрицательное решение.

Основным критерием оценки качества классификации является сочетание точности и полноты.

Точность (precision) классификации внутри класса – это доля документов, найденных классификатором и фактически принадлежащих этому классу, по отношению ко всем документам, которые система присвоила этому классу.

Полнота (recall) классификации – это доля найденных классификатором документов, которые действительно принадлежат классу, относительно всех документов этого класса в тестовой выборке.

Согласно определению, точность вычисляется следующим образом:

$$p = TP / (TP + FP). \quad (14)$$

Полнота рассчитывается по формуле

$$r = TP / (TP + FN). \quad (15)$$

F-мера – одна из самых популярных агрегированных оценочных метрик для оценки качества классификатора. Она сочетает в себе информацию о точности и полноте следующим образом:

$$F_\beta = \frac{(\beta^2+1)\,pr}{\beta^2 \cdot p+r}, \quad (16)$$

где $0 \leq \beta < \infty$.

При $0 \leq \beta < 1$ большее значение имеет точность.

При $\beta = 1$ точность и полнота равноправны, тогда F_β = 2pr / (p + r).

При $1 < \beta < \infty$ большее значение имеет полнота.

Часто можно встретить другую формулу для вычисления точности (accuracy). Эту величину иногда называют правильностью или аккуратностью метода:

$$accuracy = \frac{\mathrm{TP} + \mathrm{TN}}{TP+TN+FP+FN} \quad (17)$$

1.7 ИСПОЛЬЗУЕМЫЕ ТЕХНОЛОГИИ

В качестве основного языка разработки был выбран язык Python [10], потому что для данного языка программирования имеется большое количество готовых библиотек для работы с большими массивами данных (выгрузка, обработка, визуализация), фреймворков для реализации нейронных сетей и алгоритмов машинного обучения, а также для него существует большое количество готовых решений для разработки Telegram бота.

Так для предобработки данных использовались библиотеки Pandas [11] и Scikit-learn [12]. Для визуализации и построения графиков библиотека matplotlib [13]. Для построения модели на основе алгоритмов машинного обучения использовалась библиотека Scikit-learn, а для модели на основе нейросети фреймворк TensorFlow [14] с библиотекой Keras [15].

Реализация Telegram бота производилась на основе библиотеки AIOGram [16]. Данная библиотека обладает асинхронностью, благодаря чему скорость работы приложения получается выше последовательного аналога за счет выполнения нескольких задач параллельно. Особенно актуально это для

приложений, которые задействуют большое количество подключений через сокет [17], например Telegram бота.

Для обработки записей звонков использовались:

- Библиотека SpeechRecognition [18], которая позволяет распознавать речь с помощью готовых API (CMU Sphinx, Google Speech Recognition, Microsoft Bing Voice Recognition, IBM Speech to Text и так далее).
- Библиотека Soundfile [19] использовалась для конвертации из .ogg формата в .wav формат, так как библиотека SpeechRecognition может работать только с данным форматом звуковых файлов.
- Программа FFmpeg [20] использовался для перекодирования звуковых файлов в кодек Opus, так как именно с таким кодеком работает облачный сервис Yandex.Cloud [21].
- Облачный сервис Yandex.Cloud также использовался для распознавания речи с целью сравнения с библиотекой Speech Recognition.

ГЛАВА 2. РАЗРАБОТКА СИСТЕМЫ АВТОМАТИЧЕСКОЙ ОБРАБОТКИ ОБРАЩЕНИЙ

2.1 ИМЕЮЩИЕСЯ ДАННЫЕ

Имеется готовая база данных PostgreSQL с ранее заполненными и обработанными обращениями от граждан города Тюмени. PostgreSQL – одна из самых старых и проверенных временем СУБД. К ней прилагаются подключаемые модули для разбора запросов на естественном языке, для построения многомерных индексов, для создания собственных типов данных и для многого другого. А также она работает на самых разных платформах [22].

В данной базе данных имеется две таблицы: таблица с категориями обращений и таблица самих обращений. Структура этих таблицы приведены в таблицах 2 и 3 соответственно.

Таблица 2

Структура таблицы категорий обращений

Название колонки	**Описание**	**Тип данных**
id	Идентификационный номер записи	Integer
title	Название категории обращения	Varchar
hotline_requst_category_id	Родитель типа обращения (id)	Integer
is_claim	Булевский флаг - является ли тип жалобой	Boolean
is_info	Булевский флаг - является ли тип справочным	Boolean
organization_id	К какой организации направляется (id)	Integer

Всего в таблице категорий обращений находится 96 записей.

Таблица 3

Структура таблицы обращений

Название колонки	**Описание**	**Тип данных**
id	Идентификационный номер записи	Integer
hotline_requst_category_id	Внешний ключ категории обращения	Integer

Продолжение таблицы 3.

fio	ФИО обратившегося	Varchar
phone_number	Телефон обратившегося	Varchar
datetime	Дата и время происшествия	Datetime
route_id	Id маршрута происшествия (необязательно)	Integer
checkpoint_id	Id остановки происшествия (необязательно)	Integer
gosnumber	Гос. номер ТС (необязательно)	Varchar
text	Текст обращения	Text
result	Результат рассмотрения (принято к устранению, учтено в работе, решение отрицательное, решение положительное, принято к сведению)	Varchar

Всего в таблице обращений за выбранный период с 1 января по 30 ноября 2021 года находится 165833 записей.

Также существует хранилище записей телефонных разговоров в формате .ogg. Данное хранилище имеет следующую структуру: год/номер месяца/день месяца/время звонка_номер телефона.ogg.

2.2 ПОДГОТОВКА ДАТАСЕТА

Для подготовки датасета был выбран период с 1 января по 30 ноября 2021 года. Всего обращений за данный период было 165833, но большинство из них относилось к категории справочной информации (узнать во сколько подойдет транспортное средство на остановку, как доехать в определенное место и так далее). Однако ввиду того, что данная категория не рассматривалась для классификации, обращения, относящиеся к данному типу, были исключены из выборки. После чего всего за выбранный период осталось 16109 записей.

Изначально для перевода записей телефонных разговоров в текстовый формат использовалась библиотека SpeechRecognition с бесплатным API Google Speech Recognition. Однако после обработки одного месяца данных было

замечено, что около 50% текста распознавалось неверно, а некоторые слова вовсе не были распознаны. Ввиду этого было опробовано облачное решение Yandex.Cloud с бесплатным пробным периодом, позволяющим обработать до 3000 минут записей.

Для обработки сервис от Яндекс принимает записи в кодеке Opus, но так как имеющиеся записи были закодированы в формате Vorbis их потребовалось перекодировать в нужный формат за счет использования консольной утилиты FFmpeg. Бесплатного периода пользования сервисом Yandex.Cloud хватило больше чем на месяц записей обращений и была замечена гораздо большая точность распознавания: менее 25% слов в тексте были распознаны некорректно, а слов, которые не удалось распознать, практически не было. Для обработки оставшихся данных с февраля по конец ноября использовалась библиотека SpeechRecognition с тем же API Google Speech Recognition.

Далее происходила привязка существующих обращений в базе данных к текстам обращений полученных из записей разговоров. Данная привязка производилась путем соотнесения даты, времени звонка и номера телефона. Затем полученный набор данных был сохранен в формате “.CSV“ для дальнейшей обработки.

Код скрипта для распознавания записей разговоров и привязки их к существующим записям приведен в Приложении 1.

2.3 ПРЕДОБРАБОТКА ДАННЫХ

Для предобработки данных полученный датасет был занесен в объект DataFrame библиотеки Pandas. Данный объект представляет собой 2-мерную таблицу с данными. На рисунке 6 представлена получившаяся таблица.

	id	type	parent_type	datetime	route	checkpoint	car_number	processed_text	call_dir	full_text
0	2261785	Пропуск рейсов	Жалоба	2021-01-01 07:29:18.633984+05:00	41	ул. Лесопарковая	NaN	В период времени с 07:00 до 07:29 не было ТС.	45_ _1.ogg	NaN
1	2261803	Пропуск рейсов	Жалоба	2021-01-01 08:10:49.458588+05:00	8	ул. Алебашевская	AA865 72	Не выполнен рейс по расписанию в 08:03	5[_ ɔgg	NaN
2	2261815	Пропуск рейсов	Жалоба	2021-01-01 08:23:53.580902+05:00	45	ул. Созидателей	NaN	Не выполнен рейс по расписанию в 06:30 и 06:40.	50_ ^gg	NaN
3	2261817	Пропуск рейсов	Жалоба	2021-01-01 08:25:51.726891+05:00	24	Геологоразведка	AM361 72	Не выполнен рейс по расписанию в 08:13	22_ ogg	NaN
4	2261831	Нерегулярный рейс (отклонение более ± 3 мин.)	Жалоба	2021-01-01 08:48:47.657318+05:00	129к	с/о Водник	AO511 72	Рейс по расписанию в 08:50 водитель выполнил с...	00_ _1.ogg	NaN
...	...	...	...	...	...	...	...	...	...	...
16104	2429832	Грубость водителя или кондуктора	Жалоба	2021-11-30 19:02:35.411082+05:00	8	мкр. Восточный	AM535 72	Кондуктор допускает грубое, неуважительное отн...	32_ 1.ogg	я слушаю говорите Добрый вечер подскажите пожа...
16105	2429850	Водитель не открыл дверь для посадки (высадки)	Жалоба	2021-11-30 19:38:40.984135+05:00	36к	II Заречный микрорайон	AM527 72	Водитель остановился, но двери не открыл для ...	29_ _1.ogg	Добрый день Ну что Говорите Здравствуйте Скажи...
16106	2429851	Грубость водителя или кондуктора	Жалоба	2021-11-30 19:39:20.131677+05:00	77	Автовокзал	AM357 72	Водитель допускает грубое, неуважительное отно...	51_ .ogg	Здравствуйте слушаю вас Здравствуйте Скажите п...
16107	2429852	Высокая скорость, резкое торможение	Жалоба	2021-11-30 19:40:25.661523+05:00	77	Автовокзал	AM357 72	По мнению заявителя водитель допускает высокую...	51_ 1.ogg	Здравствуйте слушаю вас Здравствуйте Скажите п...
16108	2429915	Пропуск рейсов	Жалоба	2021-11-30 21:46:51.734001+05:00	51	мкр. Восточный	AB855 72	Не выполнен рейс по расписанию в 21:32	46_ _1.ogg	Здравствуйте слушаю вас по расписанию До сколь...

Рис. 6. Датасет

Для подсчета количества записей использовалось свойства shape объекта Dataframe. Число записей составило 16109.

Для подсчета числа пропущенных значений использовался метод isna(). Полученные результаты представлены в таблице 4.

Таблица 4

Количество пропущенных значений по каждому из признаков

Признак	**Количество пропущенных значений**
id	0
Тип обращения	0
Родительский тип обращения (жалоба, предложение, благодарность)	0
Дата время происшествия	0
Маршрут	92
Остановка	1609
Гос. номер	3094
Обработанный оператором текст	0
Директория расположения звонка	0
Текст обращения	3392

Для получения описательной статистики был использован метод describe(). Ниже в таблице 5 представлены результаты для столбца типов обращений.

Таблица 5

Описательная статистика по типам обращений

Характеристика	**Значение**
количество значений	16109
уникальных значений	71
самое встречаемое значение	Нарушение схемы м-та (проезд остановочного п-та)
частота встречаемости	2688

Всего после фильтрации остался 71 тип обращения. Распределение данных по этим типам достаточно несбалансированно и их количество слишком велико, из-за чего точность итогового классификатора может заметно снизиться [23]. На рисунке 7 приведено распределение обращений для 30 самым встречаемых типов.

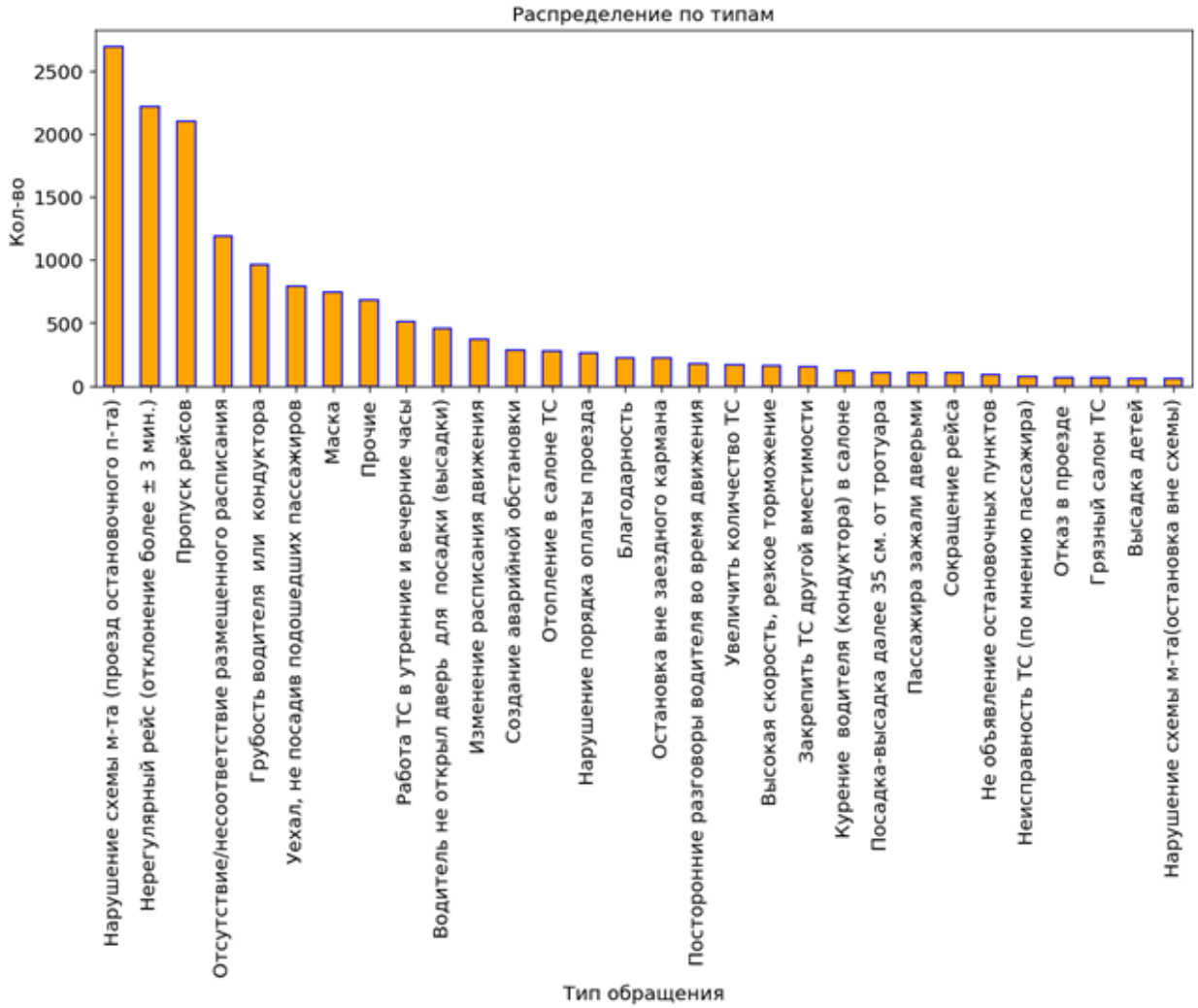

Рис. 7. Распределение обращений по исходным типам

Для уменьшения количества категорий и их балансировки было введено 11 новых типов и уже имеющиеся категории обращений были сгруппированы по ним. В таблице 6 представлены получившееся типы обращений с количеством обращений для каждого из них.

Таблица 6

Новые категории обращений

Тип обращения	**Количество записей**
Нарушение установленного расписания	4858
Нарушение установленной схемы	2852
Культура обслуживания пассажиров	2621
Нарушение правил посадки/высадки	1816
Состояние транспортной инфраструктуры	1242
Создание небезопасных условий проезда	829
Предложения	814
Содержание транспортного средства	455
Нарушение порядка оплаты проезда	324
Благодарность	223
Нарушение правил благоустройства	44

Описательная статистика для новых типов обращений с помощью метода describe() представлена в таблице 7.

Таблица 7

Описательная статистика по новым типам обращений

Характеристика	**Значение**
количество значений	16109
уникальных значений	11
самое встречаемое значение	Нарушение установленного расписания

Продолжение таблицы 7.

частота встречаемости	4858

Визуализация распределения значений для категориальных признаков производилась с помощью метода hist() библиотеки Pandas. Результаты для даты (месяца) внесения обращения в систему и новых типов представлены ниже на рисунках 8 и 9 соответственно.

Рис. 8. Распределение обращений по месяцам

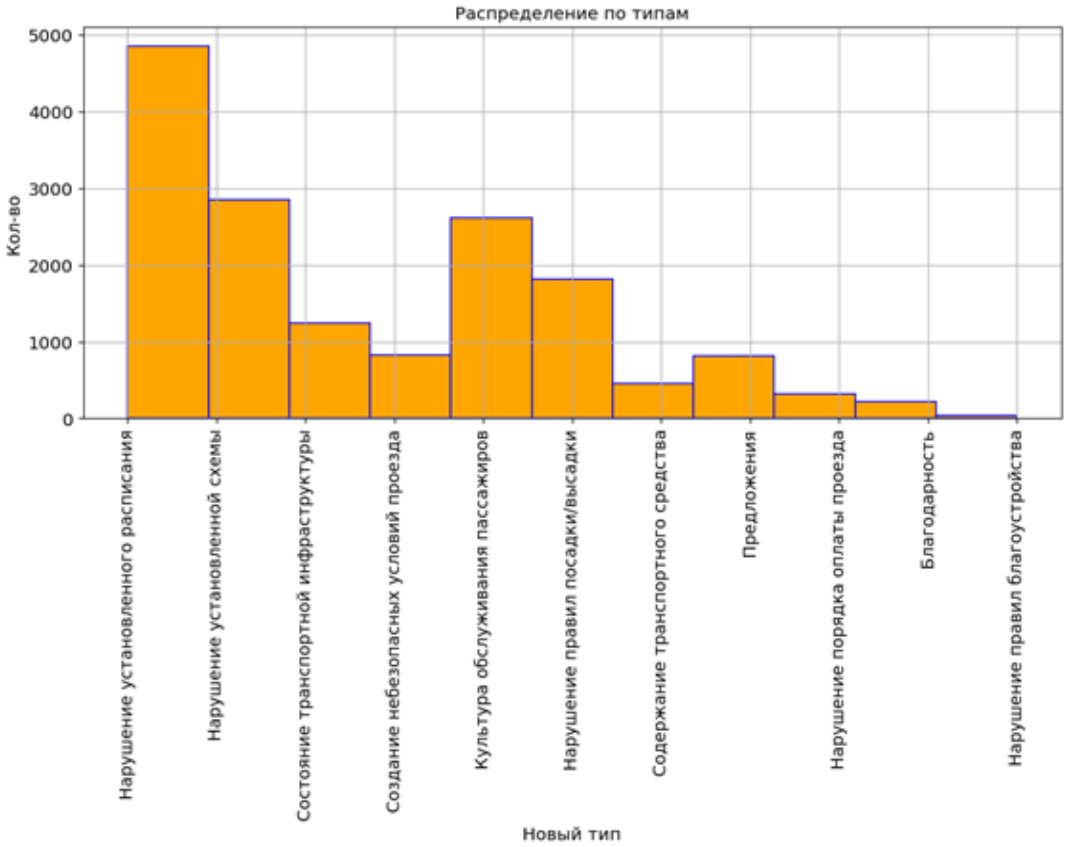

Рис. 9. Распределение обращений по новым типам

2.4 ПОДГОТОВКА ТЕКСТА

Для подготовки текстов обращений была произведена их очистка от ненужных числовых символов, лишних пробелов и символов пунктуации, а все оставшиеся слова были переведены в нижний регистр. Затем были удалены стоп-слова посредством библиотеки stop_words. На последнем этапе обработки текста производился стемминг и лемматизация слов с помощью библиотек nltk и pymorphy2 соответственно. По итогу, стемминг для каждой из моделей классификаторов показывал результаты в среднем на 1-2% лучше и в итоговых моделях использовался именно он.

Далее для анализа обработанного текста было построено облако слов с помощью библиотеки wordcloud. Получившееся облако слов приведено на рисунке 10.

Рис. 10. Wordcloud после обработки текста

Также был выполнен подсчет статистики встречаемости слов в тексте с помощью класса FreqDist библиотеки nltk. Полученные результаты для 30 самых встречаемых слов представлены в таблице 8.

Таблица 8

Статистика встречаемости слов в тексте

Слово	**Количество**
остановк	14716
автобус	11526
информац	9951
минут	8984
номер	6656
расписан	6544
сторон	6397
маршрут	6031
говор	5803
сто	5465
как	4892
маршрутк	4884
водител	4856
мог	4641
зафиксирова	4145
скаж	3944
город	3905
посмотр	3863
ваш	3774
еха	3522

Продолжение таблицы 8.

останов	3510
перед	3448
проеха	3365
уг	3227
уеха	3141
зафиксир	3052
зна	2896
фамил	2812
руководств	2713
нет	2662

2.5 РАЗРАБОТКА КЛАССИФИКАТОРА ОБРАЩЕНИЙ

Для разработки классификатора обращений использовалось два подхода: с использованием алгоритмов машинного обучения на основе библиотеки Scikit-learn и с использованием нейронных сетей на основе фреймворка TensorFlow и библиотеки Keras.

Подход с применением алгоритмов машинного обучения заключался в использовании готового линейного классификатора SGDClassifier. Предварительно производилась векторизация текстов обращений с использованием класса TfidfVectorizer, а категории обращений кодировались в числовое значение с использованием класса LabelEncoder. После чего датасет разбивался на тренировочную и тестовую выборку в соотношении 70/30.

Далее для подбора оптимальных параметров классификатора задавались списки со значениями параметров для настройки:

- параметр регуляризации penalty: [L1, L2, ElasticNet];
- коэффициент регуляризации alpha: [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5];
- параметр максимального количества проходов по обучающим данным max_iter: [5, 10, 15, 20, 25, 35, 50, 100, 500, 1500].

Во всех экспериментах использовалась функция потерь “hinge”, что означает использование линейной модели SVM. Наилучшие результаты модель показала с параметрами: penalty=’ ElasticNet’, alpha= 0.0001, max_iter=15. Так точность классификации составила 87.8% на тренировочной выборке и 71.04% на тестовой выборке, а значение показателя F1 score составило 70.72%.

На рисунке 11 представлен отчет классификации для всех категорий обращений. Как можно заметить, хуже всего классификатор справился с категориями 4 и 7, которые представляла собой типы “Предложения” и “Содержание транспортного средства” соответственно. Связано это с тем, что предложения могут относиться как к расписанию движения, так и к схеме маршрута или к транспортной инфраструктуре, то есть наблюдается достаточно сильная корреляция между этими категориями обращений. А категория “Содержание транспортного средства” достаточно часто классифицировалась как категория “Культура обслуживания пассажиров” (номер 5), что можно наблюдать на рисунке 12, на котором представлена матрица ошибок.

Лучше всего классификатор справился с категорией 8, которая представляла собой тип “Нарушение установленной схемы”. Данная категория находится на втором месте по количеству обращений после категории “Нарушение установленного расписания” и по содержанию они достаточно сильно отличаются от обращений остальных типов.

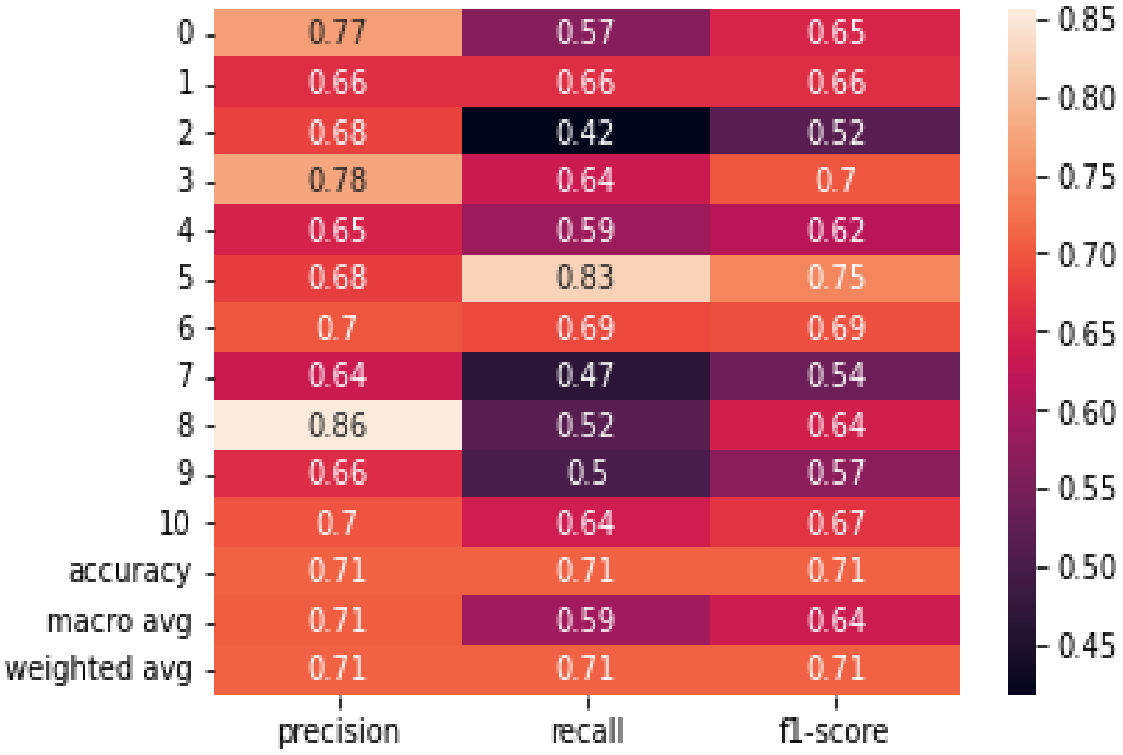

Рис. 11. Отчет классификации для линейного классификатора

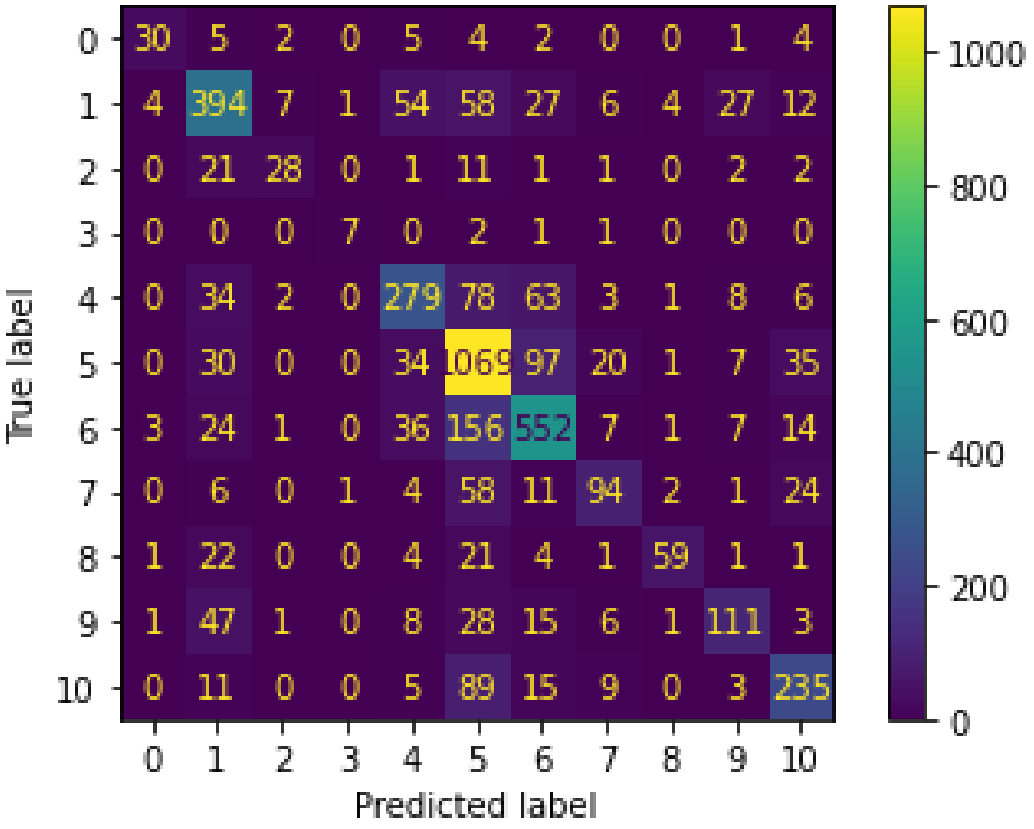

Рис. 12. Матрица ошибок линейного классификатора

Для классификатора с использованием нейронных сетей текст предварительно токенизировался с помощью класса Tokenizer, а затем производилась векторизация полученных токенов с помощью метода Tfidf. Категории обращений кодировались в числовое значение с использованием метода to_categorical(). Как и ранее датасет разбивался на тренировочную и тестовую выборку в соотношении 70/30. Далее было создано две модели: на основе простого плотно связанного слоя Dense и на основе слоя LSTM. Для обоих этих моделей формировался стек слоев Sequential.

Более простая модель состоит из плотно связанного слоя Dense с 512 узлами в нем и функцией активации “Relu”, затем идет слой Dropout с параметром регуляризация равным 0.6 для борьбы с переобучением модели и в самом конце находится еще один слой Dense с 11 узлами, что соответствует количеству категорий для классификации. Функцией активации на последнем слое была указана функция “softmax”. Представление модели с помощью функции summary можно увидеть на рисунке 13.

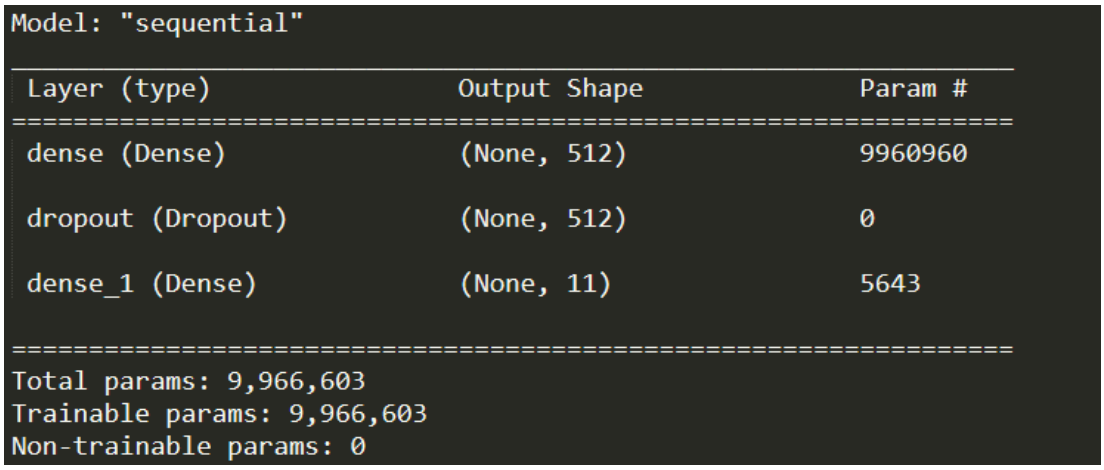

Рис. 13. Сводное представление простой плотно связанной модели

Далее модель настраивалась для обучения с помощью метода compile(), при этом использовался оптимизатор “adam”, функция потерь “categorical_crossentropy” и метрика для оценки модели “accuracy”. Так для этой модели при 5 эпохах и размере batch_size равным 128 точность на тренировочной выборке составила 90.2%, а на тестовой выборке 65.1%.

На рисунках 14 и 15 представлены графики точности и потерь модели соответственно. На основе этих графиков можно сделать вывод, что необходимо остановиться на 2 эпохах (на графиках индексация начинается с 0), так как далее точность только падает, при этом точность модели на тестовой выборке выросла до 67.4%.

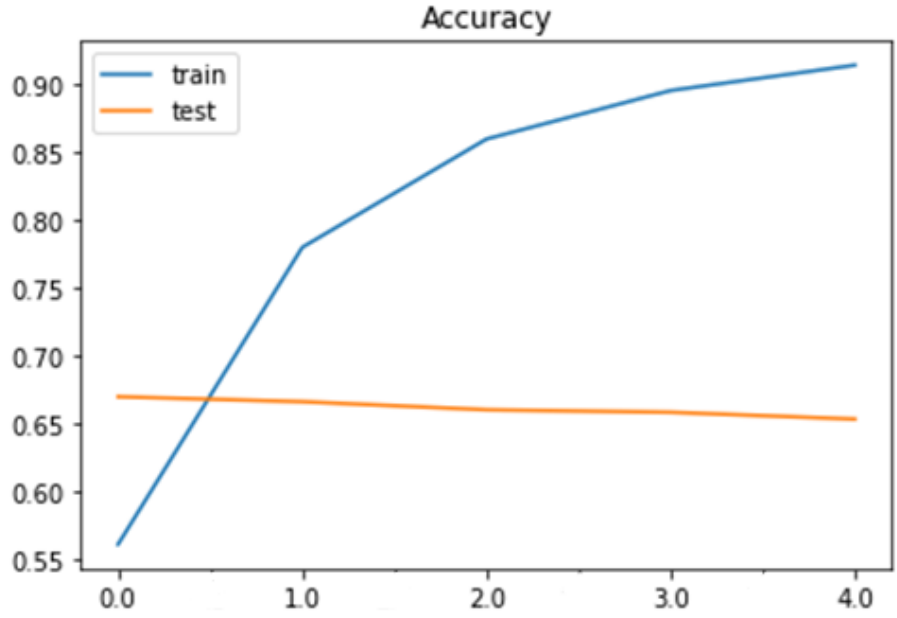

Рис. 14. График потерь простой плотно связанной модели

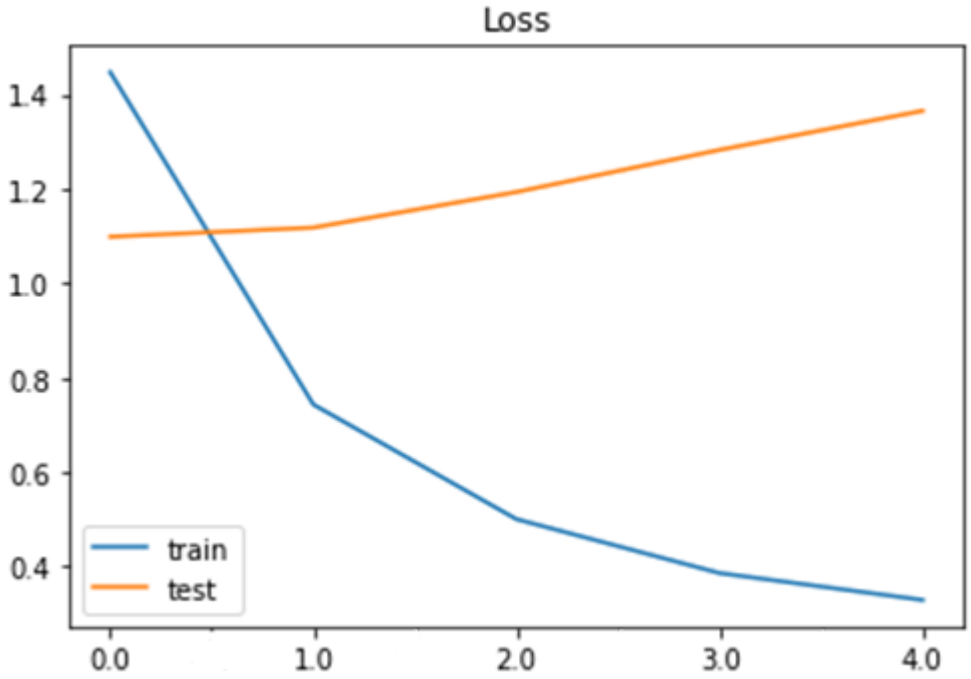

Рис. 15. График точности простой плотно связанной модели

На рисунке 16 представлен отчет классификации для всех категорий обращений. На этот раз хуже всего классификатор справился с категориями 4, 6 и 7, которые представляли собой типы “Предложения”, “Нарушение порядка оплаты проезда” и “Содержание транспортного средства” соответственно.

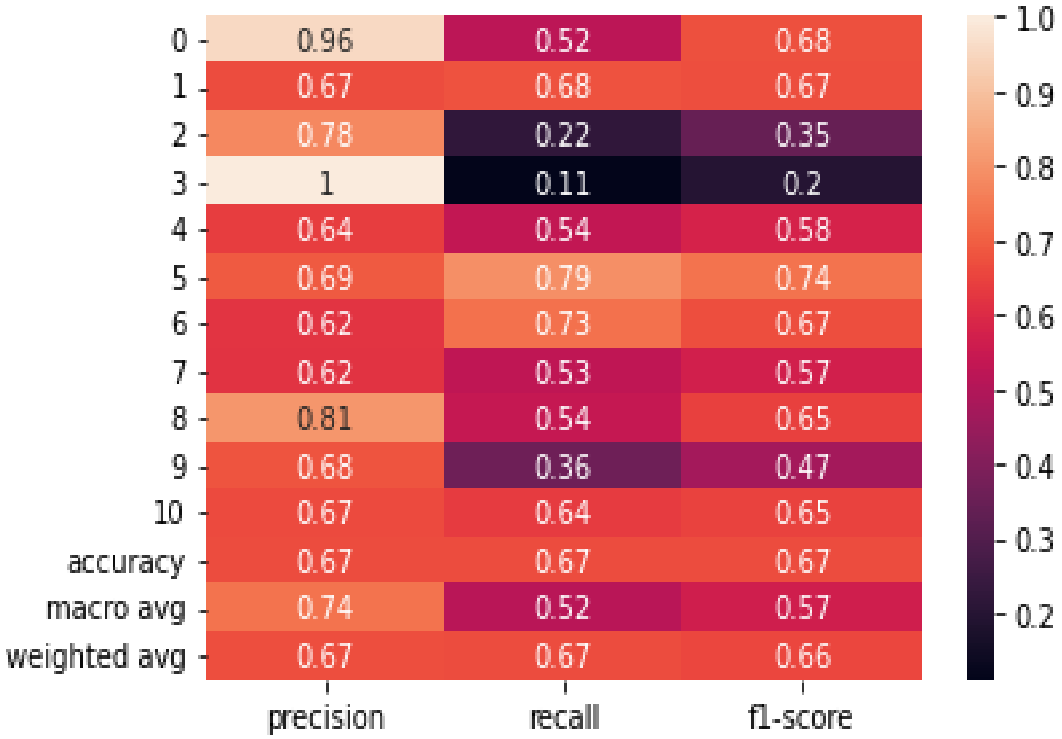

Рис. 16. Отчет классификации для простой плотно связанной модели

На рисунке 17 представлена матрица ошибок, на которой можно заметить аналогичную проблему при классификации для категорий 4, 6 и 7 (“Предложения”, “Нарушение порядка оплаты проезда”, “Содержание транспортного средства”).

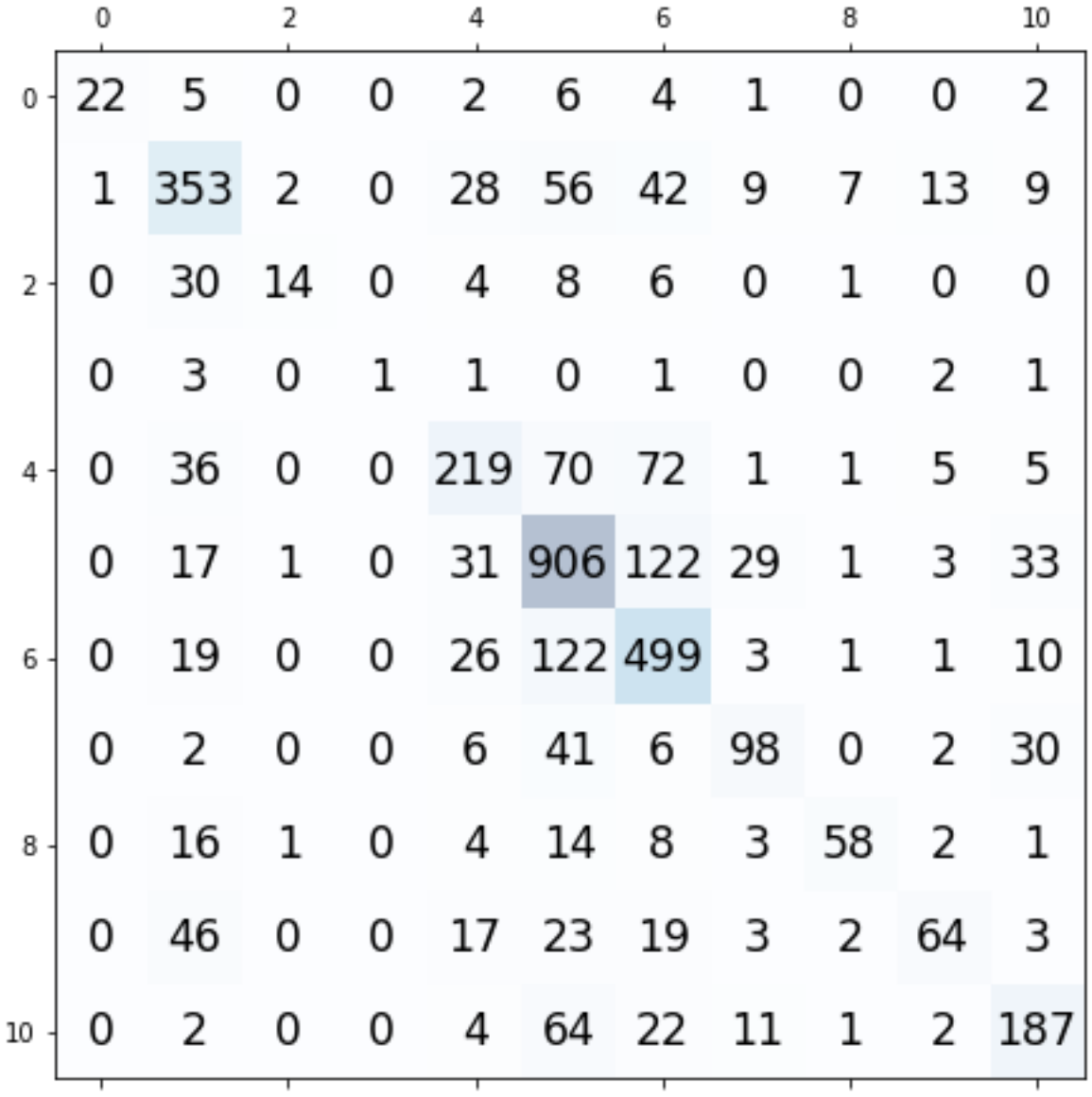

Рис. 17. Матрица ошибок простой плотно связанной модели

Модель на основе архитектуры LSTM состоит из слоя Embedding, который использовался для уменьшения размерности исходных векторов слов. В параметрах данного слоя размер словаря задавался равным 50000 и размерность выходной матрицы равной 100. Затем идет слой SpatialDropout1D с параметром регуляризации равным 0.2, далее расположен рекуррентный слой LSTM с параметром внутренних узлов равным 100. И в самом конце модели располагается плотно связанный слой Dense с 11 узлами, что соответствует количеству категорий для классификации. Функцией активации на последнем слое была указана функция “softmax”. Представление модели с помощью функции summary можно увидеть на рисунке 18.

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding (Embedding)       (None, 250, 100)          5000000

 spatial_dropout1d (SpatialD  (None, 250, 100)         0
 ropout1D)

 lstm (LSTM)                 (None, 100)               80400

 dense_2 (Dense)             (None, 11)                1111

=================================================================
Total params: 5,081,511
Trainable params: 5,081,511
Non-trainable params: 0
```

Рис. 18. Сводное представление модели на основе LSTM

Далее производилась настройка модели для обучения с помощью метода compile() со следующими параметрами: оптимизатор “adam”, функция потерь “categorical_crossentropy” и метрика для оценки модели “accuracy”. Так для этой модели при 10 эпохах и размере batch_size равным 64, точность на тренировочной выборке составила 79.7%, а на тестовой выборке 60.1%.

На рисунках 19 и 20 представлены графики точности и потерь модели соответственно.

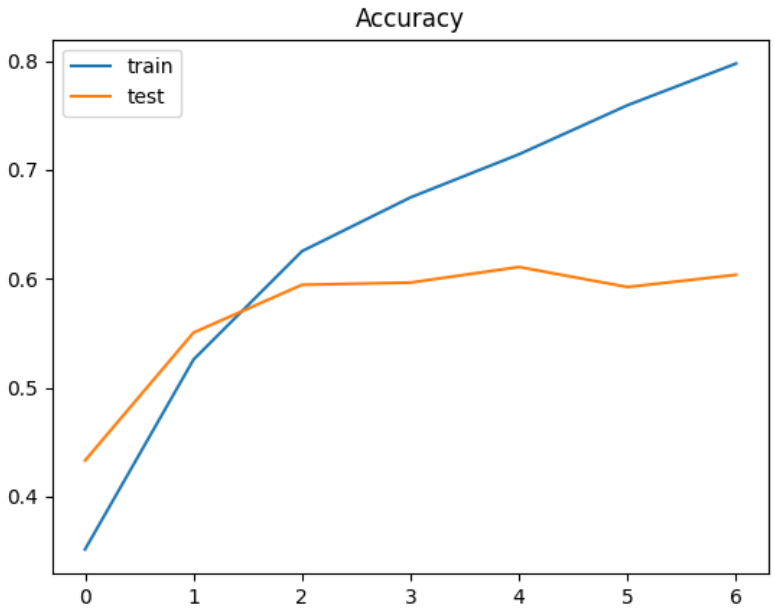

Рис. 19. График точности модели на основе LSTM

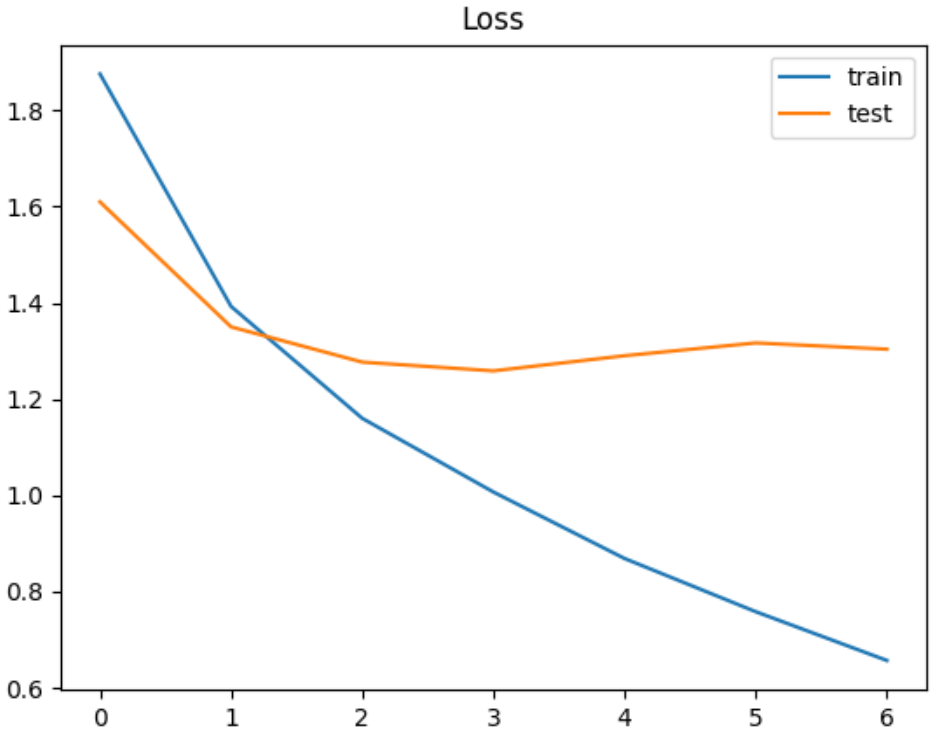

Рис. 20. График потерь модели на основе LSTM

На основе полученный результатов можно сделать вывод о том, что модели, основанные на нейронных сетях, показали себя хуже, чем линейный классификатор SVM. Связано это с тем, что для выявления зависимостей нейронным сетям требуется большее количество документов для обучения. Особенно это заметно при сравнении простой плотно связанной модели и модели на основе LSTM ввиду большей сложности архитектуры LSTM.

Код разработанных классификаторов приведен в Приложении 2.

2.6 ОПРЕДЕЛЕНИЕ ИСХОДНОЙ КАТЕГОРИИ ОБРАЩЕНИЯ

Так как исходные категории обращений были сгруппированы в 11 новых, необходимо после классификации реализовать определение изначальной категории там, где это возможно сделать. Для проведения эксперимента была выбрана категория “Нарушение установленного расписания”, так как обращений данного типа было больше всего и для данной категории достаточно точно можно алгоритмически определить какого рода нарушение было зафиксировано (проезд остановки, опоздание или что-то иное).

Так изначально производилась проверка на наличие отметки о прохождении транспортным средством требуемой остановки. Если отметка не была обнаружена, то обращению присваивалась категория “Пропуск рейсов”. В случае, если отметка о прохождении остановки транспортным средством все же была, происходит проверка на длительность отстоя. При превышении разрешенного времени простоя на остановке более 2 минут, обращение нужно отнести к категории “Простой на остановке более 2 мин”. Если время простоя находилось в границах разрешенного времени стоянки, то далее необходимо проверить насколько сильно транспортное средство отклонилась от графика. В случае отклонения более чем на 3 минуты, необходимо выбрать категорию “Нерегулярный рейс (отклонение более +- 3 мин.)”. Однако предварительно необходимо проверить плановое время прибытия на остановку, если оно находится в границах до 8 часов или после 21 часа, то для обращения необходимо выбрать категорию “Работа ТС в утренние и вечерние часы”, которая была введена как раз для таких случаев.

Для оставшихся 10 сгруппированных типов обращений алгоритмическое определение базовой категории не представляется возможным и для них необходимо разработать отдельные классификаторы, позволяющие определять исходную категория по тексту обращения. На текущем этапе эта обязанность будет закреплена за операторами горячей линии, так как подобное исследование не входит в рамки данной работы.

2.7 РАЗРАБОТКА TELEGRAM БОТА

Разработка Telegram бота велась с использованием библиотеки aiogram. Для обработки сообщений от пользователя были созданы три метода отмеченных декоратором `@dp.message_handler`: метод `command_start()` для обработки команды /start, метод `bot_message()` для обработки всех текстовых сообщений и метод `voice_message_handler()` для обработки голосовых сообщений.

При вызове команды /start выводится приветственное сообщение и появляются кнопки меню. Скриншот примера вывода представлен на рисунке 21.

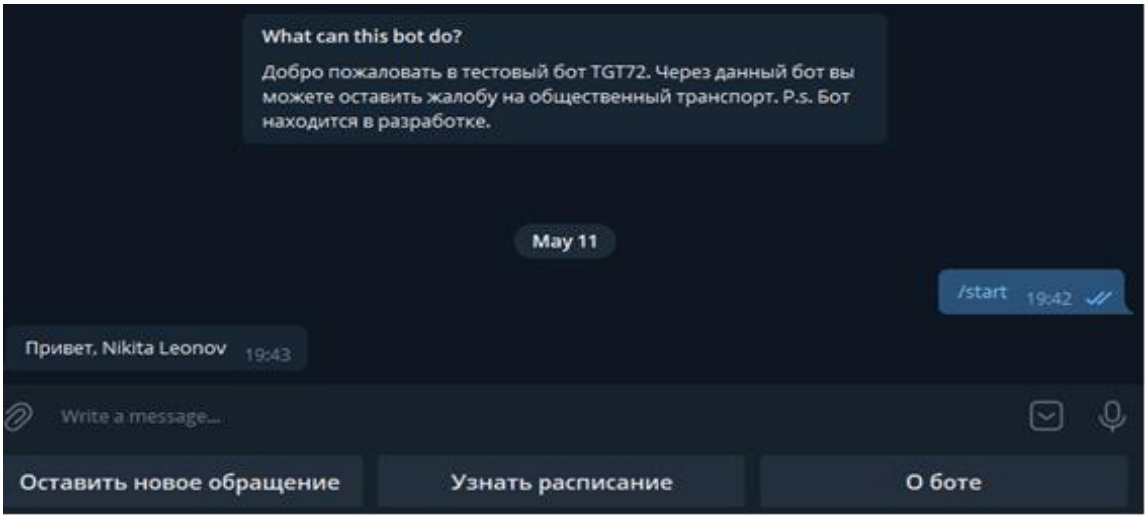

Рис. 21. Скриншот начального экрана бота

При нажатии кнопки “О боте” выводится справочная информация, скриншот данной информации представлен на рисунке 22.

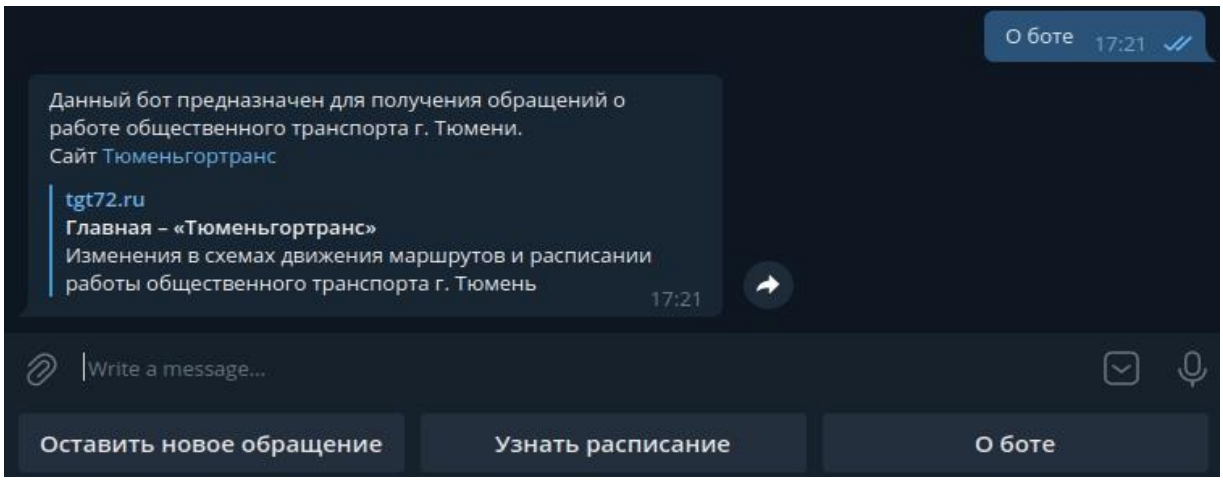

Рис. 22. Скриншот окна “О боте”

При нажатии кнопки “Оставить новое обращение” начинается процесс заполнения обращения. Последовательно у пользователя спрашивают все необходимые данные для заполнения. При этом некоторые необязательные данные (остановка, гос. номер ТС) можно пропустить с помощью кнопки меню “Пропустить”. При заполнении даты и времени происшествия, а также номера телефона происходит проверка на корректность введенных данных и если пользователь ввел информацию в некорректном формате, то его попросят ввести

их еще раз. На любом этапе заполнения обращения можно нажать на кнопку “Главное меню” для выхода из режима заполнения и возврата в главное меню. Также при вводе текста заявки есть возможность записать голосовое сообщение средствами самого Telegram и далее аудиофайл этого сообщения будет конвертирован с помощью библиотеки Soundfile в формат WAV и затем из него будет извлечен текст посредством библиотеки SpeechRecognition с бесплатным API Google Speech Recognition.

Пример вывода при заполнении обращения приведен на рисунке 23.

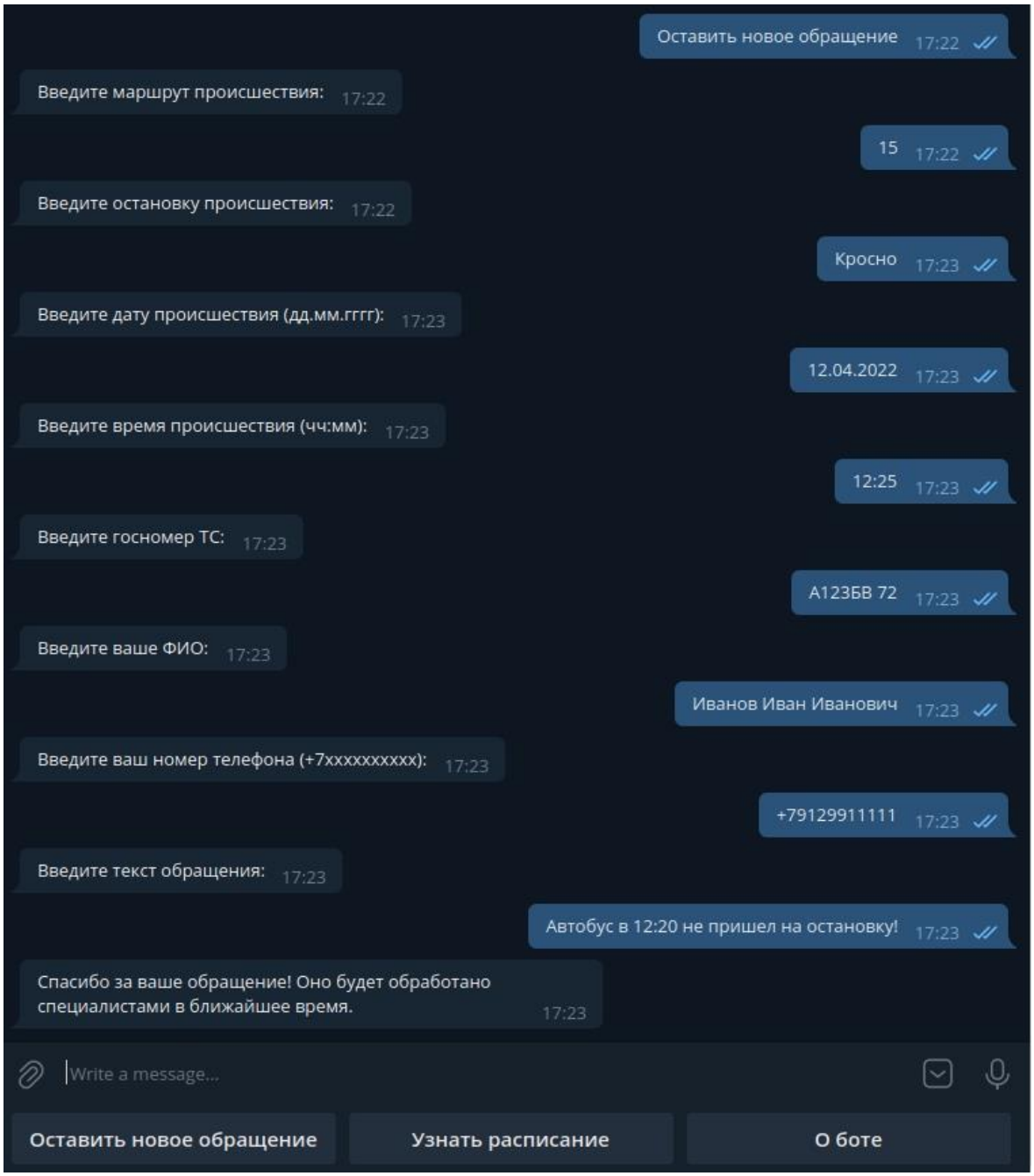

Рис. 23. Скриншот процесса заполнения обращения

Так как около 90% всех обращений на горячую линию поступает для того, чтобы узнать расписание прибытия ближайших транспортных средств, была добавлена возможность получения планового времени прибытия транспорта на остановку. Для этого в главном меню необходимо нажать на кнопку “Узнать расписание”, после чего пользователю отобразится информационное сообщение, а также пользователю будет необходимо ввести номер маршрута для поиска расписания. Скриншот данного информационного сообщения представлен на рисунке 24.

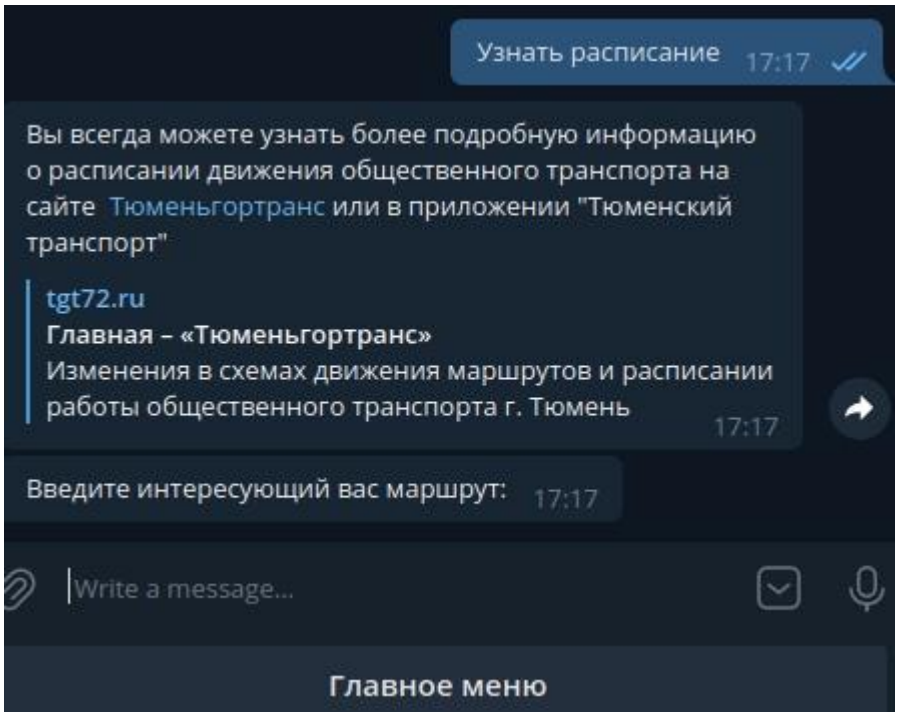

Рис. 24. Скриншот окна для получения расписания движения

После ввода номера маршрута, пользователю отобразится список всех остановок данного маршрута для выбора. Скриншот с примером данного окна для маршрута №16 представлен на рисунке 25.

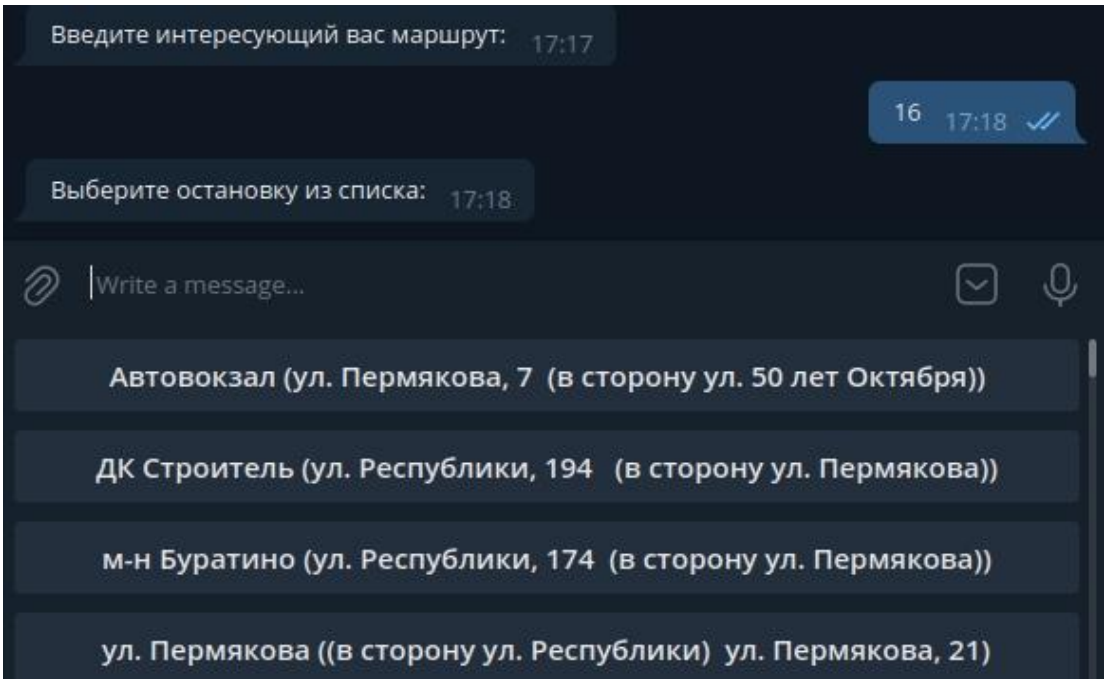

Рис. 25. Скриншот окна выбора остановки

Как только пользователь выберет остановку, ему отобразятся 10 ближайших с текущего момента времени пунктов расписания и будет произведен выход в главное меню.

Скриншот с примером вывода найденного расписания для остановки “ДК Строитель” представлен на рисунке 26.

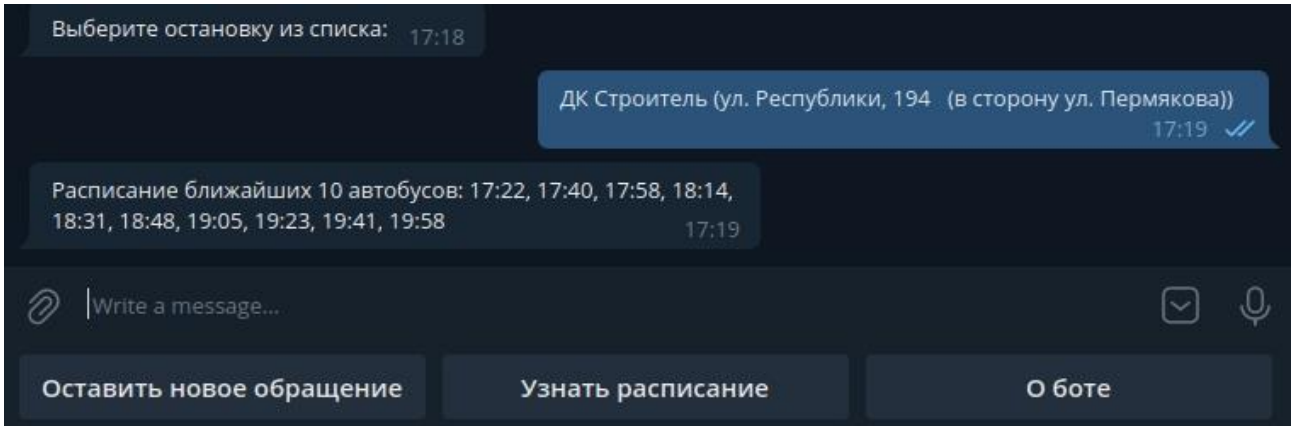

Рис. 26. Скриншот процесса заполнения обращения

Код Telegram бота приведен в Приложении 3.

ЗАКЛЮЧЕНИЕ

В ходе выполнения выпускной квалификационной работы были получены следующие результаты:

1. обработаны имеющиеся записи телефонных разговоров путем конвертации их в текстовый формат, а также был подготовлен датасет путем привязки телефонных записей обращений к готовым заявкам, хранящимся в системе;
2. выполнена предобработка и визуализация данных обращений с 1 января по 30 ноября 2021 года;
3. разработаны три классификатора для соотнесения обращений к существующим категориям на основе алгоритмов машинного обучения и нейронных сетей, а также проведено их сравнение;
4. реализован Telegram бот для получения необходимых для заполнения обращения данных от пользователя с дополнительной возможностью получения расписания движения общественного транспорта.

В результате было произведено улучшение существующей системы обработки обращений на горячую линию МКУ «Тюменьгортранс», что поспособствовало повышению скорости работы горячей линии за счет уменьшения очереди из звонящих, что в свою очередь привело к снижению нагрузки на операторов горячей линии.

В будущем данная работа может быть улучшена, а именно возможно использовать платные сервисы по распознаванию речи, что позволит получить более качественные исходные данные, тем самым, будет повышена точность классификации разработанных моделей.

Также может быть доработан Telegram бот, а именно можно улучшить интерфейс взаимодействия с пользователем, а также добавить больше функций для ответа на наиболее часто встречающиеся вопросы (поиск маршрута, вопросы по обслуживанию в салоне транспортных средств и тому подобные). Также будет полезно реализовать голосовой бот для телефонной линии, так как остаются

пользователи, такие как пожилые люди и люди с нарушением зрения, которые не имеют возможности пользоваться Telegram ботом, сайтом или мобильным приложением “Тюменьгортранс”.

СПИСОК ЛИТЕРАТУРЫ

1. Aggarwal C. C. Data Classification. Algorithms and applications. – 2014. – С. 245-273.
2. Батура Т. В. Методы автоматической классификации текстов //Программные продукты и системы. – 2017. – Т. 30. – №. 1. – С. 85-89.
3. Zhang X., Zhao J., LeCun Y. Character-level convolutional networks for text classification //Advances in neural information processing systems. – 2015. – Т. 28.
4. Ju R. et al. An efficient method for document categorization based on word2vec and latent semantic analysis //2015 IEEE International Conference on Computer and Information Technology; Ubiquitous Computing and Communications; Dependable, Autonomic and Secure Computing; Pervasive Intelligence and Computing. – IEEE, 2015. – С. 2276-2283.
5. Pontiki M., Galanis D., Pavlopoulos J., Papageorgiou H., Androutsopoulos I., Manandhar S. SemEval-2014 Task 4: Aspect based sentiment analysis. The 8th Intern. Workshop on Semantic Evaluation (SemEval 2014). Dublin, Ireland. 2014, С. 27–35.
6. Tarasov D. S. Deep recurrent neural networks for multiple language aspect-based sentiment analysis of user reviews //Proceedings of the 21st international conference on computational linguistics dialog. – 2015. – Т. 2. – С. 53-64.
7. Deep Learning, NLP, and Representations Posted [Электронный ресурс] URL: http://colah.github.io/posts/2014-07-NLP-RNNs-Representations/ (Дата обращения 10.05.2022).
8. Лыченко Н. М., Сороковая А. В. Применение LSTM-нейронных сетей для классификации индекса качества воздуха г. Бишкек //Проблемы автоматики и управления. – 2020. – №. 1.
9. Zhao X. et al. A Deep Recurrent Neural Network for Air Quality Classification //J. Inf. Hiding Multim. Signal Process. – 2018. – Т. 9. – №. 2. – С. 346-354.
10. Официальный сайт языка Pyhon [Электронный ресурс] URL: https://www.python.org/ (Дата обращения 10.03.2022)

11. Документация по библиотеке Pandas [Электронный ресурс] URL: https://pandas.pydata.org/docs/reference/frame.html (Дата обращения 12.03.2022)
12. Документация по библиотеке Scikit-learn [Электронный ресурс] URL: https://scikit-learn.org/stable/user_guide.html (Дата обращения 25.03.2022)
13. Документация по библиотеке matplotlib [Электронный ресурс] URL: https://matplotlib.org/stable/users/index (Дата обращения 12.03.2022)
14. Документация по фреймворку TensorFlow [Электронный ресурс] URL: https://www.tensorflow.org/api_docs/python/tf (Дата обращения 05.05.2022)
15. Документация по библиотеке Keras [Электронный ресурс] URL: https://keras.io/api/ (Дата обращения 05.05.2022)
16. Документация по библиотеке aiogram [Электронный ресурс] URL: https://docs.aiogram.dev/en/latest/ (Дата обращения 11.05.2022)
17. Hattingh C. Using Asyncio in Python: Understanding Python's Asynchronous Programming Features. – " O'Reilly Media, Inc.", 2020. – 166 с.
18. Документация по библиотеке SpeechRecognition [Электронный ресурс] URL: https://pypi.org/project/SpeechRecognition/ (Дата обращения 26.04.2022)
19. Документация по библиотеке Soundfile [Электронный ресурс] URL: https://pysoundfile.readthedocs.io/en/latest/ (Дата обращения 26.04.2022)
20. Документация по программе FFmpeg [Электронный ресурс] URL: https://ffmpeg.org/ffmpeg.html (Дата обращения 18.03.2022)
21. Документация по сервису Yandex.Cloud [Электронный ресурс] URL: https://cloud.yandex.ru/docs (Дата обращения 20.03.2022)
22. Уилсон Д., Редмонд Э. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL. – Litres, 2017. – 384 с.
23. Севастьянов Л. А., Щетинин Е. Ю. О методах повышения точности многоклассовой классификации на несбалансированных данных //Информатика и её применения. – 2020. – Т. 14. – №. 1. – С. 63-70.

ПРИЛОЖЕНИЕ 1

МОДУЛЬ ПРЕОБРАЗОВАНИЯ РЕЧИ ИЗ ЗАПИСЕЙ РАЗГОВОРОВ В ТЕКСТ

Код модуля:

```
#imports

def ogg_to_wav(file_name):
    import soundfile as sf
    data, samplerate = sf.read(file_name)
    sf.write(file_name.replace('ogg', 'wav'), data, samplerate)

df = pd.read_csv('dataset_processed.csv', sep=';')
df = df.where(pd.notnull(df), None)

def recognize_yandex():
    try:
        # Вставьте свой API-ключ
        key = ''
        # Вставьте свой путь к файлу в бакете. Всё, что в ссылке стоит после знака
вопроса, можно стереть — сервер всё равно это проигнорирует
        link_to_file = 'https://storage.yandexcloud.net/requests-bucket/2021/'
        url =
"https://transcribe.api.cloud.yandex.net/speech/stt/v2/longRunningRecognize"
        # Формируем заголовок запроса, в котором ссылаемся на API-ключ
        header_template = {'Authorization': 'Api-Key {}'.format(key)}

        for i, row in df.iterrows():
            # Формируем сам текст запроса
            body_template = {
                "config": {
                    "specification": {
                        "languageCode": "ru-RU"
                    }
                },
                "audio": {
                    "uri": link_to_file + df.at[i, 'call_dir']
                }
            }

            try:
                # Отправляем запрос
                req = requests.post(url, headers=header_template, json=body_template)
                # Получаем технический ответ от сервера и выводим его
                data = req.json()
                # Получаем идентификатор запроса
                req_id = data['id']
                # Ждем пока не будет произведено распознавание
                while True:
                    time.sleep(1)
                    # Пытаемся получить ответ по нашему идентификатору запроса
                    get_url = "https://operation.api.cloud.yandex.net/operations/{id}"
                    req = requests.get(get_url.format(id=req_id), headers=
header_template).json()
                    if req['done']:
                        break
                text = ''
```

```
                for ch in req['response']['chunks']:
                    text += ch['alternatives'][0]['text'] + '\n'
                df.at[i, 'full_text'] = text
            except Exception as e:
                print(e)
                df.to_csv('dataset_processed.csv', sep=';', index=False)
                break
        df.to_csv('dataset_processed.csv', sep=';', index=False)
        print('File updated!')
    except KeyboardInterrupt:
        df.to_csv('dataset_processed.csv', sep=';', index=False)
        print('File updated!')

def recognize_speechrecogintion():
    try:
        for i, row in df.iterrows():
            try:
                file_name = '/media/nikita/9d12adaa-22b4-450a-8da9-
c813881b3c42/home/nikita/copy/2021/'
                ogg_to_wav(file_name + df.at[i, 'call_dir'])
                sample_audio = sr.AudioFile(file_name + df.at[i,
'call_dir'].replace('ogg', 'wav'))
                r = sr.Recognizer()
                with sample_audio as audio_file:
                    audio_content = r.record(audio_file)
                try:
                    text = r.recognize_google(audio_content, language='ru-RU')
                except Exception as e:
                    print(e)
                    text = ''
                df.at[i, 'full_text'] = text
            except Exception as e:
                print(e)
                df.to_csv('dataset_processed.csv', sep=';', index=False)
                break
        df.to_csv('dataset_processed.csv', sep=';', index=False)
        print('File updated!')
    except KeyboardInterrupt:
        df.to_csv('dataset_processed.csv', sep=';', index=False)
        print('File updated!')


if __name__ == "__main__":
    # recognize_speechrecogintion()
    recognize_yandex()
```

ПРИЛОЖЕНИЕ 2

РЕАЛИЗАЦИЯ КЛАССИФИКАТОРОВ

Код моделей классификаторов:

```
#imports
group_new_types = {
                  'Благодарность': 'Благодарность',
                  ...
                  'Рекомендации по пересадкам': 'Справочная информация'
}

df = pd.read_csv('dataset_processed2.csv', delimiter=';')
df['datetime'] = pd.to_datetime(df['datetime'])
df['datetime'] = df.datetime.dt.tz_convert('Asia/Yekaterinburg')
df.drop(['full_name', 'phone'], axis=1, inplace=True)
df['new_type'] = df['type'].apply(lambda row: group_new_types[row])
df = df.dropna(subset=['full_text'])
df2 = pd.read_csv('dataset_processed.csv', delimiter=';')
df2['datetime'] = pd.to_datetime(df2['datetime'])
df2['datetime'] = df2.datetime.dt.tz_convert('Asia/Yekaterinburg')
df2.drop(['full_name', 'phone'], axis=1, inplace=True)
df2['new_type'] = df2['type'].apply(lambda row: group_new_types[row])
df2 = df2.dropna(subset=['full_text'])
df2 = df2.loc[df2['id'] > df.iloc[-1]['id']]
df = pd.concat([df, df2])
df = df.loc[~df['new_type'].isin(['Справочная информация', 'Парковочное
пространство'])]
print(df)

stopwords_russian = stop_words.get_stop_words('ru')
stopwords_russian.extend(['...', '«', '»', 'доброе утро', 'добрый день', 'добрый
вечер', 'вообще', 'елена', 'вера', 'слушаю', 'елена слушаю', 'вера слушаю',
'здравствуйте', 'здравствуй', 'до свидания'])


def clean_punctuation(text):
    return ''.join([ch if ch not in string.punctuation else ' ' for ch in text])


def clean_numbers(text):
    return ''.join([i if not i.isdigit() else ' ' for i in text])


def clean_multiple_spaces(text):
    return re.sub(r'\s+', ' ', text, flags=re.I)


def clean_stop_words(text):
    return ' '.join(word for word in text.split() if word not in stopwords_russian)


prep_text =
[clean_stop_words(clean_multiple_spaces(clean_numbers(clean_punctuation(text.lower()))))
) for text in
             df['full_text'].astype('str')]
df['text_prep'] = prep_text
```

```
raw_data = df.dropna(subset=['text_prep'])

snowball = SnowballStemmer(language="russian")
lemm_texts_list = []
for text in raw_data['text_prep']:
    text_lem = [snowball.stem(word) for word in text.split(' ')]
    if len(text_lem) <= 2:
        lemm_texts_list.append('')
        continue
    lemm_texts_list.append(' '.join(text_lem))
raw_data['text_lemm'] = lemm_texts_list
raw_data = raw_data[raw_data['text_lemm'] != '']

def nn__model():
    #imports

    tokenizer = Tokenizer()
    tokenizer.fit_on_texts(raw_data['text_lemm'].values)
    sequences = tokenizer.texts_to_sequences(raw_data['text_lemm'].values)
    word_index = tokenizer.word_index
    data = tokenizer.sequences_to_matrix(sequences, mode='tfidf')
    code = np.array(raw_data['new_type'])
    label_encoder = LabelEncoder()
    vec = label_encoder.fit_transform(code)
    labels = to_categorical(vec)

    X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.3,
random_state=42)

    model = Sequential()
    model.add(Dense(512, input_shape=(len(word_index) + 1,), activation='relu'))
    model.add(Dropout(0.6))
    model.add(Dense(labels.shape[1], activation='softmax'))
    model.summary()
    model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['acc'])
    history = model.fit(X_train, y_train, verbose=1, epochs=2, batch_size=128,
                        validation_split=0.2)

    accr = model.evaluate(X_test, y_test)
    print('Test set  Loss: {:0.3f}  Accuracy: {:0.3f}'.format(accr[0], accr[1]))

    y_pred = model.predict(X_test, batch_size=1)
    y_pred = np.argmax(y_pred, axis=1)
    y_test = np.argmax(y_test, axis=1)
    clf_report = classification_report(y_test,
                                       y_pred,
                                       output_dict=True)
    sns.heatmap(pd.DataFrame(clf_report).iloc[:-1, :].T, annot=True)
    plt.show()

    #imports
    MAX_NB_WORDS = 50000
    MAX_SEQUENCE_LENGTH = 250
    EMBEDDING_DIM = 100
    tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-
./:;<=>?@[\]^_`{|}~', lower=True)
    tokenizer.fit_on_texts(raw_data['text_lemm'].values)
    word_index = tokenizer.word_index
    X = tokenizer.texts_to_sequences(raw_data['text_lemm'].values)
```

```
    X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
    Y = pd.get_dummies(raw_data['new_type']).values

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.30,
random_state=42)

    model = Sequential()
    model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
    model.add(SpatialDropout1D(0.2))
    model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
    model.add(Dense(11, activation='softmax'))
    model.compile(loss='categorical_crossentropy', optimizer='adam',
metrics=['accuracy'])

    epochs = 10
    batch_size = 64

    model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,
validation_split=0.1,
                        callbacks=[EarlyStopping(monitor='val_loss', patience=3,
min_delta=0.0001)])

    accr = model.evaluate(X_test, Y_test)
    print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0], accr[1]))


def sgd_model():
    #imports

    # Приводим содержимое в нижний регистр
    appeal = list(raw_data['text_lemm'].values)
    appeal = [str(l).lower() for l in appeal]

    vectorizer = TfidfVectorizer()
    tfidfed = vectorizer.fit_transform(appeal)

    # Делим выборку на тренировочную и тестовую

    X = tfidfed
    label_encoder = LabelEncoder()
    code = np.array(raw_data['new_type'])
    vec = label_encoder.fit_transform(code)
    labels = to_categorical(vec, num_classes=11)
    y = vec
    X_train, X_test, y_train, y_test = train_test_split(X, y, train_size=0.7,
random_state=42)

    penalties = ['l2', 'l1', 'elasticnet']
    alphas = [0.00001, 0.0001, 0.001, 0.01, 0.1, 0.5]
    max_iters = [5, 10, 15, 20, 25, 35, 50]
    best_penalty = ''
    best_alpha = 0
    best_iter = 0
    best_score = 0
    for penalty in penalties:
        for alpha in alphas:
            for max_iter in max_iters:
```

```
                clf = linear_model.SGDClassifier(random_state=42, loss='hinge',
penalty=penalty, alpha=alpha,
                                                 max_iter=max_iter)
                # Обучаем модель
                clf.fit(X_train, y_train)
                acc_score = accuracy_score(y_test, clf.predict(X_test))
                if acc_score > best_score:
                    best_score = acc_score
                    best_penalty = penalty
                    best_alpha = alpha
                    best_iter = max_iter
    print(best_penalty)
    print(best_alpha)
    print(best_iter)
    clf = linear_model.SGDClassifier(random_state=42, loss='hinge',
penalty=best_penalty, alpha=best_alpha,
                                     max_iter=best_iter)
    clf.fit(X_train, y_train)
    print("Train accuracy = %.3f\n" % accuracy_score(y_train, clf.predict(X_train)))
    print("Test accuracy = %.3f\n" % accuracy_score(y_test, clf.predict(X_test)))
    from sklearn.metrics import f1_score
    print(f"F1 Score: {f1_score(y_test, clf.predict(X_test), average='weighted')}")


sgd_model()
nn__model()
```

ПРИЛОЖЕНИЕ 3

КОД TELEGRAM БОТА

Код кнопок меню Telegram бота:

```
from aiogram.types import ReplyKeyboardMarkup, KeyboardButton

btn_main = KeyboardButton('Главное меню')
# --- Main Menu ---
btn_request = KeyboardButton('Оставить новое обращение')
btn_times = KeyboardButton('Узнать расписание')
btn_info = KeyboardButton('О боте')
main_menu = ReplyKeyboardMarkup(resize_keyboard=True).add(btn_request, btn_times,
btn_info)

# --- Skip Menu ---
btn_skip = KeyboardButton('Пропустить')
skip_menu = ReplyKeyboardMarkup(resize_keyboard=True).add(btn_skip, btn_main)

# --- Skip Menu ---
exit_menu = ReplyKeyboardMarkup(resize_keyboard=True).add(btn_main)
```

Код Telegram бота:

```
#imports

TOKEN = '*'
bot = Bot(token=TOKEN)
dp = Dispatcher(bot, storage=MemoryStorage())


def audio_to_text(file_name):
    import speech_recognition as sr
    sample_audio = sr.AudioFile(file_name)
    r = sr.Recognizer()
    with sample_audio as audio_file:
        audio_content = r.record(audio_file)
    try:
        text = r.recognize_google(audio_content, language='ru-RU')
    except:
        text = ''
    return text


def ogg_to_wav(file_name):
    import soundfile as sf
    data, samplerate = sf.read(file_name)
    sf.write(file_name.replace('ogg', 'wav'), data, samplerate)


async def handle_file(file: File, file_name: str, path: str):
    Path(f"{path}").mkdir(parents=True, exist_ok=True)

    await bot.download_file(file_path=file.file_path,
destination=f"{path}/{file_name}")
    ogg_to_wav(f"{path}/{file_name}")
    text = audio_to_text(f"{path}/{file_name}")".replace('ogg', 'wav'))
```

```
@dp.message_handler(content_types=[ContentType.VOICE])
async def voice_message_handler(message: Message):
    state = dp.current_state(user=message.from_user.id)
    user_data = await state.get_data()
    state_id = user_data.get('state_id', 0)
    if state_id != 8:
        await bot.send_message(message.from_user.id,
                               'Оставить голосовое сообщение возможно только при
заполнении текста обращения.')
        return
    voice = await message.voice.get_file()
    path = "/home/nikita/voices"

    await handle_file(file=voice, file_name=f"{voice.file_id}.ogg", path=path)
    await bot_message(message)


@dp.message_handler(commands=['start'])
async def command_start(message: types.Message):
    await bot.send_message(message.from_user.id, 'Привет, {0.first_name}
{0.last_name}'.format(message.from_user),
                           reply_markup=nav.main_menu)


@dp.message_handler()
async def bot_message(message: types.Message):
    state = dp.current_state(user=message.from_user.id)
    user_data = await state.get_data()
    state_id = user_data.get('state_id', 0)

    if message.text == 'Оставить новое обращение':
        await bot.send_message(message.from_user.id, 'Введите маршрут происшествия: ',
reply_markup=nav.exit_menu)
        await state.update_data(state_id=1)
    elif message.text == 'Главное меню':
        await bot.send_message(message.from_user.id, 'Главное меню',
reply_markup=nav.main_menu)
        await state.update_data(state_id=0)
    elif message.text == 'Пропустить':
        if state_id == 2:
            await bot.send_message(message.from_user.id, 'Введите дату происшествия: ',
reply_markup=nav.exit_menu)
        elif state_id == 5:
            await bot.send_message(message.from_user.id, 'Введите ваше ФИО: ',
reply_markup=nav.exit_menu)
        else:
            await bot.send_message(message.from_user.id, 'Невозможно пропустить!',
reply_markup=nav.main_menu)
        await state.update_data(state_id=state_id + 1)
    elif message.text == 'О боте':
        await state.update_data(state_id=0)
        await bot.send_message(message.from_user.id, 'Данный бот предназначен для
получения обращений о работе'
                                                     ' общественного транспорта г.
Тюмени.\n'
                                                     'Сайт <a
href="https://tgt72.ru/">Тюменьгортранс</a>',
```

```
                                parse_mode=types.ParseMode.HTML)
    elif message.text == 'Узнать расписание':
        await state.update_data(state_id=10)
        await bot.send_message(message.from_user.id, 'Вы всегда можете узнать более
подробную информацию о расписании'
                                                     ' движения общественного
транспорта на сайте'
                                                     ' <a href="https://tgt72.ru/">
Тюменьгортранс</a> или в приложении'
                                                     ' "Тюменский транспорт"',
                                reply_markup=nav.exit_menu,
parse_mode=types.ParseMode.HTML)
        await bot.send_message(message.from_user.id, 'Введите интересующий вас маршрут:
', reply_markup=nav.exit_menu)
    else:
        if state_id == 1:
            route = message.text
            await bot.send_message(message.from_user.id, 'Введите остановку
происшествия: ', reply_markup=nav.skip_menu)
            await state.update_data(state_id=2)
        elif state_id == 2:
            checkpoint = message.text
            await bot.send_message(message.from_user.id, 'Введите дату происшествия
(дд.мм.гггг): ', reply_markup=nav.exit_menu)
            await state.update_data(state_id=3)
        elif state_id == 3:
            dt = message.text
            try:
                dt = datetime.strptime(dt, "%d.%m.%Y")
            except ValueError:
                dt = None
                await bot.send_message(message.from_user.id, 'Введите дату происшествия
(дд.мм.гггг): ',
                                       reply_markup=nav.exit_menu)
                return
            await bot.send_message(message.from_user.id, 'Введите время происшествия
(чч:мм): ')
            await state.update_data(state_id=4)
        elif state_id == 4:
            tm = message.text
            try:
                tm = datetime.strptime(tm, "%H:%M")
            except ValueError:
                tm = None
                await bot.send_message(message.from_user.id, 'Введите время
происшествия (чч:мм): ',
                                       reply_markup=nav.exit_menu)
                return
            await bot.send_message(message.from_user.id, 'Введите госномер ТС: ',
reply_markup=nav.skip_menu)
            await state.update_data(state_id=5)
        elif state_id == 5:
            gosnumber = message.text
            await bot.send_message(message.from_user.id, 'Введите ваше ФИО: ',
reply_markup=nav.exit_menu)
            await state.update_data(state_id=6)
        elif state_id == 6:
            fio = message.text
            await bot.send_message(message.from_user.id, 'Введите ваш номер телефона
(+7xxxxxxxxxx): ',
                                   reply_markup=nav.exit_menu)
```

```
            await state.update_data(state_id=7)
        elif state_id == 7:
            number = message.text
            result = re.match(r'^(\+7|7|8)?[0-9]{10}$', number)
            if not result:
                await bot.send_message(message.from_user.id, 'Введите ваш номер
телефона (+7xxxxxxxxxx): ',
                                       reply_markup=nav.exit_menu)
                return
            await bot.send_message(message.from_user.id, 'Введите текст обращения: ',
reply_markup=nav.exit_menu)
            await state.update_data(state_id=8)
        elif state_id == 8:
            text = message.text
            await bot.send_message(message.from_user.id, 'Спасибо за ваше обращение!
Оно будет обработано специалистами'
                                                         ' в ближайшее время.',
reply_markup=nav.main_menu)
            await state.update_data(state_id=0)
        elif state_id == 10:
            route_number = message.text
            route_id = None
            url = 'https://api.tgt72.ru/api/v5/routesforsearch/'
            response = requests.get(url)
            for route in response.json()['objects']:
                if route['name'] == route_number:
                    route_id = route['id']
            if route_id:
                url = 'https://api.tgt72.ru/api/v5/checkpointsforsearch/?route_id=%s' %
route_id
                response = requests.get(url)
                checkpoints_response = response.json()
                keyboard = ReplyKeyboardMarkup(resize_keyboard=True)
                cp_dict = {}
                for checkpoint in checkpoints_response['objects']:
                    button = KeyboardButton(text=checkpoint['name'] + ' (' +
checkpoint['description'] + ')')
                    keyboard.add(button)
                    cp_dict[checkpoint['name'] + ' (' + checkpoint['description'] +
')'] = checkpoint['id']
                await bot.send_message(message.from_user.id, 'Выберите остановку из
списка:',
                                       reply_markup=keyboard)
                await state.update_data(state_id=11)
                await state.update_data(cp_dict=cp_dict)
                await state.update_data(route_id=route_id)
            else:
                await bot.send_message(message.from_user.id, 'Вы ввели несуществующий
номер маршрута! ',
                                       reply_markup=nav.main_menu)
                await state.update_data(state_id=0)
        elif state_id == 11:
            checkpoint_name = message.text
            cp_dict = user_data.get('cp_dict', {})
            route_id = user_data.get('route_id', 0)
            url =
'https://api.tgt72.ru/api/v5/times/?date=%s&route_id=%s&checkpoint_id=%s' %
(current_date(), route_id, cp_dict[checkpoint_name])
            response = requests.get(url)
            response_json = response.json()
```

```
            list_times = response_json['objects'][0]['times'] +
response_json['objects'][0]['very_big_car_times'] + \
                         response_json['objects'][0]['physically_challenged_support']
            list_times = sorted(list(set(list_times)))
            current_time = datetime.now().strftime("%H:%M")
            top10_times = []
            text_reply = 'Расписание ближайших 10 автобусов: '
            for time in list_times:
                if time > current_time:
                    top10_times.append(time)
                if len(top10_times) == 10:
                    break
            text_reply += ', '.join(top10_times)
            await bot.send_message(message.from_user.id, text_reply,
reply_markup=nav.main_menu)
            await state.update_data(state_id=0)
        else:
            await message.reply('Неизвестная команда')

if __name__ == '__main__':
    executor.start_polling(dp, skip_updates=True)
```