СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Глобальный рынок маркетплейсов работающих на фрилансе специалистов показывает стабильное финансовое развитие на протяжении нескольких лет. Так, в 2018 году объем рынка составил 2,35 млрд долл., в 2020 году – 3,81 млрд долл., а к 2025 году ожидается его рост до 6,7 млрд долларов. Более того, по прогнозам специалистов, к 2030 году 80% всех частных мастеров перейдут на маркетплейсы.

Тенденции мирового рынка применимы и к отечественному: по оценкам Ассоциации предпринимателей индустрии красоты, объем российского рынка в 2019 году составил 972 млрд рублей, а за следующие несколько лет увеличился еще на 145,6% (I кв. 2022 г. к I кв. 2020 г.), при этом среднегодовые темпы роста спроса сохранялись на уровне 60%.

“Паттерн” искать мастера на онлайн-платформах у россиян сформировался в 2020 г. с началом пандемии и введением повсеместных ограничений. В период пандемии индустрия красоты стала децентрализованным рынком, на котором работают сотни тысяч частных мастеров. Повсеместное закрытие салонов красоты побудило потребителей активно искать мастеров в Интернете.

Тем не менее, на рынке маркетплейсов специалистов продолжает существовать ряд проблем. Одной из них является проблема поиска и подбора мастера клиентами. На существующих ныне маркетплейсах специалистов (Авито, Профи.ру и пр.) клиент вынужден осуществлять выбор мастера вручную. При этом он может изучить лишь публичный профиль мастера, который нередко содержит недостоверную информацию, и потратить значительное количество времени на сравнение профилей мастеров вручную. Таким образом, механический выбор является неэффективным и временно затратным.

Другая проблема возникает у мастеров и касается сложности привлечения клиентов. На сегодняшний день частный мастер, желающий самостоятельно осуществить продвижение своих услуг на маркетплейсах, вынужден проходить самостоятельное обучение основам маркетинга для построения и поддержки личного бренда, заказа рекламы и пр. Эти действия требуют траты значительных дополнительных ресурсов (в том числе финансовых), что в конечном счете отвлекает мастеров от поддержки высокого качества оказываемых услуг и/или приводит к повышению цен на услуги (либо снижению дохода мастера).

Две следующие проблемы связаны с планированием времени и закупки материалов для работы. Частные мастера должны самостоятельно планировать свое

расписание, то есть тратить значительное количество времени на коммуникацию с клиентами и ведение календаря, а также контролировать выбор, покупку и поддержание расходных мастеров, то есть закладывать дополнительное рабочее время для поиска поставщиков, сравнения и выбора этих материалов. Обозначенные выше временные промежутки было бы более продуктивнее тратить на развитие своих рабочих навыков и/или обслуживание клиентов. Кроме того, в рамках этой проблемы не стоит забывать и о простоях между заказами, время которых для мастера является не оплачиваемым. Помимо этого, мастер может выбрать материалы с неоптимальным соотношением цены и качества, что может увеличить себестоимость его услуг и привести к повышению цен на услуги (либо снизить доход мастера).

Настоящий НИОКР нацелен на автоматизацию организационных процессов деятельности специалистов бьюти-индустрии, работающих через Маркетплейс “Мой Профи”, и на обеспечение взаимовыгодного сотрудничества мастеров и пользователей, становящихся их клиентами.

Задача разрабатываемой новой версии Платформы (“Мой Профи 2.0”) – решить названные выше проблемы за счет внедрения в маркетплейс специалистов модулей автоматизации, созданных с применением методом машинного обучения и искусственного интеллекта. Задача новых разрабатываемых модулей Маркетплейса - автоматизировать:

●процесс точного подбора мастера под заказ клиента, при этом должны максимально учитываться пожелания и ожидания клиента;

●процесс формирования “бесшовной” (с минимальными простоями) ленты заказов для мастера на неделю и более длительные периоды;

●процесс управления запасами и оптимизации стоимости расходных материалов (в т.ч. косметики) для мастера.

Платформа “Мой Профи” (далее – Платформа) является классической цифровой платформой (информационный посредник), позволяющей объединить клиентов и специалистов в области индустрии красоты, а также повысить эффективность их взаимодействия. Маркетплейс включает в себя два мобильных приложения на платформах iOS и Android: “Мой Профи. КЛИЕНТ” и “Мой Профи. МАСТЕР”, которые являются интерфейсами для доступа к Маркетплейсу соответствующих категорий пользователей.

Текущая версия Платформы уже обеспечивает автоматизацию следующих основных процессов:

1. формирование портфолио мастера на Платформе с информацией о выполненных заказах, оценками и отзывами клиентов;
2. подбор клиентом мастера по параметрам;
3. размещение предложений от мастера;
4. запись клиента к мастеру и система напоминаний клиенту о сеансах;
5. безопасная онлайн-оплата за услугу с замораживанием Платформой стоимости заказа до момента подтверждения клиентом его выполнения мастером (“безопасная сделка”);
6. ведение и визуализация истории заказов;
7. анализ доходов и расходов мастера по различным параметрам;
8. система оценок и отзывов как мастеров, так и клиентов;
9. программа лояльности для клиентов.

Разрабатываемая в ходе проекта новая версия Платформы (“Мой Профи 2.0”) состоит из frontend-слоя (веб-портала с web-интерфейсами для различных категорий пользователей: мастер, клиент, администратор) и backend-слоя – совокупности микросервисов с модулями, которые в настоящее время являются технически уникальными решениями, их аналоги отсутствуют у конкурентов проекта на рынке:

1) модуль автоматического подбора исполнителя и управления его расписанием. Платформа формирует для мастеров ленту наиболее подходящих заказов, точно учитывая широкий набор факторов: данные об опыте, историю просмотров и выполненных заказов, их ценовую категорию, географическое расположение, соответствие предпочтениям клиента (“мэтчинг”) и другие. Таким образом, мастер может “откликнуться” только на заказ клиента, которому он с высокой вероятностью подойдет. Заказы клиентов, которым мастер не подходит, просто не отображаются в ленте мастера. При этом Платформа автоматически отбирает для отображения в ленте мастера заказы с учетом свободных у него слотов времени, чтобы сформировать бесшовное расписание исполнителя на неделю и более длительные периоды. Учитываются ожидаемое время выполнения заказа, время в пути, предпочтительные для работы часы и дни. Цель – обеспечить высокую удовлетворенность клиентов, максимальное использование рабочего времени мастера, финансовую достаточность для мастера работы только на Платформе, без сторонних подработок;

2) модуль автоматического учета, анализа и пополнения запасов. Алгоритм на базе искусственного интеллекта анализирует темпы расходования запасов материалов и планирует закупки продукции. При этом алгоритм предлагает мастеру заменить используемые материалы на более дешевые аналоги или материалы, доступные в

данный момент со скидкой. При этом предлагаемый материал должен иметь аналогичную заменяемому или более высокую оценку качества от других мастеров. Мастер может согласиться или отказаться от замены. С помощью интеграций с интернет-магазинами партнеров закупка осуществляется в один клик – мастеру нужно лишь подтвердить или скорректировать заказ, сформированный алгоритмом. При этом, поскольку мастера закупают косметические средства и расходные материалы через Платформу, закупка осуществляется со значительной скидкой, т.к. для в отношениях с интернет-магазином Платформа выступает как единый оптовый покупатель.

В рамках первого этапа НИОКР были разработаны следующие модели: модель фильтрации текстового запроса (TF-IDF + BOW), модель фильтрации текстового запроса на базе GBM. Помимо этого, был разработан и протестирован алгоритм матричной факторизации применимо к ранжированию пользователей.

В рамках второго этапа НИОКР был протестирован функционал фильтрации тестового запроса; собран, протестирован и доработан по результатам тестирования модуль автоматического подбора исполнителя и управления его расписанием.

В рамках третьего этапа НИОКР был разработан и протестирован алгоритм анализа расходов и планирования закупок расходных материалов, подключена интеграция с интернет-магазинами, собран модуль автоматического учета, анализа и пополнения запасов. Также было проведено испытание маркетплейса и его доработка по результатам испытаний, разработана проектная документация.

В ходе разработок и исследований производилась комплексная работа, в рамках которой дорабатывались и проходили тестирование компоненты серверной части (разрабатывались новые микросервисы), а также web-страницы и компоненты Frontend-приложения.

Стек применяемых технологий включает ОС Linux, языки программирования Python, Swift, Kotlin, TypeScript, JavaScript, фреймворки React.js, React Native, Django Rest Framework, web-сервер Nginx, СУБД PostgreSQL, Elasticsearch, Redis, брокер очередей RabbitMQ, библиотеки Scikit Learn, CatBoost, Pytorch, Numpy, Pandas, OpenAPI, JSON.

По итогам выполнения НИОКР (по результатам внедрения новых модулей Платформы) ее пользователи смогут пользоваться нижеследующим автоматизированным функционалом:

1) процесс точного подбора мастера под заказ клиента (при этом будут максимально учитываться пожелания и ожидания клиента);

2) процесс формирования “бесшовной” (с минимальными простоями) ленты заказов для мастера на неделю и более длительные периоды;

3) процесс управления запасами и оптимизации стоимости расходных материалов (в т.ч. косметики) для мастера.

Далее описываются этапы разработок, испытаний и доработок модулей Платформы.

ОСНОВНАЯ ЧАСТЬ

1. Этап 1

1.1. Разработка модели фильтрации текстового запроса (TF-IDF + BOW)

В ходе данного этапа КП была осуществлена разработка модели фильтрации текстового запроса. Ключевой задачей данной модели является интеллектуальный анализ поискового запроса клиента с целью подбора оптимального мастера.

Целевая модель была реализована на основе технологий TF-IDF и BOW.

TF-IDF (Term Frequency-Inverse Document Frequency) – это статистическая модель, используемая для более точной фильтрации текстового запроса, чтобы найти наиболее релевантные документы.

BOW (bag of words) – это метод представления текста в виде вектора, где каждое слово в словаре соответствует отдельной координате вектора. Значение координаты равно количеству раз, которое данное слово встречается в тексте.

Для подготовки обучающего и валидационного датасетов использовались накопленные исторические данные в базе данных Платформы. Разметка датасета происходила в ручном режиме.

При препроцессинге данных перед разметкой при анализе поискового запроса из него исключаются стоп-слова (например, знаки препинания, служебные слова). Все запросы путем выгрузки из базы данных сортируются по категориям (то есть типам услуг), после чего по каждой из категорий производится отбор наиболее полных запросов с точки зрения описания. Таким образом было сформировано облако наиболее релевантных для обучения запросов для каждого типа услуги.

Далее в ручном режиме сотрудниками производилось сопоставление каждого запроса с рекомендуемым мастером посредством анализа его параметров описания и отзывов из базы мастеров Платформы “Мой Профи”.

Размер подготовленной размеченной выборки составил 20 000 экземпляров.

При препроцессинге данных для поискового запроса клиента используется полнотекстовый поиск Elasticsearch по истории заказов, описаний и отзывов. Затем модель делает регрессию по входному вектору данных и выбирает мастера со схожей специализацией с максимальным коэффициентом соответствия запросу клиента.

Для целей осуществления разработки модели фильтрации текстового запроса нами были использованы библиотеки Scikit Learn, Numpy и Pandas для языка

программирования Python 3.8.

Разработанная модель на выходе определяет процент семантического соответствия текстового запроса клиента той информации, которая была внесена у мастера в профиль, перечень услуг, отзывы. В качестве граничного значения, при котором считается, что мастер подходит для данного поискового запроса, выбран порог в 75%.

В результате тестирования модели на тестовой выборке из 5000 экземпляров процент верных сопоставлений запроса клиента с мастером, при которых процент семантического соответствия текстового запроса клиента информации, внесенной у мастера выше 75%, составил 81.3%. По результатам тестирования также установлено среднее время обработки запроса, которое составляет 0.35 секунды. В результате переобучения качество работы выросло до 86.8%.

Для работы модели был создан микросервис на базе Python, который запускается в автономном режиме как слушатель очереди RabbitMQ в виде Docker-контейнера. При получении запроса от API Платформы микросервис собирает данные о клиенте и мастерах из PostgreSQL, формирует входные данные и вызывает функцию предсказания модели. Затем отправляет результат в очередь RabbitMQ. Архитектура Платформы с интегрированным микросервисом изображена на Рис. 1.

По результатам тестирования микросервиса проводилась оптимизация структур данных и исходного кода для повышения производительности (уменьшения времени отклика). С учетом среднего времени обработки запроса моделью среднее суммарное время отклика удалось уменьшить до 0.37 секунд.

На последнем этапе микросервис интегрировался в продакшн-решение и развертывался на продуктивном стенде Платформы.

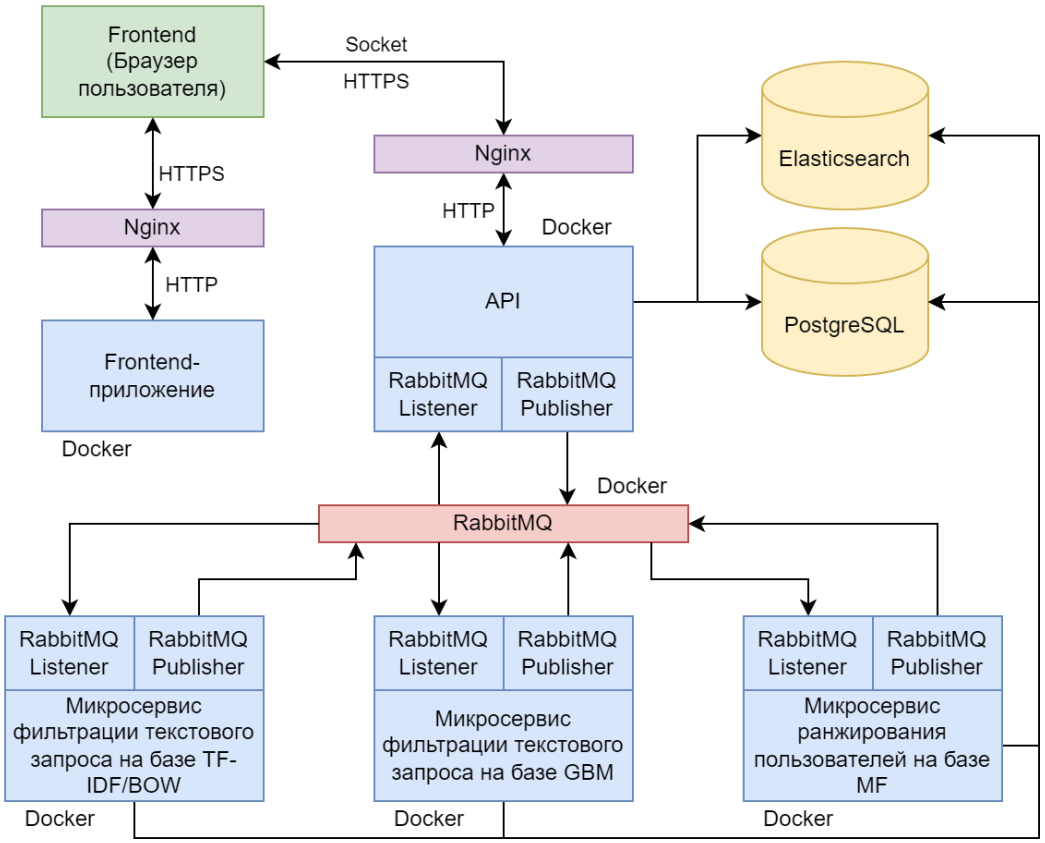


Рисунок 1 – Общая архитектура модулей в составе сервисов Платформы

1.2. Разработка модели фильтрации текстового запроса на базе GBM

Целью данного мероприятия являлось создание модели фильтрации текстового запроса, которая будет сравнивать параметры мастера и клиента, чтобы определить степень их близости. Это также необходимо для выбора наилучшего варианта из списка мастеров, формируемом в Модуле 1 (двухступенчатая фильтрация). Для этих целей была внедрена модель на базе технологии GBM.

Для решения нашей задачи используется библиотека CatBoost для языка Python. Подготовка разметки обучающего и валидационного датасета производились вручную специалистами-разметчиками. Целью разметки являлся выбор профилей мастеров, которых выбирали клиенты с успешным завершение сеанса (заказа) с высокими оценками или положительными отзывами. Для модели были составлены наборы признаков мастера и клиента. Далее было произведено обучение на исторических данных, чтобы определить степень близости конкретного мастера или клиента. Критерием успешности были успешно завершенный сеанс и положительный отзыв клиента.

Для формирования обучающей выборки с целью последующего обучения модели на базе градиентного бустинга производится программный отбор успешных сеансов (из базы данных Платформы), для которых клиент выбрал мастера и в результате остался доволен услугой / сеансом. Наборы данных формируются на основе накопленной базы маркетплейса “Мой Профи”.

Далее для каждого кейса была произведена выборка из базы данных параметров мастера и клиента. Все такие пары наборов вручную помечаются позитивной меткой, используя Excel-файл, который далее легко может быть обработан инструментами библиотеки Pandas. Для формирования набора для негативного класса выбираются случаи с отрицательными отзывами и досрочно завершенными сеансами по разным причинам.

Размер сформированной выборки составляет по 15 000 экземпляров для положительного и негативного классов.

Для прогнозирования совместимости разработана модель на базе библиотеки CatBoost на языке Python. Входные данные: на вход модели подаются числовые вектора параметров (все параметры в БД представлены в числовом виде) мастера и клиента. Выходные данные: на выходе модель определяет степень совместимости в диапазоне от 0 до 100%. Исходя из полученной степени происходит последующий мэтчинг.

Для обработки категориальных данных в модели CatBoost мы не используем one-hot кодирование, а просто задаем соответствующие параметры с помощью параметра cat_features. Это позволяет увеличить скорость обучения и улучшить качество прогнозов. CatBoost в своей реализации использует стандарт Scikit Learn. Код алгоритма реализован на языке Python с применением библиотек Pandas, Numpy.

По результатам тестирования средняя абсолютная ошибка составила 12.34%. Общее качество работы модели составило 89.2%. Среднее время отклика (время обработки запроса моделью) составило 0.05 секунды.

Микросервис модели работает в автономном режиме, в частности он является слушателем очереди RabbitMQ. Когда от API Платформы приходит сообщение о необходимости выбрать из списка мастеров наиболее оптимального, микросервис по идентификатору клиента собирает данные о нем, а также данные о профилях всех мастерах в списке для последующего формирования входных данных и вызова функции предсказания алгоритма модели. После предсказания микросервис отправляет ответное сообщение с результатом в очередь RabbitMQ. API Платформы далее может выдать результат пользователю в web-интерфейс посредством socket-соединения. Архитектура Платформы с интегрированным микросервисом изображена на Рис. 1.

Для внедрения микросервиса в систему на тестовом стенде были разработаны файлы: Dockerfile, Docker-compose.yaml и ci-cd.yaml (для настройки деплоя в GitLab CI/CD). Также в настройках CI/CD указываются переменные окружения, аутентификационные данные и прочие настройки, используемые микросервисом. Далее из панели управления GitLab CI осуществляется вызов команды на сборку и деплой микросервиса на указанную виртуальную машину или сервер. CI/CD-runner на целевой машине осуществляет установку необходимых библиотек и пакетов, скачивает исходный код и запускает процесс сборки и запуска целевого приложения. Запущенный в docker-контейнере сервис готов к работе.

В качестве главной метрики оценки качества модели была применена средняя абсолютная ошибка (MAPE), которая отражает степень вероятности для каждой пары мастера и клиента, что их сеанс успешно завершится (то есть не будет отменен) и клиент будет удовлетворен услугой и оставит положительный отзыв мастеру. Метрика измеряется в процентах от 0 до 100. Вычисление метрики производится алгоритмом CatBoost по формуле (1).

$$\frac{\sum_{i=1}^{N} w_i \frac{|a_i - t_i|}{Max(1, |t_i|)}}{\sum_{i=1}^{N} w_i} \quad (1),$$

где N – размер тестовой выборки,

wi – вес, который отражает вклад экземпляра в качество всей системы,

ai – истинное значение величины,

ti – прогнозное значение.

По результатам тестирования средняя абсолютная ошибка составила 12.34%.

Далее модель была интегрирована в разработанный под нее микросервис и проверена на тестовом и продакшен-стендах.

1.3. Разработка алгоритма матричной факторизации применимо к ранжированию пользователей

Целью данного исследования являлось исследование применения алгоритма матричной факторизации по отношению к ранжированию пользователей. Результатом стала разработка алгоритма матричной факторизации (MF) применимо к ранжированию пользователей.

В ходе исследований было установлено, что алгоритмы на базе подхода рекомендательных систем, в частности - матричной факторизации, являются наиболее сбалансированным по эффективности и сложности реализации.

В нашем случае данный алгоритм на основе подхода рекомендательных систем с реализацией в виде матричной факторизацией предназначается для определения рекомендуемых релевантных материалов и услуг пользователю Платформы посредством ранжирования пользователей с похожими профилями, историей запросов, услугами и сеансами. В широком смысле системы рекомендаций основаны на одной из двух стратегий.

Первая стратегия - это контентная фильтрация, которая основана на анализе содержимого, уже просмотренного пользователем. Например, система может рекомендовать фильмы, которые похожи на те, которые пользователь уже смотрел и оценил положительно.

Элементно-ориентированный подход (или компонентный подход) - это методология разработки, которая основывается на создании модульных компонентов, которые могут быть повторно использованы в различных приложениях. Компоненты обычно создаются с использованием объектно-ориентированного программирования и могут включать в себя как код, так и ресурсы, такие как изображения, шрифты и файлы конфигурации.

В своей базовой форме матричная факторизация характеризует как элементы, так и пользователей по векторам факторов, выведенных из рейтинговых моделей элементов. Высокое соответствие между элементным и пользовательским факторами приводит к рекомендации. Эти методы стали популярными в последние годы благодаря сочетанию хорошей масштабируемости и точности прогнозирования.

Кроме того, они предлагают большую гибкость для моделирования различных реальных ситуаций. Рекомендующие системы полагаются на различные типы входных данных, которые часто помещаются в матрицу с одним измерением,

представляющим пользователей, и другим измерением, представляющим элементы, представляющие интерес.

Наиболее удобными данными являются высококачественные явные обратные связи, которые включают в себя явный ввод данных пользователями относительно их интереса к продуктам.

Одним из достоинств матричной факторизации является то, что она позволяет включать дополнительную информацию. Когда явная обратная связь недоступна, системы референтов могут делать выводы о предпочтениях пользователя, используя неявную обратную связь, которая косвенно отражает мнение, наблюдая за поведением пользователя.

Таким образом, по результатам исследований была выбрана технология матричной факторизации (MF).

Обучающая выборка была взята из базы данных “Мой Профи”, которая содержит накопленные за два года сведения из истории успешно завершенных сеансов мастеров и клиентов. Размер выборки для обучения составил 25 тысяч сеансов и 10 тысяч пользователей. Разметка для выборки не требуется исходя из особенностей алгоритма MF.

Матричные факторизационные модели отображают как пользователей, так и элементы в единое латентное факторное пространство размерности f таким образом, что взаимодействия между пользователем и элементами моделируются как внутренние продукты в этом пространстве.

Соответственно, каждый элемент i ассоциируется с вектором $q_i \in R^f$ и каждый пользователь u связан с вектором $p_u \in R^f$. Для данного пункта i, элементы q_i измеряют степень, в которой данный пункт обладает этими факторами, положительными или отрицательными. Для данного пользователя u, элементы p_u измеряют степень интереса, который пользователь имеет к элементам, высоким по соответствующим факторам, опять же, положительным или отрицательным.

Получившийся точечный продукт, $q^T_i p_u$ фиксирует взаимодействие между пользователем u и объектом i - общий интерес пользователя к характеристикам объекта. Это приблизительно соответствует рейтингу пользователя u пункта i, который обозначается r_{ui}, что приводит к оценке:

$$\hat{r}_{ui} = q^T_i p_u.$$

Главная задача заключается в вычислении соотношения между каждым элементом и пользователем с учетом векторов q_i, $p_u \in R^f$. После того, как рекомендательная система выполнит это отображение, она может легко оценить рейтинг, который пользователь присвоит любому элементу с помощью уравнения выше.

Такая модель тесно связана с сингулярной величиной декомпозиции (SVD), хорошо зарекомендовавшей себя методикой выявления скрытых семантических факторов при поиске информации.

Применение SVD в области совместной фильтрации требует факторинга матрицы рейтинга пользовательского элемента. Это часто вызывает трудности из-за высокой доли пропущенных значений, вызванной редкостью в матрице рейтингов пользовательского элемента. Обычное SVD не определено, когда знания о матрице неполные. Более того, неосторожное обращение только к относительно небольшому количеству известных элементов очень склонно к переподгонке.

Ранние системы полагались на приписывание, чтобы заполнить недостающие рейтинги и сделать матрицу рейтинга плотной. Однако приписывание может быть очень дорогостоящим, так как оно значительно увеличивает объем данных. Кроме того, неточные вменения могут значительно искажать данные. Поэтому в более поздних вариантах алгоритма предлагалось моделировать только непосредственно наблюдаемые рейтинги, избегая при этом переполнения через упорядоченную модель. Для изучения факторных векторов (p_u и q_i) система минимизирует ошибку регуляризованного квадрата на множестве известных рейтингов:

$$\min_{q \cdot p \cdot} \sum_{(u,i) \in \mathbb{K}} \left(r_{ui} - q_i^T p_u\right)^2 + \lambda \left(\|q_i\|^2 + \|p_u\|^2\right) \quad (2),$$

где K - это набор пар (u, i), по которым известен r_{ui} (тренировочный набор). Система знакомится с моделью, подгоняя ее под ранее наблюдавшиеся рейтинги. Однако цель состоит в том, чтобы обобщить эти предыдущие рейтинги таким образом, чтобы предсказать будущие, неизвестные рейтинги.

Таким образом, система должна избегать переподгонки наблюдаемых данных путем регуляризации выученных параметров, величины которых оштрафованы. Константа λ контролирует степень регуляризации и обычно определяется перекрестной проверкой.

Алгоритм реализуется с помощью Python-библиотек Numpy, Scikit Learn и Pandas. Все три библиотеки хорошо подходят для использования в алгоритмах ALS благодаря их способности эффективно хранить и обрабатывать большие наборы

данных, а также их поддержке различных алгоритмов машинного обучения, таких как классификация, регрессия, кластеризация и т. д. Numpy особенно полезен в ALS из-за его способности быстро и эффективно выполнять операции линейной алгебры с большими наборами данных, в то время как Scikit Learn можно использовать для оценки различных моделей в одном и том же наборе данных с использованием методов перекрестной проверки или выбора модели.

Pandas можно использовать для быстрого суммирования больших наборов данных в более мелкие подмножества данных с помощью функции группировки или визуализации результатов с помощью встроенных возможностей построения графиков.

Alternating Least Square (ALS) является алгоритмом матричной факторизации и работает параллельно. ALS может быть реализован как в Apache Spark ML, так и с помощью инструментов Scikit Learn и создан для масштабных задач совместной фильтрации. ALS хорошо справляется с решением задач масштабируемости и разреженности данных, и это просто и хорошо масштабируется для очень больших наборов данных. Особенности, заключающиеся в ALS:

- ALS использует регуляризацию L2.
- ALS минимизирует две функции потерь поочередно. Сначала он фиксирует матрицу пользователя и запускает градиентный спуск с матрицей элементов. Затем он фиксирует матрицу элементов и запускает градиентный спуск с пользовательской матрицей.
- ALS запускает градиентный спуск параллельно через несколько разделов базовых данных обучения из кластера машин.

Как и другие алгоритмы машинного обучения, ALS имеет собственный набор гиперпараметров. Наиболее важные гиперпараметры в Alternating Least Square (ALS):

- maxIter: максимальное количество итераций для запуска (по умолчанию 10).
- ранг: количество скрытых факторов в модели (по умолчанию 10).
- regParam: параметр регуляризации в ALS (по умолчанию 1.0).

2. Проверка модели на тестовых данных.

Общее количество пользователей в примененной тестовой выборке составило 5 тысяч. В результате, средневзвешенная ошибка по результатам тестирования составила 10.73%. Качество работы модели составило 87.4%. Среднее время отклика модели матричной факторизации составило 0.12 секунд.

3. Реализация микросервиса модели на языке Python для работы в автономном режиме в составе Платформы, интеграция модели в микросервис.

После завершения работы над моделью производится реализация микросервиса, который работает в автономном режиме, в частности он является слушателем очереди RabbitMQ.

Когда от API Платформы приходит сообщение о необходимости выдать ранжированные рекомендации услуг, микросервис по идентификатору клиента собирает данные о нем из PostgreSQL, а также данные о всех сеансах за год для последующего формирования входных данных и вызова функции предсказания алгоритма модели. После предсказания микросервис отправляет ответное сообщение с результатом в очередь RabbitMQ. API Платформы далее может выдать результат пользователю в web-интерфейс посредством socket-соединения.

Архитектура Платформы с интегрированным микросервисом изображена на рис. 5.

4. Внедрение микросервиса на тестовом стенде в систему и тестирование работы Платформы с учетом реализации API-вызовов из Frontend-приложения.

Для внедрения микросервиса систему на тестовом стенде разрабатываются файлы: Dockerfile, Docker-compose.yaml и ci-cd.yaml (для настройки деплоя в GitLab CI/CD). Также в настройках CI/CD указываются переменные окружения, аутентификационные данные и прочие настройки, используемые микросервисом. Далее из панели управления GitLab CI осуществляется вызов команды на сборку и деплой микросервиса на указанную виртуальную машину или сервер. CI/CD-runner на целевой машине осуществляет установку необходимых библиотек и пакетов, скачивает исходный код и запускает процесс сборки и запуска целевого приложения. Запущенный в docker-контейнере сервис готов к работе.

5. Интеграция микросервиса в продакшен-решение. На данном шаге производится деплоймент микросервиса на продуктивный стенд Платформы.

Далее приводится описание технологии матричной факторизации подробнее в нашем проекте (см. рис. 7).

Как алгоритм, матричная факторизация (MF) — это операция разложения матрицы на простые составляющие. Матричная факторизация осуществляется с

помощью алгоритма ALS — это итерационный алгоритм, используемый для решения задач матричной факторизации.

Он обычно используется в рекомендательных системах для разложения большой матрицы пользовательских элементов на две меньшие матрицы, которые представляют пользовательские предпочтения и характеристики элементов.

Затем две матрицы перемножаются для того, чтобы сгенерировать прогнозируемый рейтинг для каждой пары пользователь-элемент. ALS работает, чередуя фиксацию одной из двух матриц и решение для другой, пока не будет достигнута сходимость.

В нашем случае строится матрица, в которой по вертикали располагаются заказы пользователей, по горизонтали – различные параметры (данные профилей клиентов, параметры заказов, бюджет, срок, оценка, эмоциональная окраска отзыва и т.д.).

Строки с наибольшим количеством совпадений с параметрами данного идентифицированного клиента составляют список кандидатов для осуществления последующего отбора и ранжирования.

Ранжирование осуществляется в зависимости от количества и весов совпавших параметров (поскольку разные параметры вносят разный вклад в важность всего набора), чем больше параметров совпало и чем больше их вес, тем выше является ранг конкретной подходящей по параметрам строки.

После разработки модели было проведено ее тестирование с целью оценки качества работы. Для этого был проведен анализ наиболее подходящих методик, учитывающих различные разнородные параметры.

Для целей осуществления оценки качества работы алгоритма выдачи нами применяется метрика средневзвешенной ошибки в процентном выражении WAPE (см. формулу (3)).

$$WAPE = \frac{\sum_{t=1}^{n} |A_t - F_t|}{\sum_{t=1}^{n} |A_t|} \quad (3),$$

где At – это фактическое значение (количество строк, выбранных и отсортированных в ходе разметки для тестовой выборки, или исходя из исторических сеансов),

Ft – прогнозное значение (количество правильно выбранных и отсортированных строк алгоритмом).

Для обучения алгоритма применялась выборка из 50 000 сеансов/кейсов оказания услуг пользователям.

Общее количество пользователей в примененной для теста выборке составило 10 тысяч.

В результате, средневзвешенная ошибка по результатам тестирования составила 10.73%.

Общий алгоритм работы модуля ранжирования представлен на Рис. 2.

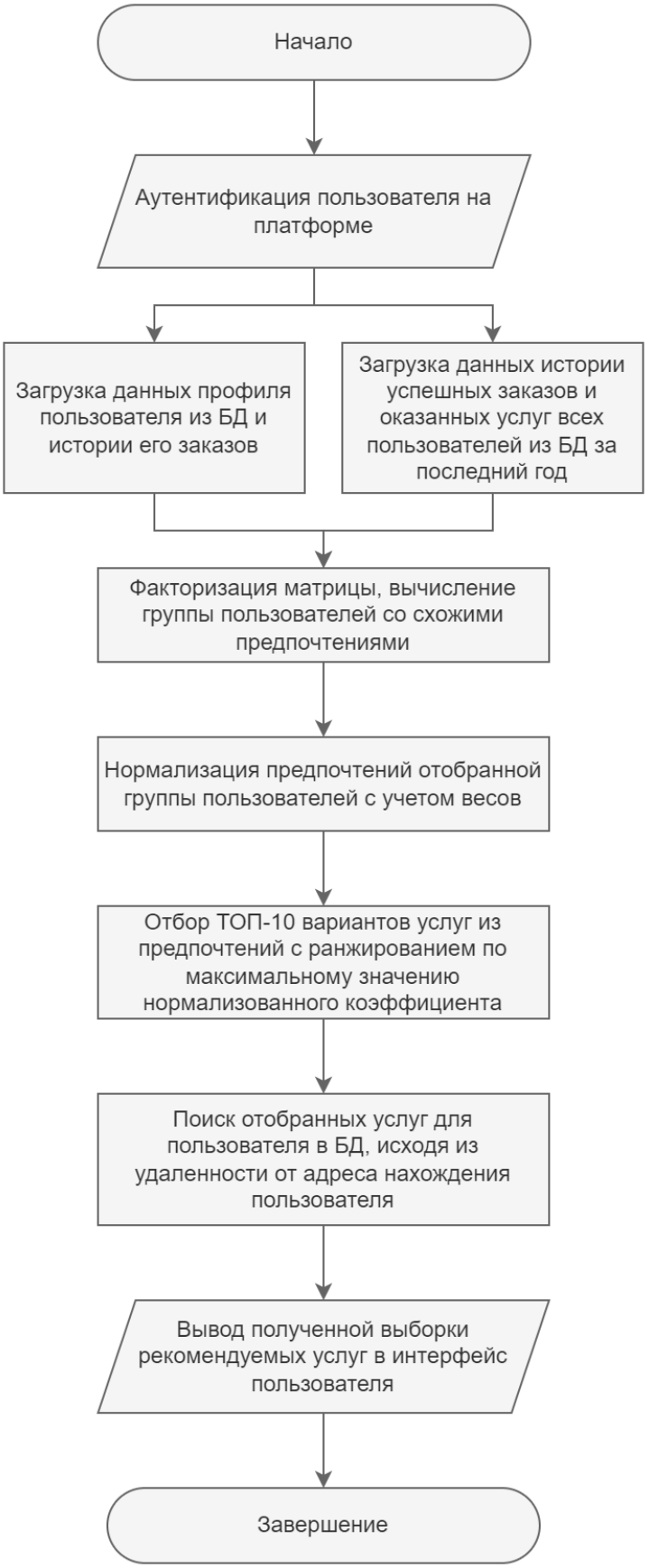


Рисунок 2 – Алгоритм работы модуля ранжирования пользователей

Далее приводится текстовое описание работы модуля целиком.

После аутентификации пользователя на Платформе и переходе на целевую страницу, на которой должны отображаться рекомендуемые услуги, API Платформы отправляет запрос формирование рекомендаций в модуль ранжирования через брокер очередей RabbitMQ.

Модуль ранжирования (архитектура изображена на рис. 5) при получении сообщения с параметрами, включающими идентификатор пользователя, обращается в СУБД PostgreSQL и считывает параметры профиля пользователя с его историей приобретения услуг и проведения сеансов (история заказов) из Elasticsearch. Также модуль получает отфильтрованную выборку истории успешных заказов всех пользователей за последний год из Elasticsearch.

Далее полученные данные подаются на вход модели матричной факторизации, в которой производится вычисление групп пользователей со схожими предпочтениями по заказам и сенсам. С учетом весов по каждому из типов заказов производится нормализация предпочтений. Нормализованная таблица предпочтений сортируется по максимальным значениям весов (интерпретация рейтинга в форме значимости заказа и их количества по популярности среди других пользователей) - нормализационных коэффициентов.

Далее из таблицы отбираются верхние 10 предпочтений (заказов) и по каждому из них осуществляется поиск ближайших таких услуг в базе данных, исходя из удаленности от адреса местонахождения клиента.

Отобранная выборка в виде JSON-данных сообщением через RabbitMQ отправляется в очередь, которую слушает обработчик в API. Обработчик API, получил сообщение, отправляет данные рекомендаций в web-интерфейс через socket-соединение. Данные сохраняются в кэш для исключения частого перерасчета.

При перезагрузке страницы в web-интерфейсе процесс повторяется после экспирации кэша (15 минут).

1.4. Тестирование алгоритма матричной факторизации применимо к ранжированию пользователей

Тестирование алгоритма матричной факторизации проводилось для оценки его эффективности при ранжировании пользователей на основе их прошлых действий, а также для проверки работы данного функционала в составе всей системы.

Для тестирования применялось функциональное и ручное тестирование. Данный метод тестирования дает представление как на уровне отдельных функций, так и на уровне системы в целом.

Тестовая выборка была взята из базы данных “Мой Профи”, которая содержит накопленные за два года сведения из истории сеансов, мастеров и клиентов. Размер выборки для тестирования составил 25 тыс. сеансов и 5 тыс. пользователей. Алгоритм был протестирован путем запуска его на выборочных данных и анализа результатов, чтобы определить, насколько точно он может ранжировать пользователей на основе их прошлых действий. Затем результаты теста использовались для оценки эффективности алгоритма ранжирования пользователей.

Для тестирования модуль с моделью был сконфигурирован на работу с тестовым датасетом вместо всей базы данных. Алгоритм выбирал параметры пользователей из набора в 5000 шт. итеративно и определял проранжированные рекомендуемые услуги/сеансы на выходе. Далее выходные данные сравнивались с историческими результатами: исходными успешно завершенными сеансами этих клиентов и оставленными данными клиентами отзывами.

Также производилась визуальная проверка качества подбора услуг алгоритмом при подаче на вход параметров случайного пользователя. Сотрудник сравнивал параметры пользователя, его историю и оценивал предложенные услуги по бинарной классификации – “подходит”/“не подходит”. Для такого тестирования было проверено не менее 1000 случайных кейсов без повторов. Качество визуального тестирования составило 87.4%.

Мы убедились, что в итоге нами получена успешно обученная на исторических данных модель, которая в целом корректно выполняет свою функцию. Средневзвешенная ошибка по результатам проведенного тестирования составила 10.73%.

2. Этап 2

2.1. Тестирование функционала фильтрации текстового запроса

Тестирование алгоритма фильтрации текстового запроса проводилось для оценки его эффективности при подготовке параметров сравнения и поиска подходящих пар мастер-клиент, а также для проверки работы данного функционала в составе всей системы.

Для тестирования применялось функциональное (в нашем случае Python) и ручное тестирование. Данный метод тестирования дает представление как на уровне отдельных функций, так и на уровне системы в целом.

Тестовая выборка была взята из базы данных “Мой Профи”, которая содержит накопленные за три года сведения из истории поисковых запросов, сеансов, мастеров и клиентов.

Размер подготовленной выборки для тестирования составил 10 тысяч мастеров и 10 тысяч клиентов. Алгоритм был протестирован путем запуска его на выборочных данных и анализа результатов, чтобы определить, насколько точно он может находить соответствия между мастерами и клиентами. Затем результаты теста использовались для оценки эффективности алгоритма поиска соответствий.

2.1.1. Тестирование блока TF-IDF + BOW

Для этого произведена подготовка обучающей и валидационной выборок.

Подготовка обучающего и валидационного датасета началась с сбора данных. Это включает в себя накопление исторических данных из базы данных платформы, что дает возможность использовать реальные данные от пользователей, чтобы обучить модель.

Во входном векторе обучающей выборки помимо самого текстового поискового запроса учитываются другие элементы, такие как: название услуги, специализации мастера, город, станция метро, ценовой диапазон, дата, время и т.д. Информация берется из профиля мастера.

Данные поисковых запросов клиентов использовались за последний год. При использовании реальных данных для обучения модели она становится более способной прогнозировать и классифицировать будущие запросы, которые подаются в систему.

В завершение, после того как модель была обучена на обучающем наборе данных, она тестировалась на валидационном наборе данных.

В результате тестирования модели на тестовой выборке из 10000 экземпляров процент верных сопоставлений запроса клиента с мастером, при которых процент

семантического соответствия текстового запроса клиента информации, внесенной у мастера выше 75%, составил 81.3%.

Также было проведено нагрузочное тестирование на базе JMeter. Для получения точных и надежных результатов тестирование было проведено несколько раз и в разное время, чтобы учесть возможные колебания в загрузке сети и сервера.

Был создан тестовый сценарий (или "тест-план"), который имитирует отправку запросов к серверу и API. JMeter затем замеряет время ответа на каждый запрос и собирает эти данные для дальнейшего анализа.

После проведения нагрузочного теста собранные данные были проанализированы, для этого применялись встроенные отчеты и графики JMeter. Среднее время обработки запроса было вычислено как среднее значение измеренных времен ответа.

Выданные результаты по каждому тестированию были стабильными, без существенных погрешностей.

По результатам тестирования установлено среднее время обработки запроса, которое составляет 0.35 секунды.

Затем была проведена подготовка обучающей и проверочной (валидационной) выборок для реализуемой модели, включая разметку и проверку качества разметки.

Подготовка разметки обучающего и валидационного датасета производились вручную специалистами-разметчиками. Целью разметки являлся выбор профилей мастеров, которых выбирали клиенты с успешным завершением сеанса (заказа) с высокими оценками или положительными отзывами. Для модели были составлены наборы признаков мастера и клиента. Далее было произведено обучение на исторических данных, чтобы определить степень близости конкретного мастера или клиента. Критерием успешности были завершенный сеанс и положительный отзыв клиента.

Разработка модели состояла из следующих этапов:

1) определение признаков клиента:
    - пол;
    - город;
    - геолокация;
    - история прошедших сеансов (услуги, стоимость, дата, время, мастер, оставленные отзывы);
    - нужная услуга;
    - нужная дата;

- нужное время;

2) определение признаков мастера:
   - пол;
   - город;
   - геолокация;
   - история прошедших сеансов (услуги, стоимость, дата, время, клиент, оставленные отзывы);
   - список оказываемых услуг;
   - свободные даты (мастер формирует свой календарь рабочих дней);
   - свободное время (мастер указывает свободные временные слоты).

С целью формирования обучающей выборки, рассчитанной для последующего обучения модели на базе градиентного бустинга, производился программный отбор успешных сеансов (из базы данных Платформы), для которых клиент выбрал мастера и в результате остался доволен услугой / сеансом, сообщив об этом в режиме обратной связи. Такие наборы данных формируются на основе накопленной базы маркетплейса “Мой Профи” (а именно в датасет включаются данные сеансов мастеров и клиентов за последние два года).

Далее для каждого кейса была произведена выборка из базы данных параметров мастера и клиента в соответствии со списком выше. Все пары наборов вручную помечались позитивной меткой в Excel-файле, который далее легко может быть обработан инструментами библиотеки Pandas. Для формирования набора для негативного класса выбираются случаи с отрицательными отзывами и досрочно завершенными сеансами по разным причинам.

Размер сформированной выборки составляет по 20 000 экземпляров для положительного и негативного классов.

2.1.2. Проверка модели на тестовых данных

Для автоматизации создания тестового набора данных применялся PyTest. В рамках настройки тестового окружения для тестирования модуль с моделью был сконфигурирован на работу с тестовым датасетом вместо всей базы данных с использованием функций setup и teardown в PyTest.

Алгоритм выбирал параметры клиентов из набора в 10 000 шт. итеративно и определял соответствия с выборкой мастеров 10 000 шт. Далее выходные данные сравнивались с историческими результатами: поисковыми запросами клиентов и их

фактической записью на сеанс к конкретным мастерам из выборки. Проверки в PyTest были выполнены с использованием assert-выражений.

Также производилась визуальная проверка качества подбора мастеров для клиентов алгоритмом при подаче на вход параметров случайного пользователя. Сотрудник сравнивал параметры пользователя, его профиль и оценивал предложенных мастеров по бинарной классификации – “подходит”/“не подходит”. Для такого тестирования было проверено не менее 1000 случайных кейсов без повторов. Качество визуального тестирования составило 87.4%.

Средневзвешенная ошибка по результатам проведенного тестирования составила 10.73%, что мы считаем оптимальным результатом.

2.2. Сборка модуля автоматического подбора исполнителя и управления его расписанием

Сборка модуля автоматического подбора исполнителя и управления его расписанием сводилась к последовательному подключению всех его модулей к общей Платформе. Это производилось сначала на тестовом стенде, а затем на продукционном стенде.

В общем виде микросервис TF-IDF + BOW представляет собой Docker-контейнер, внутри которого запускается Python-приложение, задачей которого является обработка сообщений от брокера RabbitMQ и отправка результирующего сообщения после завершения работы вычислительного алгоритма.

Когда от API Платформы приходит сообщение о новом поисковом запросе, микросервис по идентификатору клиента собирает данные о нем из PostgreSQL, а также данные о профилях всех подходящих мастеров для последующего формирования входных данных и вызова функции предсказания алгоритма модели. После предсказания микросервис отправляет ответное сообщение с результатом в очередь RabbitMQ.

Следующий модуль, участвовавший в сборке, — модуль GBM. Когда от API Платформы приходит сообщение о необходимости выбрать из списка мастеров наиболее оптимального, микросервис по идентификатору клиента собирает данные о нем, а также данные о профилях всех мастерах в списке для последующего формирования входных данных и вызова функции предсказания алгоритма модели. После предсказания микросервис отправляет ответное сообщение с результатом в очередь RabbitMQ. API Платформы далее может выдать результат пользователю в web-интерфейс посредством socket-соединения.

Для внедрения микросервиса в систему на тестовом стенде разрабатываются файлы: Dockerfile, Docker-compose.yaml и ci-cd.yaml (для настройки деплоя в GitLab CI/CD). Также в настройках CI/CD указываются переменные окружения, аутентификационные данные и прочие настройки, используемые микросервисом. Далее из панели управления GitLab CI осуществляется вызов команды на сборку и деплой микросервиса на указанную виртуальную машину или сервер. CI/CD-runner на целевой машине осуществляет установку необходимых библиотек и пакетов, скачивает исходный код и запускает процесс сборки и запуска целевого приложения. Запущенный в docker-контейнере сервис готов к работе.

Что касается модуля матричной факторизации, когда от API Платформы приходит

сообщение о необходимости выдать ранжированные рекомендации услуг, микросервис по идентификатору клиента собирает данные о нем из PostgreSQL, а также данные о всех сеансах за год для последующего формирования входных данных и вызова функции предсказания алгоритма модели. После предсказания микросервис отправляет ответное сообщение с результатом в очередь RabbitMQ. API Платформы далее может выдать результат пользователю в web-интерфейс посредством socket-соединения.

Внедрение микросервиса на тестовом стенде в систему и тестирование работы Платформы с учетом реализации API-вызовов из Frontend-приложения.

Для внедрения микросервиса систему на тестовом стенде разрабатывались файлы: Dockerfile, Docker-compose.yaml и ci-cd.yaml (для настройки деплоя в GitLab CI/CD). Также в настройках CI/CD указывались переменные окружения, аутентификационные данные и прочие настройки, используемые микросервисом. Далее из панели управления GitLab CI осуществлялся вызов команды на сборку и деплой микросервиса на указанную виртуальную машину или сервер. CI/CD-runner на целевой машине осуществлял установку необходимых библиотек и пакетов, скачивал исходный код и запускал процесс сборки и запуска целевого приложения. Запущенный в docker-контейнере сервис готов к работе.

Финальная сборка на базе Docker представила собой последовательную запаковку микросервисов фильтрации текстового запроса на базе TF-IDF/BOW, фильтрации текстового запроса на базе GBM, ранжирования пользователей на базе MF и подключению их к общему API Платформы.

Сборка была проведена на тестовом стенде, а после успешного тестирования перенесена на продукционный стенд.

Общая архитектура модулей в составе сервисов Платформы приводится на Рис. 3.

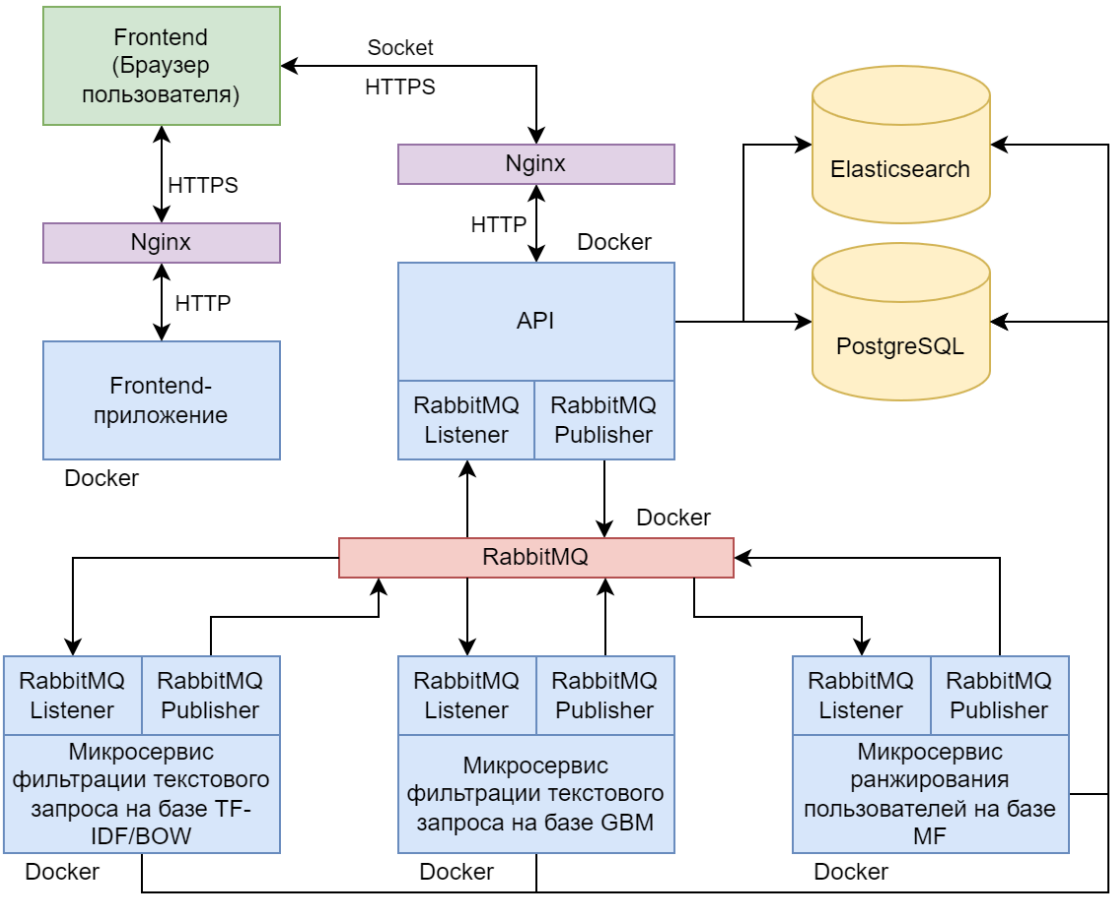


Рисунок 3 – Общая архитектура модулей в составе сервисов Платформы

2.3. Испытание модуля автоматического подбора исполнителя и управления его расписанием

В рамках этапа было проведено ручное тестирование нового программного обеспечения на тестовом и продукционном стендах. Алгоритмы были внедрены только в 10 городах, чтобы исключить региональные различия и точнее оценить их эффективность. По результатам тестирования, сотрудники отметили улучшение качества работы алгоритмов: релевантность клиентов, более точный подбор и увеличение конверсии в запись на 10.67%, уменьшение процента отказов на 20.76% и повышение рейтинга отзывов на 0.75 балла.

2.4. Доработка модуля автоматического подбора исполнителя и управления его расписанием

По результатам испытаний, модель была дообучена: были скорректированы слои, имеющие наиболее абстрактные представления, что позволило уменьшить риск переобучения и сделать модель еще более подходящей для решения задачи. Дообучение включало в себя добавление своих слоев к предварительно обученной модели, их обучение и последующее размораживание и обучение нескольких верхних слоев вместе с добавленной частью модели.

По результатам тестирования проводился анализ того, на какие запросы модель подбирает списки мастеров с ошибочными вариантами. Для всех таких запросов выборка дополнялась размеченными экземплярами с правильным соответствием параметров мастера. Далее на расширенной выборке проводилось переобучение модели. В результате переобучения качество работы выросло до 89.2%.

Анализ запросов к обученной текстовой нейронной модели подбора исполнителя для услуги был проведен путем подачи различных текстовых запросов на вход модели и анализа ее выходных результатов.

В процессе анализа были оценены следующие характеристики модели:

1. Точность классификации;
2. Полнота классификации;
3. F-мера;
4. Оценка качества ранжирования.

По всем оцененным характеристикам результаты оценок и сравнения были собраны в диаграмму. На рисунке 4 представлена столбчатая диаграмма результата анализа запросов к обученной текстовой нейронной модели. Анализ проводился на случайной выборке из 1000 элементов. F-мера получилась равной 0,915.

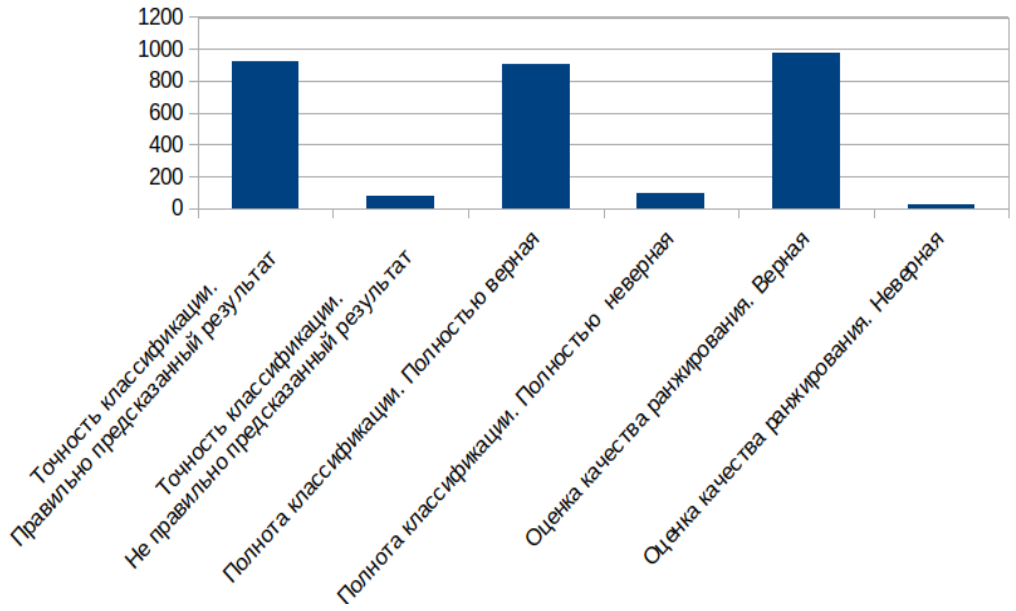


Рисунок 4 – Столбчатая диаграмма результата анализа запросов к обученной текстовой нейронной модели

В ходе проведения анализа запросов к обученной модели было определено, что модель успешно выполняет задачу подбора исполнителей для услуги на основе текстовых запросов пользователей. Модель предлагает точные и релевантные варианты мастеров, учитывая все параметры и предпочтения пользователей. Тестирование и оценка результатов после внесения изменений также подтвердили, что ошибочные варианты были успешно устранены, что является хорошим результатом анализа.

По результатам тестирования проводилось оптимизация структур данных исходного кода для повышения производительности. Был проведен анализ использования структур данных, внесены изменения в приложение, где это было необходимо. Оптимизация структур данных модуля включала улучшение скорости поиска с использованием оптимизированных структур данных и хеш-таблиц. Это ускорило процесс поиска исполнителей для запросов и сократило время ожидания пользователей со 100 мс до 0,1 мс. Для предотвращения коллизий использовался криптографический алгоритм SHA-512, который генерирует 512-битный хеш, что снижает вероятность коллизий до минимума.

Также проведена оптимизация хранения данных: для этого мы использовали сериализацию данных для уменьшения объема памяти, занимаемого структурами данных. Использовали модуль Pickle для сериализации и десериализации данных, что уменьшило объем занимаемой памяти на 20%. 3. Улучшили эффективность операций обновления: разработали эффективные алгоритмы обновления данных с использованием дерева отрезков, что минимизировало время и ресурсы, требуемые для

обновления данных. Если ранее обновление занимало O(n+m), то теперь оно занимает только O(1).

Первоначальные замеры показали, что обновление данных для базы из 1000 исполнителей и 5000 интервалов занимало бы в среднем 0,5 секунды. После оптимизации это время уменьшилось до 0,05 секунды.

Поскольку модуль обрабатывает большой объем запросов и имеет высокую нагрузку, были использованы параллельные вычисления на базе Python - Multiprocessing для ускорения операций подбора исполнителей и управления расписанием. Было добавлено распределение запросов между несколькими потоками.

В частности, с использованием multiprocessing.Pool(), был создан пул процессов, размер которого определяется количеством доступных ядер процессора. После создания пула данные распределяются между процессами функцией map. После обработки данных каждым процессом, результаты собираются обратно в главном процессе. Функция map автоматически собирает эти результаты в список.

Проведено профилирование кода, чтобы определить, какие части кода являются "узкими местами", замедляющими общую производительность. Затем мы сосредоточились на улучшении эффективности этих участков кода, переписывая их более эффективным образом или используя более быстрые алгоритмы. Профилирование осуществлялось через cProfile. Результаты профилирования для последующего анализа сохранялись с помощью модуля pstats:

```
import cProfile
    import pstats
    import re
    def test():
       [re.compile('test') for _ in range(10000)]
    # Запуск профилирования с сохранением результатов в файл 'output.pstats'
    cProfile.run('test()', 'output.pstats')
    # Чтение данных профилирования из файла и их анализ
    p = pstats.Stats('output.pstats')
    p.sort_stats('cumtime').print_stats(10)  # Сортируем и печатаем статистику для 10 самых "тяжелых" функций
```

С учетом среднего времени обработки запроса моделью среднее суммарное время отклика удалось уменьшить до 0.25 секунд, что значительно улучшило производительность.

Последующая доработка в этой части не требуется.

3. Этап 3

3.1. Разработка алгоритма анализа расходов и планирования закупок расходных материалов

Целью данного мероприятия является исследование применения алгоритма LSTM модели к решению проблемы анализа расходов и планирования закупок расходных материалов.

Алгоритм LSTM (Long Short-Term Memory) — это специализированный тип рекуррентной нейронной сети (RNN), разработанный для решения проблемы исчезающего градиента, с которой сталкиваются традиционные RNN. Это достигается благодаря внедрению структуры ячеек памяти, которые позволяют сети сохранять информацию на длительные периоды времени.

Работа алгоритма состоит из нескольких последовательных этапов и строится следующим образом. Задача алгоритма на выходе состоит в предсказании на каждый последующий день (в горизонте месяца вперед) необходимости заказа позиций и их количества в единицах по материалам каждого типа с учетом многих факторов. Если необходимость заказа определена положительно, то производится умный поиск вариантов по каждой позиции с учетом цен, скидок и оценок.

По результатам прогнозирования на каждый будущий день производится агрегация одинаковых позиций с целью единовременного заказа материала на несколько дней или недель вперед, в соответствии со сроками хранения/годности позиции и предпочтительным периодом, выбранным конкретным мастером. Например, мастер может выбрать в приложении минимальный период, на который он предпочитает обычно формировать запас материалов.

В первую очередь производится выборка и подготовка входных данных для последующего анализа необходимости пополнения запасов. Основная часть данных, необходимых для анализа вводится периодически самим мастеров в приложении, другая часть формируется на основе истории. Вектор признаков включает следующие составляющие:

- Средний расход материала на одного клиента;
- Текущий запас материала;
- Количество клиентов на день расчета с учетом расписания мастера;
- Прогноз количества клиентов на день расчета исходя из исторических данных, формируемый моделью (на базе искусственных нейронных сетей типа LSTM) и учитывающий сезонность, день недели, выходные и праздники;

- Исторические данные за предыдущий месяц (при наличии такой истории) по расходу и покупкам данной позиции материала, нормированный на количество клиентов, как для текущего мастера, так и у других мастеров, оказывающих одинаковые услуги в схожем по профилю регионе;
- Исторические данные за аналогичный месяц год назад (при наличии такой истории) по покупкам данной позиции материала, нормированный на количество клиентов для учета сезонности.

Далее данный вектор подается в модель прогнозирования необходимости пополнения запаса. Таким образом на данном шаге алгоритм включает пайплайн из двух LSTM-моделей (Рис. 5).

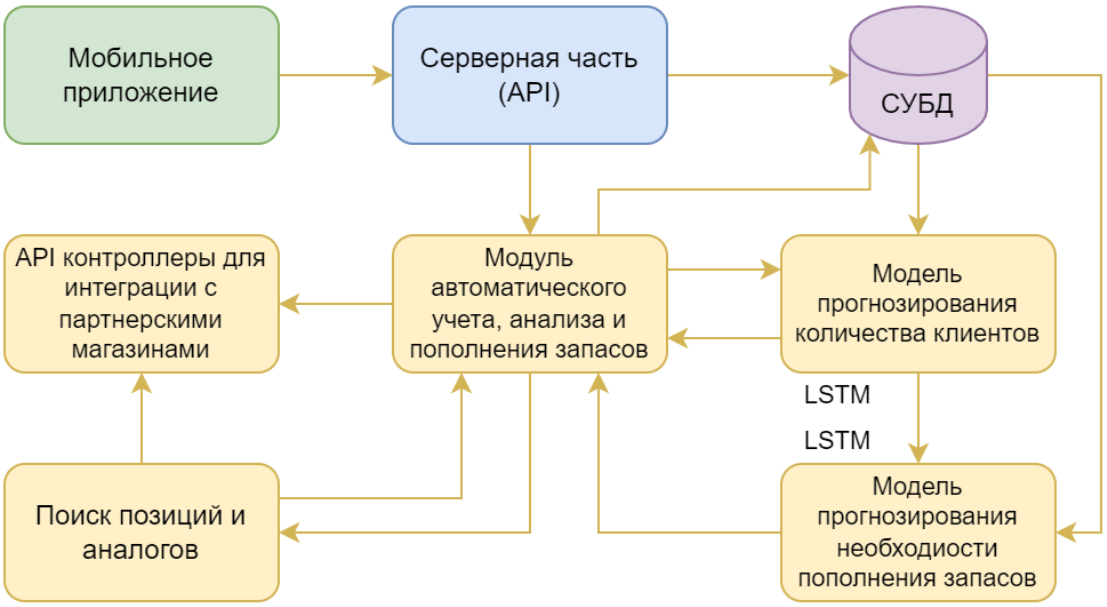


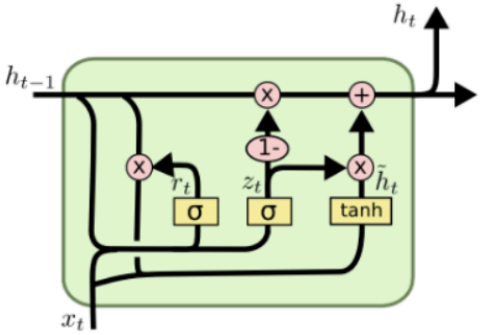
Рисунок 5 – Алгоритм работы моделей LSTM

Первая модель предсказывает возможное среднее количество клиентов на каждый день на месяц вперед для донасыщения вектора признаков. Причем прогнозирование производится за один проход, так как применяется модель типа Multiple Parallel Input - Multiple Output LSTM, которая на выходе выдает вектор со значениями количества ожидаемых клиентов на каждый день периода.

Вторая модель типа Multiple Parallel Input - Single Output LSTM по указанному выше вектру признаков осуществляет итерационное прогнозирование для каждой используемой в работе мастером позиции на каждый день вперед (X позиций * 30 дней) и определяет числовое значение необходимого количества материала. Если значение

равно 0, то необходимости пополнения запасов данного материала в конкретный день нет.

Для реализации моделей используется архитектура LSTM-сетей (рисунок 6). LSTM (Long short-term memory) - это искусственная нейронная сеть, содержащая специальные рекуррентные модули, способные запоминать значения как на короткие, так и на длинные промежутки времени. Ключом к данной возможности является то, что LSTM-модуль не использует функцию активации внутри своих рекуррентных компонентов, что дает преимущество LSTM-сетям в обучаемости по сравнению с RNN.



$$z_t = \sigma\left(W_z \cdot [h_{t-1}, x_t]\right)$$
$$r_t = \sigma\left(W_r \cdot [h_{t-1}, x_t]\right)$$
$$\tilde{h}_t = \tanh\left(W \cdot [r_t * h_{t-1}, x_t]\right)$$
$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

Рисунок 6 — Архитектура LSTM-сетей

Для программной реализации моделей используются библиотеки PyTorch, Numpy и Pandas для языка программирования Python 3.8.

Для каждого элемента во входной последовательности каждый слой сети вычисляет следующие функции:

$$\begin{aligned}
i_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{t-1} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{t-1} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{t-1} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{t-1} + b_{ho}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

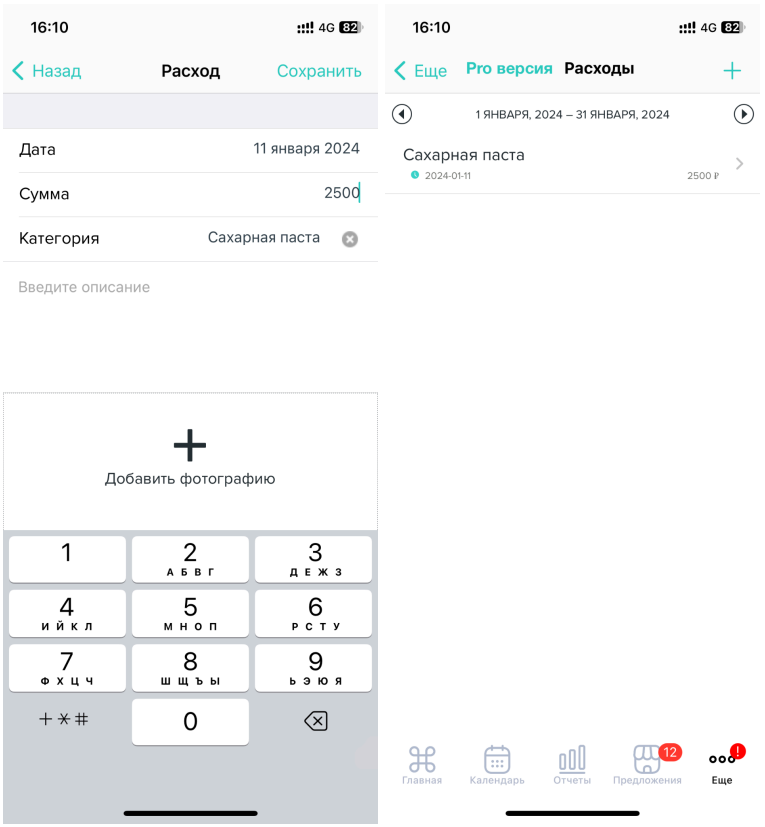
Для оценки качества при решении задачи регрессии при прогнозировании результатов в обеих моделях применяется среднеквадратичная ошибка MSE.

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

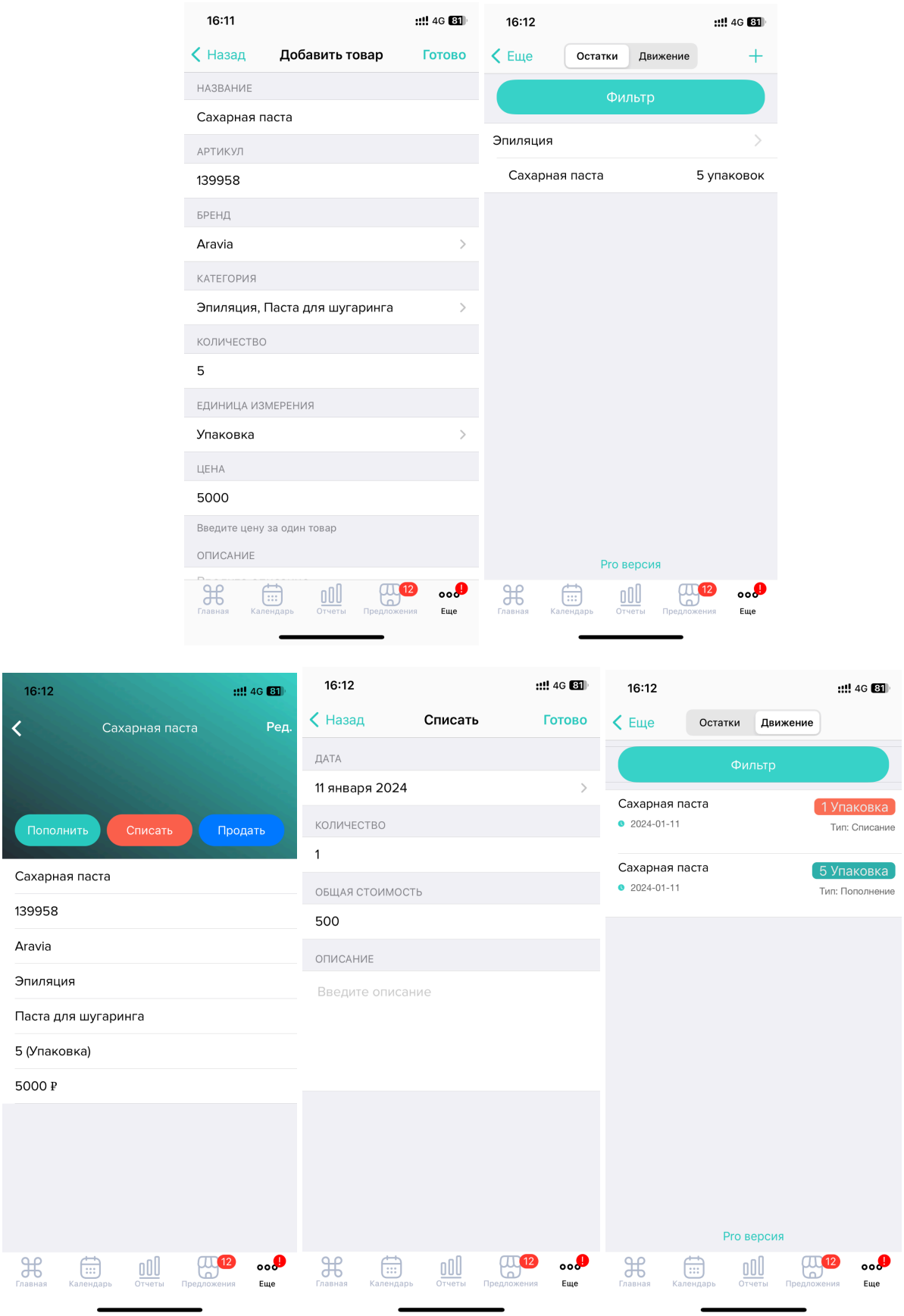
Для обучения модели производится разметка данных, включающих историю за 3 года по различным позициям материалов.

Наиболее полный стек технологий с учетом серверной части и доработок мобильных приложений включает: Python, Swift, Kotlin, React Native, TypeScript, JavaScript, Nginx, Django Rest Framework, PostgreSQL, MongoDB, Redis, RabbitMQ, Pytorch, Numpy, Pandas, OpenAPI, JSON.

Пользовательские интерфейсы приведены на рисунках 7-.



Рисунки 7-8 — Пользовательский интерфейс учета расходов



Рисунки 9-13 — Пользовательский интерфейс учета остатков материалов

3.2. Тестирование алгоритма анализа расходов и планирования закупок расходных материалов

Тестирование алгоритма анализа расходов и планирования закупок проводилось для оценки его эффективности при планировании объема закупок (в том числе в условиях изменяющейся сезонности), а также для проверки работы данного функционала в составе всей системы.

Для тестирования применялось функциональное и ручное тестирование. Данный метод тестирования дает представление как на уровне отдельных функций, так и на уровне системы в целом.

Функциональное тестирование рассматривает продукт или системный компонент, состоящий из набора классов, процессов, модулей и данных, в целом. На этом этапе проверяется его общая производительность, функциональные и технические характеристики, а также бизнес-логика, включая показатели качества. Проверка выполняется в нескольких конфигурациях аппаратной среды и наборов данных. Это тестирование реализуется путем разработки и запуска модульных тестов на языке программирования (в нашем случае Python).

Ручное тестирование — выполнение тестировщиком прохода тестового цикла вручную, с последующей ручной фиксацией результатов по каждому тесту в отчете. Для данного вида тестирования тестировщик использует интерфейс пользователя и осуществляет повторение пользовательских сценариев, отражающих работу тестируемого модуля или функционала.

Тестовая выборка была взята из базы данных “Мой Профи”, которая содержит накопленные за три года сведения из истории внесенных расходов и фактических закупок, осуществленных мастерами.

Мы проводили тестирование алгоритма планирования закупок на точность, потому что, во-первых, оно помогает убедиться, что алгоритм работает должным образом и что результаты надежны. Во-вторых, оперативное проведение тестирования также способствует выявлению любых потенциальных проблемы с алгоритмом (например, недообучение), которые могут привести к неточным результатам. В-третьих, тестирование на точность также помогает определить любые потенциальные области улучшения алгоритма, которые могут помочь усовершенствовать его эффективность работы.

Размер подготовленной выборки для тестирования составил 5 тысяч мастеров и 100 тысяч итераций закупок мастерами расходных материалов. Алгоритм был

протестирован путем запуска его на выборочных данных и анализа результатов, чтобы определить, насколько точно он может прогнозировать планирование закупок по сравнению с историческими данными (данные брались за следующий год, после тех трех, что использовались для обучения). Затем результаты теста использовались для оценки эффективности алгоритма планирования закупок. По результатам предварительного тестирования точность работы предиктивной модели составила 93,1%.

3.3. Подключение интеграций с интернет-магазинами

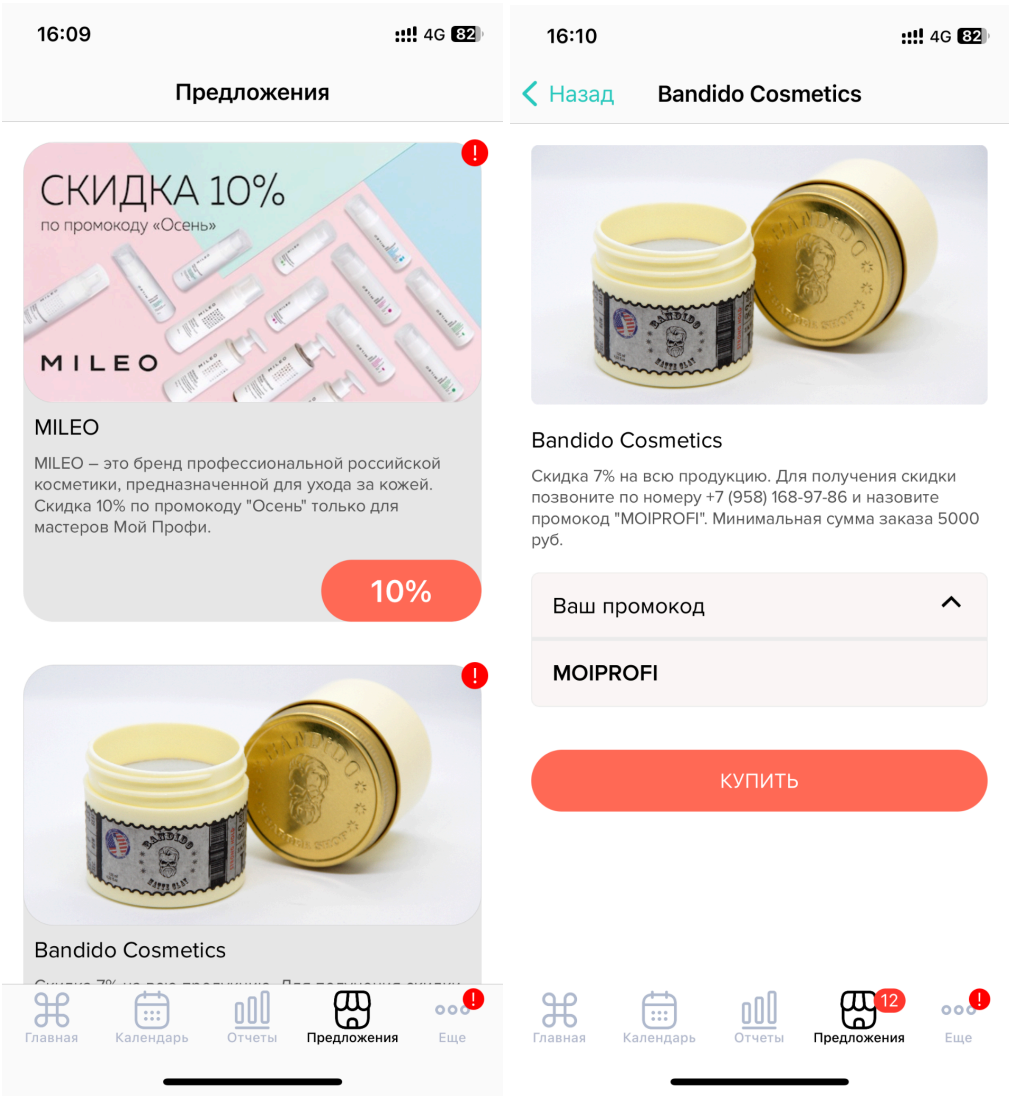
На этом этапе производится агрегация позиций и поиск в интернет-магазинах партнеров данных позиций и аналогов с целью предложения менее дорогих вариантов с учетом скидок. При поиске учитываются сроки поставки и наличие необходимого количества. Для интеграции с партнерскими магазинами разрабатываются контроллеры REST API для каждой площадке.

Список партнерских магазинов:

- Mileo
- Bandido Cosmetics
- Elegant Lash
- Lak Store
- Perfleor
- VlasonShop
- VogueNailsRu
- Ibeauty Russia
- Cracy
- Крымская дива
- Paul Oscar
- 1753 Cosmetics
- Mascia Brunelli
- Беатриче

API позволяет узнать актуальные цены и каждый вид товара и выбрать наиболее выгодное предложение. API позволяет автоматически выполнить заказ товара в нужном количестве и к нужной дате (с учетом времени доставки).

Пользовательский интерфейс маркетплейса представлен на рисунках 14-15:



Рисунки 14-15 — Пользовательский интерфейс маркетплейса

3.4. Сборка модуля автоматического учета, анализа и пополнения запасов

Сборка модуля автоматического учета, анализа и пополнения запасов сводилась к его подключению к общей Платформе. Это производилось сначала на тестовом стенде, а затем на продукционном стенде.

Компоненты модуля “Модель прогнозирования количества клиентов” и “Модель прогнозирования необходимости пополнения запасов” последовательно были подключены друг к другу и к СУБД PostgreSQL. После этого модуль был подключен к API интеграции с партнерскими магазинами и к основному API серверной части.

Для внедрения микросервиса в систему на тестовом стенде разрабатываются файлы: Dockerfile, Docker-compose.yaml и ci-cd.yaml (для настройки деплоя в GitLab CI/CD). Также в настройках CI/CD указываются переменные окружения, аутентификационные данные и прочие настройки, используемые микросервисом. Далее из панели управления GitLab CI осуществляется вызов команды на сборку и деплой микросервиса на указанную виртуальную машину или сервер. CI/CD-runner на целевой машине осуществляет установку необходимых библиотек и пакетов, скачивает исходный код и запускает процесс сборки и запуска целевого приложения. Запущенный в docker-контейнере сервис готов к работе.

На шаге интеграции микросервиса в продакшен-решение производится деплоймент микросервиса на продуктивный стенд Платформы.

Далее приводится текстовое описание работы модуля целиком.

Модуль с автоматически запускается на сервере раз в сутки и производит планирование закупок расходных материалов.

Модель прогнозирования количества клиентов делает запрос в СУБД и строит прогноз по количеству клиентов на год вперед.

Модель прогнозирования необходимости пополнения запасов, строит прогноз на скорость расходования материалов и с учетом имеющегося количества на складе получает ближайшую дату окончания конкретного материала.

Через API партнерских магазинов происходи заказ материалов в нужном количестве с поправкой на срок доставки.

Цикл повторяется на следующие сутки.

Финальная сборка представила собой запаковку микросервиса автоматического учета, анализа и пополнения запасов и подключению его к общему API Платформы. Под запаковкой подразумевается процесс упаковки отдельных компонентов программного обеспечения (в данном случае микросервисов) в стандартизированные

единицы развертывания, которые могут быть независимо развернуты и запущены в любой среде. Это позволяет избежать проблем с зависимостями и обеспечивает однородность среды развертывания, что существенно упрощает процесс разработки, тестирования и внедрения ПО.

Нами был выбран наиболее распространенный подход к "запаковке" — им является использование контейнеров, таких как Docker. Контейнеры позволяют упаковывать и изолировать приложения с их окружением, что обеспечивает более удобное и надежное развертывание приложений. Выбор в пользу Docker-контейнеров связан с тем, что они предлагают ряд преимуществ, которые делают их более привлекательными для некоторых сценариев, по сравнению с другими способами запаковки и распространения ПО, такими как виртуальные машины или обычные исполняемые файлы. Вот некоторые из этих преимуществ:

- Изоляция и стандартизация: Docker обеспечивает стандартизированное окружение, которое изолировано от хост-системы.

- Легковесность: Docker-контейнеры, в отличие от виртуальных машин, не содержат полной операционной системы. Они используют ядро операционной системы хоста и добавляют только те компоненты, которые необходимы для работы конкретного приложения. Это делает их более быстрыми и эффективными по сравнению с виртуальными машинами.

- Модульность и масштабируемость: Docker позволяет создавать модульные компоненты, которые могут быть развернуты и масштабированы независимо друг от друга. Это может облегчить процесс развертывания и управления сложными системами.

- Версионирование и управление зависимостями: Docker позволяет явно указывать и управлять зависимостями вашего приложения, что облегчает его развертывание и поддержку. Благодаря поддержке версионирования можно легко откатиться к предыдущей версии приложения, если что-то работает недостаточно корректно в новой версии.

- Удобство для разработки: Docker упрощает процесс настройки окружения для разработки, позволяя разработчикам работать в условиях, максимально приближенных к производственным, и минимизировать проблемы, связанные с различием окружений.

- Поддержка CI/CD: Docker очень хорошо интегрируется с системами непрерывной интеграции и непрерывной доставки (CI/CD), что делает его привлекательным инструментом для современных разработчиков.

В контексте микросервисов каждый микросервис был упакован в свой собственный контейнер, что позволяет развертывать и масштабировать каждый микросервис независимо.

Сборка была проведена на тестовом стенде, а после успешного тестирования перенесена на продукционный стенд.

3.5. Проведение испытаний маркетплейса

Испытания проводились сначала на тестовом стенде, а затем на продукционном стенде.

Тестовый стенд воспроизводил реальную операционную среду, что позволило убедиться, что ПО работает корректно в условиях, максимально приближенных к реальным.

Продукционный стенд в контексте тестирования программного обеспечения - это реальная операционная среда, где конечные пользователи взаимодействуют с программным продуктом. Здесь модуль работает в нормальном режиме, обрабатывая реальные запросы и выполняя свои функции.

В качестве вида тестирования было выбрано ручное тестирование программного обеспечения - это процесс проверки функциональности и корректности работы программного обеспечения вручную, без использования автоматизированных инструментов или скриптов. В рамках этого процесса тестировщики выполняют заданные тестовые сценарии и пытаются найти возможные ошибки или несоответствия заданной спецификации. Основное преимущество ручного тестирования в том, что оно позволяет тестировщикам внимательно исследовать различные аспекты программного обеспечения и обнаружить ошибки, которые могут быть пропущены автоматизированными тестами. Однако ручное тестирование может быть трудоемким и требует значительных усилий и времени, что было учтено при разработке программы тестирования и привлечении тестировщиков .

Для сравнительного анализа и оценки эффективности новых алгоритмов было принято решение внедрения модуля анализа расходов и планирования закупок расходных материалов в 10 городах (для исключения региональных факторов):

- Воронеж
- Екатеринбург
- Санкт-Петербург
- Самара
- Новосибирск
- Краснодар
- Красноярск
- Казань
- Сыктывкар
- Челябинск

Причем чтобы исключить возможные погрешности, было принято решение в ряде городов применять алгоритм модуля только к некоторым услугам.

Таким образом в Воронеже, Екатеринбурге и Санкт-Петербурге в испытании участвовали все услуги:

- Ногтевой сервис
- Ресницы
- Брови
- Эпиляция
- Парикмахерские услуги
- Косметология
- Перманентный макияж
- Массаж
- Спа-процедуры

В Самаре и Новосибирске в испытании участвовали только следующие услуги:

- Ногтевой сервис
- Ресницы

В Красноярске и Краснодаре в испытании участвовали только следующие услуги:

- Брови
- Эпиляция

В Казани и Сыктывкаре в испытании участвовали только следующие услуги:

- Парикмахерские услуги
- Косметология
- Перманентный макияж

В Челябинске в испытании участвовали только следующие услуги:

- Массаж
- Спа-процедуры

По результатам испытания, все сотрудники, принимавшие участие в тестировании, заявили о релевантности прогнозов трат расходных материалов и адекватности заказываемых объемов. Погрешность (избыточность) прогнозируемых заказов составила 4,2%.

3.6. Доработка маркетплейса по результатам испытаний

По результатам испытаний было проведено дообучение модели модуля анализа расходов и планирования закупок расходных материалов.

Дообучение — это процесс добавления новых данных, полученных в результате проведения тестирования модуля анализа расходов и планирования закупок расходных материалов на продукционном стенде. Суть дообучения заключается в размораживании последних слоёв нейронной сети и их обучении. Таким образом, корректируются слои, которые имеют наиболее абстрактные представления. Производя дообучение только нескольких слоев, мы уменьшаем риск переобучения (overfitting). И самое главное, это позволило сделать текущую модель ещё более подходящей к нашей задаче.

Если верхние слои следует дообучить, то полносвязные слои нужно заменить на свои и тоже обучить. В нашем случае процедура дообучения состояла в следующем:

1. Заморозили все слои предварительно обученной модели.
2. Добавили свои слои к обученной модели.
3. Обучили добавленные слои.
4. Разморозили несколько верхних слоев.
5. Обучили эти слои и добавленную часть вместе.

Согласно результатам тестирования, модель показала некоторые ошибки в прогнозировании закупки с ошибочными вариантами. Выявление ранее неучтенного фактора – срока годности продукции, позволило скорректировать алгоритмы планирования. Теперь система заказывает меньше расходных материалов с коротким сроком годности.

После корректировки выборки проводилось переобучение модели, которое привело к снижению объема заказов на 5,7%. Итоговая точность алгоритма персонализированной ленты заказов достигла 96,04%, что является хорошим результатом. Это означает, что система теперь способна более точно прогнозировать спрос и оптимизировать процесс закупки.

Последующая доработка в этой части не требуется.

3.7. Разработка документации

На протяжении всего цикла разработки, программистами использовались автодокументация кода, что значительно повышает ясность понимания каждого отдельного алгоритма и его частей.

Кроме того была разработана документация по программе:

1) Заключительный научно-технический отчет;
2) Спецификация (Приложение 1);
3) Программа и методика испытаний (Приложение 2);
4) Протокол испытаний (Приложение 3);
5) Описание программы (Приложение 4);
6) Текст программы (Приложение 5).

Была разработана документация по тестированию: программа и методика испытаний, протокол испытаний.

4. Регистрация интеллектуальной собственности

ООО “Умный Бизнес” является правообладателем первой версии Платформы “Мой Профи” (регистрационный номер 2022665967 от 23.08.2022 г.).

В рамках НИОКР, ООО “Умный Бизнес” приняло ряд мер по защите интеллектуальной собственности, в частности:

1) Установлен актуальный режим коммерческой тайны.
2) Подано заявление о включении сведений о продукции, созданной за счет полученного гранта, в Единый реестр российских программ для электронных вычислительных машин и баз данных.

В частности, в поданном заявлении упомянуто, что продукт разработан при поддержке ФГБУ “Фонд содействия развитию малых форм предприятий в научно-технической сфере" (Фонд содействия инновациям) в рамках реализации федерального проекта "Искусственный интеллект", конкурс "Развитие-ИИ" (очередь III).

Результатом данного мероприятия является получение свидетельства о регистрации прав на программу для ЭВМ “Мой Профи 2.0”, что соответствует критерию результата реализации проекта в соответствии с приказом Министерства экономического развития Российской Федерации от 29 июня 2021 г. № 392 «Об утверждении критериев определения принадлежности проектов к проектам в сфере искусственного интеллекта».

ЗАКЛЮЧЕНИЕ

В рамках первого (промежуточного) этапа НИОКР были проведены следующие работы. Разработана модель фильтрации текстового запроса, предназначенная для интеллектуального анализа поискового запроса клиента и подбора оптимального мастера на основе технологий TF-IDF и BOW. Разработана Модель фильтрации текстового запроса на базе GBM для сравнения параметров мастера и клиента с целью выбора наилучшего варианта. Разработан алгоритм матричной факторизации для ранжирования пользователей с использованием Python-библиотек Numpy, Sklearn и Pandas. Для оценки эффективности и проверки работы функционала в составе системы было проведено тестирование MF-алгоритма. Результаты показали средневзвешенную ошибку 10,73% и качество работы модели 87,4%. Визуальное тестирование показало качество 87,4% для 1000 кейсов.

По результатам работ второго (промежуточного) этапа было проведено тестирование функционала фильтрации текстового запроса. Были произведены сборка модуля автоматического подбора исполнителя и управления его расписанием. Модуль включает микросервисы фильтрации текстового запроса на базе TF-IDF/BOW, фильтрации текстового запроса на базе GBM и ранжирования пользователей на базе MF. По результатам тестирования модуля было выявлено увеличение точности подбора и конверсии в запись на 10.67%, уменьшение процента отказов на 20.76% и повышение рейтинга отзывов на 0.75 балла. В рамках доработки по результатам испытаний удалось существенно улучшить производительность модели, снизить риск переобучения и увеличить качество работы до 89.2%. Использование хеш-таблиц, криптографического алгоритма SHA-512, сериализации данных и параллельных вычислений позволило сократить время ожидания пользователей, улучшить эффективность приложения и обеспечить безопасность данных. Оптимизация операций обновления данных и профилирование кода также способствовали повышению производительности приложения.

Во время третьего (заключительного) этапа НИОКР были проведены следующие работы. Разработан алгоритм анализа расходов и планирования закупок расходных материалов на базе LSTM моделей. Первая модель типа Multiple Parallel Input - Multiple Output LSTM предсказывает среднее количество клиентов на каждый день на месяц вперед для донасыщения вектора признаков. Вторая модель типа Multiple Parallel Input - Single Output LSTM осуществляет итерационное прогнозирование для каждой позиции на каждый день вперед и определяет необходимое количество материала. Если значение

равно 0, то пополнение запасов не требуется. Стек технологий включает: Python, Swift, Kotlin, React Native, TypeScript, JavaScript, Nginx, Django Rest Framework, PostgreSQL, MongoDB, Redis, RabbitMQ, Pytorch, Numpy, Pandas, OpenAPI, JSON.

Протестирован алгоритм анализа расходов и планирования закупок расходных материалов путем запуска его на выборочных данных и анализа результатов. По результатам предварительного тестирования точность работы предиктивной модели составила 93,1%.

Произведена интеграция с интернет-магазинами на базе контроллеров REST API для каждой площадки.

Осуществлена сборка модуля автоматического учета, анализа и пополнения запасов на базе Docker. Модуль запускается ежедневно, делает прогнозы по количеству клиентов и необходимости пополнения запасов, заказывает материалы через API партнерских магазинов с учетом сроков доставки. Финальная сборка модуля интегрирована в продакшен-решение через общее API Платформы.

Проведены испытания маркетплейса. По результатам испытания, все сотрудники, принимавшие участие в тестировании, заявили о релевантности прогнозов трат расходных материалов и адекватности заказываемых объемов. Погрешность (избыточность) прогнозируемых заказов составила 4,2%.

По результатам испытаний было проведено дообучение модели модуля анализа расходов и планирования закупок расходных материалов. Итоговая точность алгоритма персонализированной ленты заказов составила 96.04%.

Разработана программная документация, включающая следующие документы:

1) Спецификация (Приложение 1);
2) Программа и методика испытаний (Приложение 2);
3) Протокол испытаний (Приложение 3);
4) Описание программы (Приложение 4);
5) Текст программы (Приложение 5).

Подана заявка на регистрацию интеллектуальной собственности (реквизиты заявления: номер 293892 от 13.02.2024 г.).

Новые модули обеспечивают следующие функциональные возможности маркетплейса “Мой Профи 2.0”:

1) Формирование персонализированной ленты заказов, соответствующих свободным временным слотам и подходящих конкретному мастеру по параметрам (ожидаемое время выполнения заказа, время в пути, предпочтительные для работы часы и дни) с точностью более 95%;

2) Формирование бесшовного расписания мастера;
3) Автоматический учет, анализ и пополнение запасов с точностью более 92% и функциями:

a. Учета остатков расходных материалов и косметики;
b. Рекомендаций аналогов товаров по более низкой цене;
c. Планирования закупок;
d. Напоминаний о закупках;
e. Интеграции с интернет-магазинами для оформления заказа не выходя из маркетплейса “Мой Профи”.

Также в случае ручного формирования расписания или редактирования предполагаемого системой, под каждое свободное окно в расписании будет подбираться заказ с учетом среднего времени выполнения на основе исторических данных.

Выполненные работы по объему и достигнутым функциональным характеристикам соответствует изначальным планам проекта, представленным в техническом задании на выполнение НИОКР (см. Табл. 1).

Таблица №1

Пункт ТЗ	**Требуемые показатели**	**Фактические показатели**	**Выполнено (+/–)**	**Подтверждающие документы**
Функции, выполнение которых должен обеспечивать разрабатываемый научно-технический продукт	Формирование персонализированной ленты заказов, соответствующих свободным временным слотам и подходящих конкретному мастеру по параметрам: ожидаемое время выполнения заказа, время в пути, предпочтительные для работы часы и дни	В рамках проекта (1, 2 этап) разработан Модуль автоматического подбора исполнителя и управления расписанием, обеспечивающий следующие функциональные возможности: - Формирование персонализированной ленты заказов, соответствующих свободным временным слотам и подходящих конкретному мастеру по параметрам: ожидаемое время выполнения заказа, время в пути, предпочтительные для работы часы и дни;	+	1. Описание программы (Приложение 4); 2. Протокол испытаний (Приложение 3)
	Формирование бесшовного расписания мастера		+	

		- Формирование бесшовного расписания мастера.		
	Учет остатков расходных материалов и косметики	В рамках проекта (3 этап) разработан Модуль автоматического учета, анализа и пополнения запасов, обеспечивающий следующие функциональные возможности: - Учет остатков расходных материалов и косметики; - Рекомендации аналогов товаров по более низкой цене; - Планирование закупок; - Напоминания о закупках.	+	
	Рекомендации аналогов товаров по более низкой цене		+	
	Планирование закупок		+	
	Напоминания о закупках		+	
	Интеграции с интернет-магазинами для оформления заказа не выходя из приложения/платформы “Мой Профи”	В рамках проекта (3 этап) подключена интеграция с интернет-магазинами для оформления заказа не выходя из приложения/платформы “Мой Профи”	+	
Количественные параметры, определяющие выполнение научно-техническим продуктом своих функций	Точность работы предиктивной модели модуля автоматического учета, анализа и пополнения запасов - не ниже 92%	93,1%	+	1. Программа и методика испытаний (Приложение 2); 2. Протокол испытаний (Приложение 3)
	Точность алгоритма персонализированной ленты заказов - не ниже 95%	96,04%	+	
	Рост записей к мастерам на 10%	10%	+	
	Повышение LTV клиента мастера в среднем на 25%	25%	+	
	Увеличение среднего чека на 15%	20%	+	

Требования по патентной охране	Разработка должна соответствовать требованиям к патентной чистоте, прописанным в законодательстве РФ Государственная регистрация разработки в качестве программы для ЭВМ запланирована на декабрь 2023 года	Подана заявка на регистрацию интеллектуальной собственности	+	Заявление о включении сведений о программном обеспечении в реестр российского программного обеспечения (реквизиты заявления: номер 293892 от 13.02.2024 г.).

Кроме того, как следует из исследований, проведенных экспертами нашей компании, Платформа имеет значительные конкурентные преимущества перед иностранными маркетплейсами специалистов и не имеет прямых конкурентов на российском рынке, фактически являясь уникальным в своём роде программным решением.