Федеральное государственное автономное образовательное учреждение
высшего образования
«Национальный исследовательский университет
«Высшая школа экономики»

Факультет компьютерных наук
Основная образовательная программа
Прикладная математика и информатика

ГРУППОВАЯ КУРСОВАЯ РАБОТА

Программный проект

На тему

“Чат бот рекомендует”

Выполнили студенты:

Кудрявцева Софья Павловна, гр. БПМИ-174

Попов Илья Иванович, гр. БПМИ-174

Никифоров Алексей Владимирович, гр. БПМИ-174

Руководитель КР:

Ляпина Светлана Юрьевна,

Главный аналитик Института менеджмента инноваций НИУ ВШЭ

Куратор:

_____

_____

Москва, 2019

Оглавление

Оглавление **2**

Аннотация **4**

Annotation **4**

Ключевые слова **4**

Введение **5**

Цель проекта 7

Обзор литературы и существующих решений **7**

Структура проекта **10**

Глава 1 – Подготовка данных **10**

Признаки 10

Обучающая выборка 11

Корректировка выборки 11

Данные о товарах и категориях 13

Методы получения данных и принцип их работы 13

Получение данных о товарных категориях и товарах 17

Глава 2 – Алгоритмы для подбора подарка **20**

Метрики 21

Модель. Простейшие классификаторы. 24

Baseline 24

Метод ближайших соседей (KNN) 24

Быстрый поиск соседей 26

Brute Force 26

Древовидные подходы 26

KD-Дерево 27

Ball Tree 27

Модель. Классификаторы. 28

Многоклассовая классификация 28

Случайный лес 29

Методы, основанные на градиентном бустинге 30

Нейронные сети 32

Многослойный персептрон 33
Результаты и выводы 35
Дополнительное обучение модели 36
Глава 3 – Интерфейс взаимодействия с пользователем и инфраструктура 37
Telegram-бот 39
Общий принцип работы 39
Сообщения и элементы взаимодействия с пользователем 44
База данных 47
Заключение 49
Список источников 52
Приложение 1 – Скрипт для получения перечня товарных категорий сервиса Яндекс.Маркет 57
Приложение 2 – Функция для получения заданного количества популярных товаров в категории 58
Приложение 3 – Функция для получения информации о товаре по ID 60

Аннотация

В современном мире люди привыкли на любые вопросы искать ответы в интернете. Интернет – это огромная мировая база знаний, и действительно – воспользоваться готовым, проверенным решением гораздо проще и эффективнее, чем проходить все этапы решения проблемы самому. Извечный вопрос «Что подарить?» не является исключением. Однако советы наподобие «Книга – лучший подарок» давно не актуальны. Пользователи хотят, чтобы подарок был оригинальным и учитывал интересы получателя подарка. Эту проблему могут решить рекомендательные системы, которые на основе информации о человеке могут порекомендовать более индивидуальный подарок.

Annotation

Nowadays, people are used to looking for answers to any questions on the Internet. The Internet is a huge global knowledge base, and it is much easier and more effective to use a ready-made, proven solution than to go through all the stages of solving the problem yourself. And the eternal question “what to give” is no exception. However, tips like “the book is the best present” seemed to be no longer relevant. People want the present to be original and consider the interests of the gift recipient. This problem can be solved by the recommendation systems that can recommend an individual gift based on some information about the gift recipient.

Ключевые слова

Рекомендательная система, машинное обучение, чат бот, дерево решений, градиентный бустинг, метод ближайших соседей, нейронная сеть, классификация с пересекающимися классами.

Введение

В рамках данного проекта была поставлена задача разработать сервис, с помощью которого пользователь мог бы быстро и удобно ввести описание человека, и получить подборку товаров из разных интернет-магазинов, которые с большой вероятностью являются оптимальным подарком для описанного человека. Поставленную задачу можно описать как разработку рекомендательной системы для подбора подарка.

В стандартном понимании задачи о рекомендательной системе, на вход подаются пользователи, у которых известна история взаимодействия с товарами, описание этих пользователей и, соответственно, сами товары, с различными характеристиками. В нашем случае задача немного иная - мы не знаем историю взаимодействия с подарками у человека, для которого делаем прогноз, поэтому беря за основу идеи рекомендательных систем, мы будем строить свой алгоритм для прогноза подарков.

Первые системы рекомендаций были основаны на коллаборативной фильтрации. Такая системы была применена впервые в видеомагнитофоне TiVo, который учитывал предпочтения пользователя при показе видео. При этом использовалась информация о поведении пользователей в прошлом – для каждого пользователя отслеживалось, что нравилось ему смотреть, и если находились люди со схожими интересами, ему предлагалось посмотреть то, что он не смотрел, но что понравилось похожим зрителям.

При коллаборативной фильтрации сходство пользователей определяется через сходство истории их поведения. В нашем же случае это не может быть использовано, так как каждый запрос будет от нового пользователя и нам нужно определять сходство пользователей другим способом. В данной работе

мы определяли сходство пользователей на основе близости векторов, описывающих 2 людей:

- Мы взяли принцип коллаборативной фильтрации, и для каждого нового пользователя определяли множество пользователей, похожих на нового пользователя. Затем выделяли товары, которые понравились этому множеству больше всего. Мы предположили, что для определения множества пользователей, больше всего похожих на нового пользователя, хорошо подходит метод ближайших соседей, и протестировали его.

В 2000-х годах начали разрабатываться контентные модели. В этом подходе смотрят на описание каждого товара, и сравнивают сходство описаний новых товаров с товарами, которые пользователь оценил в прошлом. Новые исследования в основном используют байесовскую классификацию, которая позволяет понять и оценить, почему пользователь предпочитает тот или иной контент. Здесь же опять встает проблема отсутствия какой-либо информации о пользователе в разрезе понравившихся ему товаров.

Также, можно пойти другим путем: так как у нас есть вектор, описывающий пользователя, можно предсказывать для каждого товара вероятность, что этот товар подходит для пользователя, то есть решать задачу классификации с пересекающимися классами. В ходе исследования мы пришли к выводу, что удобнее будет предсказывать не сами товары, а категории товаров. А затем показывать топ товаров из предсказанных категорий.

В нашей работе мы сравнили два описанных подхода к решению нашей задачи. Проверили, что оба варианта могут быть применены на практике и предоставили результаты сравнения.

В процессе написания сервиса, мы проанализировали различные интерфейсы взаимодействия с пользователем и пришли к выводу, что наиболее

удобным на старте проекта является чат-бот в мессенджере Telegram, так как он относительно просто программируется на языке Python и обладает необходимым спектром интерактивности для взаимодействия с пользователем.

Собор и хранение данные о пользователях и их взаимодействии с сервисом мы осуществляем в базе данных PostgreSQL. Это необходимо для совершенствования модели и повышения её точности, так как на данных, собранных в ходе работы, будет проходить дополнительное обучение модели.

На выходе от модели мы планируем получать наиболее релевантные категории товаров для подарка, опираясь на дерево товарных категорий сервиса Яндекс.Маркет [35], после чего для полученных категорий мы будем получать топ товаров по рейтингу и популярности, и сортировать их по рейтингу покупателей. Для получения этой информации также потребуется получить информацию от сервиса Яндекс.Маркет [35] – например, отправляя запросы к предоставляемому им API.

В качестве обратной связи мы планируем отслеживать переходы по ссылкам с товарами и оставленный пользователем рейтинг, а также то, пропустил ли пользователь страницу с этим товаром или выбрал что-то на ней.

Цель проекта

Создание алгоритма персонализированных рекомендаций подарков для пользователей, учитывая вводимое пользователем описание человека, контекста подарка и желательную стоимость товара.

Обзор литературы и существующих решений

На текущий момент в сети Интернет можно найти несколько сервисов для подбора подарка. Некоторые из них – это тематические страницы на сайтах крупных интернет-магазинов в предпраздничные периоды. Например, в декабре

перед новогодними праздниками и в феврале накануне гендерных праздников (23 февраля и 8 марта). Похожая рекомендательная система подарков описывается в статье Mobile Gift Recommendation Algorithm [8].

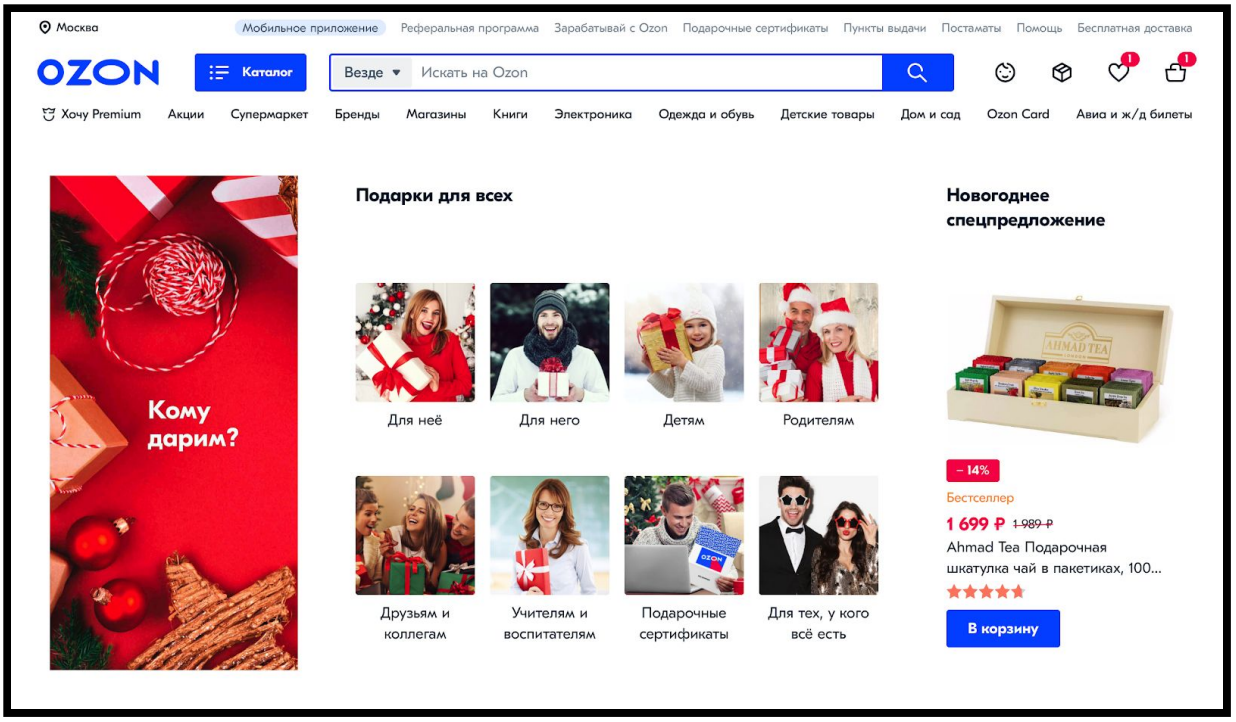

Иллюстрация 1 - Пример страницы подбора подарков на сайте интернет-магазина OZON.ru

Однако большинство из них представляют собой лишь фильтры по категориям, и зачастую просто вручную настраиваются маркетологами и категорийными менеджерами. Такой подход удобен и прост в реализации, однако требует работы человека для определения перечня возможных подарков для разных групп людей, и, в связи с этим – имеет низкую точность, в силу невозможности рассмотрения более мелких групп с более узкими интересами. Свою роль в релевантности предложенного подарка также играет и то, что такие сервисы предлагают только те товары, которые можно купить в данном интернет-магазине. В нашем проекте мы ставим перед собой цель избежать этих неточностей путем применения алгоритмов машинного обучения, а также – индексировать предложения с разных интернет-магазинов, чтобы охватить максимальный перечень возможных подарков.

Также существуют несколько работающих рекомендательных систем для подбора подарков, которые составляют рекомендации на основе анкеты с информацией о получателе и поводе для подарка, такие как delpodarki.ru[2] и bezpodarkov.net[3], однако эти сервисы также не индексируют предложения сторонних интернет-магазинов, и осуществляют подбор по усеченному списку параметров о получателе подарка.

Еще одним интересным решением является сервис Goody Bag[4], осуществляющий подбор подарка по информации о получателе с его страницы в социальной сети ВКонтакте.

В нашем проекте мы хотим совместить описанные выше подходы, разработав сервис, который на основе небольшой анкеты подбирал бы наиболее релевантный подарок, исходя из данных об интересах получателя, его социальной связи с дарителем, поводе для подарка и других.

Как было сказано ранее, для определения оптимального подарка мы решали задачу многоклассовой классификации с пересекающимися классами. Данная проблема широко исследуется в текущее время. Существует два подхода к решению этой проблемы - либо преобразуется задача и сводится к серии бинарных классификаторов, либо используется алгоритм, который позволяет решать задачу мультиклассовой классификации. Среди алгоритмов, которые могут решать исходную задачу без преобразования можно выделить метод ближайших соседей [33], бустинг (идея метода в объединении нескольких слабых моделей в одну сильную) [10], нейросети [9]. Классические подходы к преобразованию задачи делятся на 4 категории - Label Power-set (LP), Binary Relevance (one-versus-all), PairWise (all-versus-all) и Label Ranking [34]. В подходе Binary Relevance - задача преобразуется в классификацию отдельных бинарных классов, Label Ranking-подходы преобразуют задачу

классификации в задачу ранжирования всех имеющихся категорий по релевантности и значимости, а Label Power-set сводит классификацию с несколькими категориями к классификации с несколькими классами, рассматривая каждый отдельный набор категорий как уникальный класс.

Структура проекта

Работа над проектом была разделена на три основных блока:

1. Подготовка и проектирование структуры данных;
2. Разработка алгоритма подбора подарка;
3. Разработка инфраструктуры;

Глава 1 – Подготовка данных

Признаки

Мы изучили представленные на текущий момент решения, и используя их опыт выделили те данные, которые будут запрашиваться у пользователя. На основе этих данных мы будем обучать модель, которая будет подбирать оптимальный подарок. Это описание человека, которому предназначается подарок:

- Хобби – категориальный признак.
- Пол – категориальный признак.
- Возраст – Численный признак.

А также описание контекста подарка:

- Повод для подарка – категориальный признак.

Категориальные признаки мы закодировали One-Hot-Encoding (кодирование, при котором исходный категориальный столбец с n категориями заменяется на n столбцов, где каждый столбец отвечает за принадлежность объекта данной категории).

Обучающая выборка

Обучающая выборка была собрана из данных двух источников. Первый источник - опрос, который производился при помощи сервиса Google Forms. Опрос прошло 180 человек, однако не все анкеты подходили для использования в обучающей выборке, например, некоторые из участвующих в опросе указывали в качестве желаемого подарка путёвки, билеты на концерты, животных или какие-либо крайне специфические товары, которые не продаются в обычных интернет-магазинах. После отсеивания неподходящих анкет в выборке осталось 114 строк, две трети из них (76 строк) мы включили в обучающую выборку, а оставшуюся треть использовали в качестве эталонной (валидационной) выборки, на которой проверяли, насколько релевантны предсказания модели с человеческой точки зрения.

Второй источник данных - сервис “Mywishlist” [11]. Это сервис, где любой человек может составить список товаров, которые он бы хотел получить в подарок. Пользователи указывают свой возраст, пол, а каждый товар помечен тегом (категорией товара). С этого сервиса мы получили информацию дополнительно о 8500 парах “пользователь + товар”. Таким образом, в собранных данных 8576 строк и 476 уникальных категорий подарков.

Корректировка выборки

У собранных данных есть не все признаки, которые должны быть в обучающей выборке. Например, пользователи сервиса “Mywishlist” [11] не отмечают свои хобби и праздник, на который они бы хотели получить этот

товар. Поэтому мы самостоятельно для каждого товара разметили какие хобби могли быть у потребителя данного товара. Например для тега “Книги” мы отмечали, что у пользователя есть хобби “Чтение”, а для тега “Спорт” мы соответственно отмечали хобби “Спорт”.

Данные из опроса мы дублировали несколько раз с различным шумом. Мы добавляли людей с аналогичными интересами, а возраст генерировали случайно из нормального распределения с математическим ожиданием, равным возрасту человека из анкеты, и дисперсией равной 2. Также было обработано большое количество анкет, в которых люди указывали, что им не существенно, на какой именно праздник они хотят получить выбранный товар в качестве подарка. Соответственно, для них мы добавляли несколько записей с таким товаром, указывая несколько случайных праздников (учитывая при этом гендерность праздников: для мужчин не может быть добавлено 8 марта, а для женщин – 23 февраля). Аналогично мы добавляли праздники в выборку данных из сервиса “Mywishlist”[11] .

В имеющейся обучающей выборке каждому человеку соответствует ровно один подарок. Также в выборке есть множество строк с одинаковой информацией о человеке, но с разными категориями товара. Для каждого уникального клиентского портрета мы посчитали вероятность каждой из категорий (количество строк, где желаемым подарком для клиента, соответствующего данному портрету, была данная категория, разделим на общее количество строк, соответствующих данному клиентскому портрету), затем создали выборку *df_new*, где для каждого клиента мы по 10 раз сгенерировали категорию подарка в соответствии с найденным для данного портрета распределением, таким образом каждому клиенту будет соответствовать от 1 до 10 категорий подарка. Эта выборка содержит 16077

строк и 502 столбца, 26 из которых отвечают за характеристики клиента, а 476 за категории подарка. На этой выборке и будем обучать наши модели.

Данные о товарах и категориях

Методы получения данных и принцип их работы

В процессе реализации проекта перед нами встал вопрос – где можно взять перечень товаров, которые будет рекомендовать модель? Эта проблема – одна из главных сложностей в реализации проекта.

В качестве первого варианта рассматривалась возможность найти в открытом доступе в интернете базу товаров одного или нескольких крупных интернет-магазинов. Этот вариант наиболее простой, но имеет ряд существенных недостатков, из-за которых его использование становится нецелесообразным:

- Такая база товаров статична и не имеет возможности обновления. Это означает, что бот не сможет рекомендовать актуальные модели товаров, и его использование в скором времени станет бессмысленным;
- Перечень предлагаемых ботом товаров будет ограничен ассортиментом тех интернет-магазинов, чью базу удалось найти. Это означает, что многие нишевые области товаров не будут рекомендоваться, а именно они зачастую и являются идеальными подарками;
- Необходимость сопоставление товаров из обучающей выборки и товаров из найденной базы, что само по себе является достаточно сложной задачей.

Вторым возможным вариантом решения этой задачи является написание каталожного парсера – программы с помощью которой можно было бы один раз в заданный интервал времени проходить весь ассортимент нескольких крупных интернет-магазинов, и таким образом решать проблему актуальности

базы и проблему широты ассортимента товаров. Однако, такой вариант все равно не решает проблему с сопоставлением товаров, и даже усложняет её, ведь теперь у нас не одна база, а несколько, и необходимо дополнительно провести сопоставление товаров в них между собой.

Таким образом мы пришли к выводу, что рекомендовать конкретные товары, в общем случае, не имеет смысла. Такая модель без большого дополнительного объема данных быстро потеряет актуальность. На этом этапе мы перешли к гипотезе о том, что модель должна предсказывать не конкретные товары, а категории товаров, внутри которых опираясь на максимальную цену и другие ограничения мы могли бы выбрать лучшее предложение.

За основу было решено использовать данные о товарах и категориях сервиса Яндекс.Маркет [35] – крупного агрегатора товарных предложений из разных интернет магазинов. С использованием данного сервиса можно быстро найти практически любой товар, продающийся в русскоязычном сегменте интернета, а также ознакомиться с наиболее популярными товарами в своей категории, что идеально подходит для наших задач. Так как новые категории товаров появляются редко, то можно считать перечень категорий статичным. Однако внутри категорий регулярно появляются новые товары, и с помощью поиска популярных товаров в категории можно решить проблему с необходимостью актуализации перечня товаров.

Для получения необходимой нам информации можно использовать один из следующих подходов:

- Покупка лицензии на использование Контентного API Яндекс.Маркет [35]. Такой вариант является оптимальным для использования в нашем проекте, однако лицензия с необходимым лимитом на количество

запросов стоит дорого, и на согласование иных подробностей может уйти продолжительное время;

- Использование парсера также не подходит в нашем случае, так как требует дополнительной разработки сложного программного обеспечения и разворачивания дополнительной инфраструктуры;
- Вариант, на котором мы остановились при реализации нашего проекта – полученные данных с помощью запросов к API, найденных путем отслеживания трафика мобильного приложения с использованием уязвимости Man In The Middle (MITM).

Man In The Middle, или MITM (рус. “Человек посередине”) – вид атаки, при котором третье лицо тайно ретранслирует и при необходимости изменяет связь между двумя сторонами, которые считают, что они непосредственно общаются друг с другом. В нашем случае, мы использовали MITM для активного прослушивания трафика, передающегося между мобильным приложением и сервером. Для осуществления этого было использовано свободное ПО mitmproxy [22]. Сразу стоит отметить, что этот аспект нашей работы находится в серой зоне права, поэтому мы подчеркиваем, что в соответствии с пунктом 6.2 пользовательского соглашения сервисов Яндекса [23] мы используем контент сервиса Яндекс.Маркет [35] исключительно для личного некоммерческого использования с сохранением всех знаков охраны авторского права, смежных прав, товарных знаков, других уведомлений об авторстве, сохранения имени (или псевдонима) автора/наименования правообладателя в неизменном виде. Все данные, полученные в ходе выполнения данной курсовой работы, предназначены для личного некоммерческого использования и исключительно в целях демонстрации возможности существования подобного сервиса. Мы уважаем работу,

проделанную командой сервиса Яндекс.Маркет, и не стремимся ущемить их исключительное право на весь контент, предоставляемый сервисом.

Процесс прослушивания трафика с использованием программы mitmproxy происходит следующим образом:

1. На устройство, трафик которого планируется прослушивать, добавляется сертификат mitmproxy в список доверенных сертификатов. Таким образом, программа получает возможность устанавливать защищенное SSL-соединение;
2. После подключения устройства к proxy, созданному программой mitmproxy, пользователь программы получает возможность в режиме реального времени просматривать весь входящий и исходящий интернет-трафик с устройства:
    a. В случае использования незашифрованного HTTP-соединения процесс перехвата трафика происходит следующим образом: mitmproxy принимает соединение от клиента (например, от браузера на мобильном устройстве), сохраняет информацию о нем в памяти, после чего возвращает клиенту ответ от получателя запроса.
    b. В случае использования защищенного HTTPS-соединения. процедура перехвата трафика выглядит следующим образом:
        i. Клиент устанавливает соединение с mitmproxy;
        ii. Mitmproxy отправляет клиенту ответ с кодом 200 (соединение установлено);
        iii. Клиент взаимодействует с mitmproxy так же, как и с удаленным сервером, и устанавливает SSL-соединение;

iv. Mitmproxy подключается к серверу и устанавливает SSL-соединение, используя указанное клиентом имя хоста;

v. В ответе сервер передает SSL-сертификат, содержащий значения параметров CN и SAN, на основе которых затем будет создан подменный сертификат;

vi. Mitmproxy генерирует подменный сертификат и продолжает SSL-диалог с клиентом, приостановленный на этапе iii;

vii. Клиент отправляет запрос через установленное SSL-соединение;

viii. Mitmproxy передает запрос серверу через SSL-соединение, установленное на этапе iv;

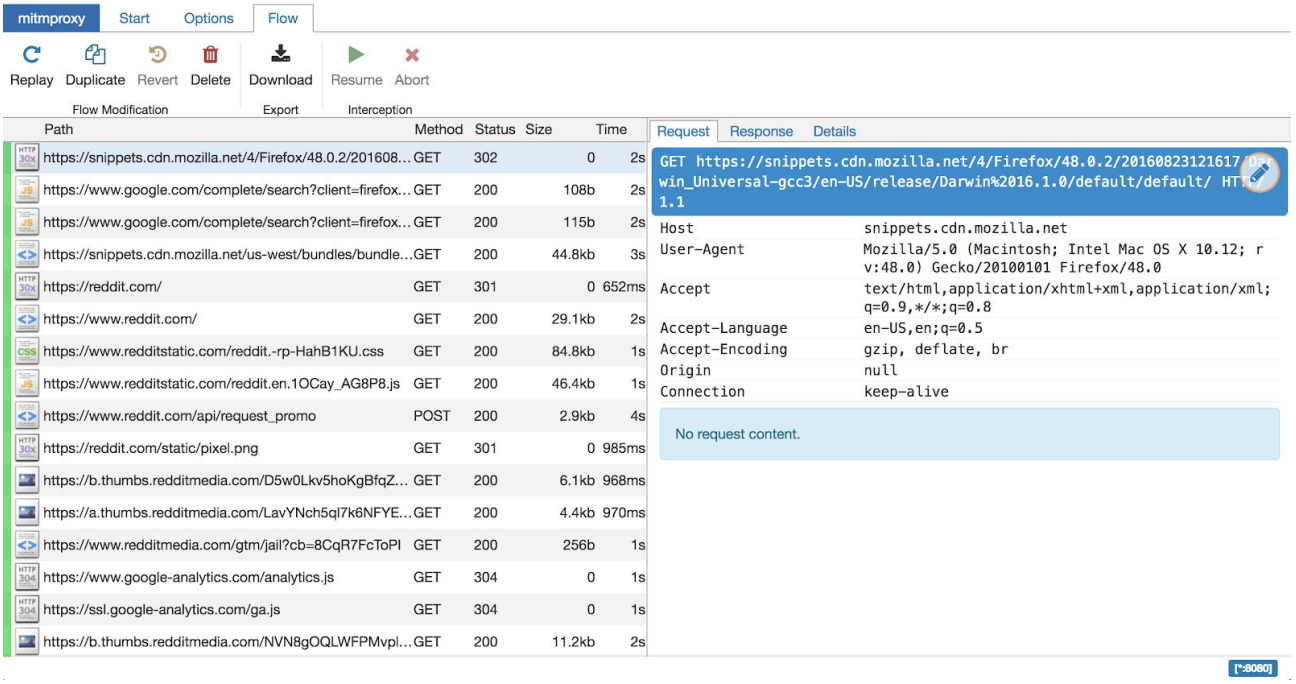

Иллюстрация 2 - Интерфейс программы MITMproxy

Получение данных о товарных категориях и товарах

Изначально, перечень категорий товаров сервиса Яндекс.Маркет [35] был обнаружен в открытом доступе на странице сервиса Яндекс.Справка [25],

посвященной отображению предложений магазинов на Яндекс.Маркете [35]. Однако, этот перечень оказался непригодным для использования в качестве таргета нашей модели по причине того, что в нем отсутствует необходимый для использования в последующих запросах уникальный идентификатор категории. Также, нам не удалось обнаружить запросы на получение полного списка категорий среди остальных запросов к серверу при просмотре трафика мобильного приложения Яндекс.Маркет [35]. Тогда была выдвинута гипотеза о том, что для получения полного списка категорий можно попробовать использовать вызов методов, аналогичных GET /{version}/categories [26] и GET /{version}/categories/{id}/children [27] из контентного API Яндекс.Маркета [35].

С помощью скриптов, указанных в приложении 1, был получен список корневых категорий, а впоследствии - и полное категорийное дерево.

Аналогичные действия были проделаны и для получения информации о популярных товарах в выбранной категории. Для этого была написана функция getTopInCategory, указанная в приложении 2. Ее принцип работы заключается в следующем:

- С помощью вызова POST-запроса, содержащего в payload информацию о категории товара, максимальную и минимальную цену, количество товаров к выводу и прочую техническую информацию, происходит получение JSON-файла, содержащего в структуре информацию о товарных предложениях, проранжированных по популярности;
- Из JSON-файла для каждого товарного предложения извлекается следующая информация:
    - ID – товарного предложения;
    - description – описание товара;

- offersCount – количество магазинов, предлагающих для покупки данных товар;
- reviewsCount – количество обзоров на данный товар;
- min_price – минимальная цена, предложенная магазинами на данный товар;
- max_price – максимальная цена, предложенная магазинами на данный товар;
- avg_price – средняя цена, предлагаемая магазинами на данный товар;
- rating – рейтинг товара на Яндекс.Маркет [35];
- ratingCount – количество отзывов покупателей, на основе которых составлен рейтинг товара;
- raw_title – название товара в формате, в котором оно указывается в поисковой выдаче;

Для получения информации, необходимой для вывода пользователю, была создана функция getDataByYandexID, указанная в приложении 3. Ее принцип работы заключается в следующем:

- С помощью запроса GET /{version}/models/{id} [27] происходит получение наименования товара, его описания, фотографии и ссылки на Яндекс.Маркет [35];
- С помощью запроса GET /{version}/models/{id}/offers [28] происходит получение о предложениях продавцов на товар, а в частности:
  - Цена минимального предложения;
  - Цена максимального предложения;
  - Количество предложений от различных продавцов;
  - Ссылка на сайт, предлагающий товар по минимальной цене;

- С помощью запроса GET /{version}/models/{id}/specification [30] происходит получение об основных характеристиках модели.

Глава 2 – Алгоритмы для подбора подарка

В результате работы алгоритма мы хотим получить *k* проранжированных категорий товаров, товары которых наиболее подходят в качестве подарка. В нашей реализации мы взяли *k=5*. Если пользователь захочет просмотреть больше чем 5 подарков, мы будем выдавать другие товары из тех же категорий.

В рамках данной работы мы сравнивали две группы моделей. Первый подход основан на коллаборативной фильтрации, когда новому человеку подбираются категории товаров на основе интересов похожих пользователей. Вторая группа моделей более сложные классификаторы. Для этих моделей мы сводим исходную проблему к задачи мультиклассовой классификации, где один класс – это одна категория товара, про которую необходимо решить, рекомендовать её пользователю или нет. Причём для работы сервиса нам потребуется использовать вероятности принадлежности объектов к каждому классу для того, чтобы рекомендовать несколько категорий, которые наиболее подходят в качестве подарка для данного человека. Иначе говоря, в данном случае решается задача многоклассовой политематической классификации (multilabel classification - мультиклассовая классификация либо классификация с пересекающимися классами).

Для обеих групп моделей нам необходимы метрики, по которым мы будем определять пригодность моделей.

Структура библиотек, которой мы придерживались при написании моделей, встраиваемых в наш сервис:

```
FILENAME = 'model1.sav'  ### файл, в который сохраняется модель
def learn_model1():
    #### функция, которая создает саму модель, обучает ее и сохраняет
    для дальнейшего использования в файл FILENAME

class model1():
    #### класс, делающий предсказание и возвращающий подарки
    def __init__(self, load_model, k = 5, n = 1):
        self.model = load_model


    def predict(self, X_test):
        ##### предсказывает вероятность для всех категорий
        return pred


    def get_categories(self, X_test):
        ##### возвращает  k предсказанных категорий
        return categories


    def get_gifts_(self, X_test, max_cost = 10000, min_cost = 0):
        ##### по предсказанным категориям получает подарки
        return gifts

##### функция, вызываемая из бота
def get_gifts(X_test, max_cost = 10000, min_cost = 0):
    loaded_model = pickle.load(open(FILENAME, 'rb'))
    model = model1(loaded_model)
    X_test = X_test[columns]
    return model.get_gifts_(X_test, max_cost, min_cost)
```

Иллюстрация 3 – Структура библиотек моделей

Метрики

В нашем случае главный фактор применимости модели будет релевантность ее предсказаний. Это довольно сложно определить стандартными математическими метриками, поэтому мы создали отдельную валидационную выборку, на которой мы смотрим что модель предлагает в качестве подарка разным людям. После того, как модель показала работоспособность на валидационной выборке, мы смотрели на метрики, вычисляемые автоматически.

У моделей мы будем отслеживать несколько метрик, которые будут влиять на выбор итоговой модели.

Модели в нашей задаче будут возвращать для каждого класса (каждой категории подарка) вероятность, что этот класс подходит для описанного человека. Рассмотрим основные метрики для классификации с пересекающимися классами:

Coverage error. Данная метрика позволяет понять, сколько категорий с самой большой предсказанной вероятностью мы должны включить в ответ, не пропустив ни одной основной истинной категории.

$$coverage(y, \hat{f}) = \frac{1}{n_{\text{samples}}} \sum_{i=0}^{n_{\text{samples}}-1} \max_{j:y_{ij}=1} \text{rank}_{ij}$$

Ranking Loss определяется как количество неправильно упорядоченных категорий подарков по отношению к количеству правильно упорядоченных категорий. Самое лучшее значение функции потери может быть равно нулю.

$$Ranking{-}Loss\left(y, \hat{f}\right) = \frac{1}{n_{samples}} * \sum_{i=0}^{n_{samples}-1} \frac{1}{\|y_i\|_0 * (n_{labels} - \|y_i\|_0)} \left|\left\{(k,l) : \hat{f_{ik}} \leq \hat{f_{il}}; y_{ik} = 1, y_{il} = 0\right\}\right|$$

Discounted Cumulative Gain (DCG) является метрикой измерения качества ранжирования. Он в основном используется в задачах поиска информации, таких как измерение эффективности алгоритма поисковой системы путем ранжирования отображаемых статей в соответствии с их релевантностью с точки зрения ключевого слова поиска.

$$\sum_{r=1}^{\min(K,M)} \frac{y_{f(r)}}{\log(1+r)}$$

Так как нам нужно уметь хорошо ранжировать категории только в топе (то есть те категории, у которых предсказанная вероятность максимальна), то

мы будем измерять *DCG@k*. Также, мы будем нормировать эту метрику по идеальному ранжированию – и такая метрика уже будет называется *nDCG@k*.

В качестве функции потерь мы используем кросс-энтропию: $logloss(a, y) = -ylog(a) - (1 - y)log(1 - a)$.

Отдельный класс метрик будет измерять прикладное качество нашей модели. Например, в нашей задаче необходимо, чтобы алгоритм выдавал несколько разнообразных категорий, которые максимально отличаются друг от друга – чтобы дать пользователю выбор, либо несколько идей для подарка. Для этого, при обучении модели, мы отслеживаем **метрику разнообразия категорий (diversity)** - насколько далеко выдаваемые категории находятся друг от друга в дереве категорий Yandex market. Максимальное значение этой метрики - 4, минимальное - 0.

$$diversity(x) = \frac{1}{l} \sum_{n=1}^{l} \frac{2}{(k+1) * k} \sum_{i,j=1}^{k} \rho\,(i,j)$$

Покрытие возможных пользователей. Для подсчета данной метрики мы вычисляли процент пользователей, которым модель может рекомендовать хотя бы одну категорию.

Покрытие каталога товаров. Так как мы предсказываем категории, то логично отслеживать данную метрику как процент категорий, которые мы рекомендуем пользователям, от всего количества имеющихся категорий.

Среднее количество товаров, которые мы рекомендуем пользователю. Также будем отслеживать, сколько в среднем модель может предложить категорий для пользователя.

Этот класс метрик мы вычисляли на валидационной выборке.

Модель. Простейшие классификаторы.

Baseline

Суть алгоритма состоит в следующем: мы выбираем людей, максимально похожих на нашего человека и смотрим на категории товаров, которые им понравилось. При этом максимально похожими на человека мы считаем людей, у которых возраст отличается от возраста человека не более чем на epsilon (мы подобрали *epsilon* равный 2). А так же у которых нет хобби, которых нет у человека, а количество их одинаковых хобби максимально.

Данное решение подбирает релевантные подарки для людей, судя по валидационной выборке. Плюс данного решения в том, что мы точно не будем рекомендовать товары, которые подходят под хобби, которого нет у человека (например, мы точно не будем рекомендовать “коньки”, если человек не любит спорт). Однако, метрика “полнота покрытия пользователей” = 0.52, из чего видно, что почти в половине случаев мы ничего не можем рекомендовать человеку. Возможно этот алгоритм будет применим, когда обучающая выборка расширится за счет обратной связи нашего сервиса.

Изменим подход к поиску людей, максимально похожих на нового человека.

Метод ближайших соседей (KNN)

Рассмотрим вектора выборки как точки в *M*-мерном пространстве. Для новой точки *x_pred* будем определять *k* ближайших категорий.

В качестве функции расстояния мы будем использовать Расстояние Минковского второго порядка.

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + \cdots + (p_n - q_n)^2} = \sqrt{\sum_{k=1}^{n} (p_k - q_k)^2}.$$

Для нормализации признаков использовалась минимакс-нормализация. Заметим, что все признаки, кроме возраста, были уже нормализованы, так как принимали дискретные значения 0 или 1. Такие атрибуты, как возраст или пол мы считали более важными признаками, соответственно увеличивая веса перед этими признаками в моделях кластеризации.

В нашем алгоритме есть гиперпараметр *d=100* - количество ближайших точек с *x_pred*, среди которых мы ищем *k* ближайших классов к нашему вектору.

Для каждой из категорий, попавшей в ближайшие 100 точек, мы считаем оценку близости:

$$Q_j = \sum_{i=1}^{n} \frac{1}{d(x, a_i)^2},$$

d(x, a) - расстояние от *x_pred* до точки, принадлежащей классу, который мы рассматриваем. И затем, выбирается *k* классов, у которых *Q* – больше.

В валидационной выборке находятся подарки, наиболее подходящие для указанных людей. Результаты алгоритма: метрика *diversity@k* - 3.24, покрытие каталога - 12%, полнота покрытия пользователей – 100 % (что логично, так как всегда найдутся пусть далекие, но сравнительно ближайшие соседи), что делает этот метод значительно более работоспособным, нежели baseline решение.

Время работы алгоритма на всей выборке составило 0.1 секунды для построения модели и 0.28 секунды для определения *k=5* выдаваемых категорий.

Быстрый поиск соседей

Так как необходимо, чтобы предсказание считалось как можно быстрее, мы рассмотрели алгоритмы быстрого поиска ближайших соседей. Методы быстрого поиска соседей разделяются на точные и приближенные. Среди

приближенных методов можно выделить Locality-sensitive Hashing (метод, основанный на хэш -функциях) и Navigable Small World (основанный на построении графа). Среди точных самыми популярными являются KD-деревья и Ball-деревья.

Далее мы рассмотрим подробнее точные методы.

Brute Force

Быстрое вычисление ближайших соседей - это активная область исследований в области машинного обучения. Наиболее наивная реализация поиска соседей включает в себя грубое вычисление расстояний между всеми парами точек в наборе данных: для *N* точек с *D* признаками сложность этого подхода $O(DN^2)$.

Поиск соседей с помощью Brute Force может быть очень эффективным для небольших выборок данных. Однако по мере того, как число точек растет, подход Brute Force быстро становится неосуществимым.

Древовидные подходы

Для устранения вычислительной неэффективности подхода Brute Force были изобретены различные древовидные структуры данных. В общем случае эти структуры пытаются уменьшить требуемое количество вычислений расстояния путем эффективного кодирования агрегированной информации о расстоянии для выборки. Основная идея заключается в том, что если точка *A* очень далека от точки *B*, а точка *B* очень близка к точке *C*, то мы знаем, что точки *A* и *C* очень далеки, без необходимости явно вычислять их расстояние. Таким образом, вычислительная стоимость поиска ближайших соседей может быть уменьшена до $O(DNlog(N))$. Это значительное быстрее поиска с помощью Brute Force для большого количества точек.

KD-Дерево

Ранним подходом к использованию этой совокупной информации была структура данных *K*-мерного дерева. KD-Дерево - это двоичная древовидная структура, которая рекурсивно разбивает пространство параметров вдоль осей данных, разделяя его на вложенные области, в которые записываются точки данных. KD-Дерево очень быстро для поиска соседей с низкой размерностью ($D < 20$), однако при больших значениях D оно становится неэффективным.

Ball Tree

Для устранения неэффективности деревьев KD в более высоких измерениях была разработана структура данных ball tree. Ball Tree (так же известные как метрические деревья) - это пространственная структура данных, разделяющая пространство для организации точек в многомерном пространстве. Дерево шаров получило свое название из-за того, что оно разбивает точки данных на набор гиперсфер, известных как "шары".

Таблица 5. Время работы точных алгоритмов с разными подходами к поиску соседей

Алгоритм	Время обучения	Время предсказания категории	Время предсказания товара	Сложность алгоритма
Brute Force	0.115 секунды	0.28 секунды	6.17 секунды	$O(DN^2)$
KD-Дерево	0.160 секунды	0.26 секунды	5.89 секунды	$O(DNlog(N))$
Ball Tree	0.12 секунды	0.24 секунды	5.66 секунды	$O(DNlog(N))$

Время предсказания товара складывается из времени предсказания категории и запроса к yandex market за топ n товаров в предсказанной категории.

Быстрее всего обучается алгоритм Brute Force, однако он дольше всего предсказывает товар. Так как нам не принципиально, сколько будет обучаться модель, так как она обучается заранее, а не во время запроса пользователя, мы будем использовать Ball Tree, который показал лучшее время на предсказании категории и товара.

Даже точные методы показывают высокую скорость работы (большая часть времени не зависит от алгоритма), так что мы не стали рассматривать приблизительные алгоритмы.

Модель. Классификаторы.

Многоклассовая классификация

Для многоклассовой классификации с пересекающимися классами есть два классических подхода к применению различных моделей – one-versus-all и all-versus-all. В первом варианте строится *k* бинарных классификаторов, каждый их которых предсказывает вероятность принадлежности объекта к определенному классу. Во втором способе для каждой пары классов подбирается классификатор, а в момент предсказания выбирается класс, получивший наибольшее число голосов среди этих классификаторов. В нашем случае количество категорий равно 476, то есть при использовании all-versus-all подхода будет необходимо обучить 113050 классификаторов. Это заметно скажется на времени предсказания, что является критическим фактором в нашей задаче, поэтому мы приняли решение использовать one-versus-all подход.

Для начала мы изучили как работают древовидные модели для решения задачи классификации с пересекающимися классами. Дерево решений – это древовидная модель, похожая на блок-схему: в узлах дерева записано условие, а ветви дерева - это результат “теста” в узле. Спускаясь от корня дерева к

листьям, объект классифицируется, а результат классификации записывается в листе. Интуитивно, можно предположить, что дерево решений может давать неплохие результаты в данной задаче, так как люди при выборе подарка в большинстве случаев руководствуются схожими размышлениями – если человеку, которому нужно подарить подарок, нравится футбол, то бутсы могут быть хорошим подарком, но при этом он может не любить читать – и тогда книги исключаются из рассмотрения.

На основе решающего дерева существует несколько методов, которые показывают лучшие результаты . Например, это ансамбли решающих деревьев – Random Forest (“случайный лес”), а также – градиентный бустинг. Random Forest строит несколько решающих деревьев и результат классификации выдает путем голосования по этим деревьям. Градиентный бустинг же идет немного другим путем, который будет описан ниже.

Для таких моделей как случайных лес или градиентный бустинг мы использовали one-versus-all подход, оборачивая классы моделей в OneVsRestClassifier из библиотеки sklearn.multiclass [36].

Случайный лес

Для реализации многоклассовой классификации с помощью случайного леса, мы взяли стандартную функцию из класса sklearn.ensemble.RandomForestClassifier. В качестве функции потерь были протестирован критерий Джини и кросс-энтропия. Также, мы перебрали такие гиперпараметры модели как *n_estimators* (количество деревьев) = [20, 50, 100, 200], *max_depth* (максимальная глубина деревьев) = [4, 10, 20, ‘None’]. Однако результаты не сильно отличались друг от друга, а скорее наоборот – сильно ухудшались, если устанавливать значение max_depth отличным от ‘None’.

Лучший результат случайного леса получился на *n_estimators* = 100, результаты представлены в таблице 1.

Таблица 1. Таблица метрик модели на основе случайного леса.

Кросс-энтропия	nDCG@k	coverage error	label ranking average precision score	label ranking loss
14.64	0.81	119.89	0.68	0.12
Разнообразие предсказаний	Среднее количество предсказаний на пользователя	Полнота покрытия каталога	Полнота покрытия пользователей	
2.06	2.2	0.66	1.0	

Методы, основанные на градиентном бустинге

Бустинг - один из наиболее мощных методов построения предсказательных моделей, он отлично подходит как для решения задач классификации, так и для решения задач регрессии. Идея метода заключается в поочередном обучении нескольких несложных моделей, где каждая последующая модель обучается минимизировать ошибку предыдущих. В итоге, полная модель является ансамблем базовых моделей. Наиболее эффективным на данный момент считается бустинг над решающими деревьями, где добавление нового дерева стремится уменьшить ошибку уже построенной модели. Главным недостатком этого метода является очень долгое обучение, однако в нашей задаче время обучения не столь важно, так как модель обучается заранее.

Для нашей задачи мы протестировали три реализации градиентного бустинга: xgboost.XGBClassifier [37], lightgbm.LGBMClassifier [38] и CatBoostClassifier [39]. К различиям реализаций относятся как различия в реализации самих деревьев, в способах градиентного спуска (способа

уменьшения ошибки ансамбля классификаторов) так и различные способы взаимодействия с категориальными признаками – XGBClassifier не принимает на вход незакодированные категориальные признаки, CatBoostClassifier же специализируется на категориальных признаках, автоматически находя закономерности между категориями.

Таблица 2. Таблица метрик модели на основе XGBoost

Кросс-энтропия	nDCG@k	coverage error	label ranking average precision score	label ranking loss
11.6	0.73	108.64	0.65	0.12
Разнообразие предсказаний	Среднее количество предсказаний на пользователя	Полнота покрытия каталога	Полнота покрытия пользователей	
1.83	1.83	0.38	0.93	

Время обучения модели – 7 минут.

Время предсказания – 0.73 секунды.

Таблица 3. Таблица метрик модели на основе LGBMClassifier

Кросс-энтропия	nDCG@k	coverage error	label ranking average precision score	label ranking loss
32.57	0.69	219.94	0.55	0.25
Разнообразие предсказаний	Среднее количество предсказаний на пользователя	Полнота покрытия каталога	Полнота покрытия пользователей	
2.4	2.45	0.55	0.96	

Время обучения модели – 1 минута 7 секунд.

Время предсказания подарка для одного человека – 1.5 секунд.

Таблица 4. Таблица метрик модели на основе CatBoost

Кросс-энтропия	nDCG@k	coverage error	label ranking average precision score	label ranking loss
12.91	0.8	112.76	0.67	0.11
Разнообразие предсказаний	Среднее количество предсказаний на пользователя	Полнота покрытия каталога	Полнота покрытия пользователей	
2.0	2.06	0.63	0.98	

Время обучения модели – примерно 1.5 часа.

Время предсказания подарка для одного человека – 0.39 секунд.

Наиболее релевантные из моделей градиентного бустинга на валидационной выборке предсказывает XGBoost. Это подтверждают и метрики, отвечающие за качество модели - Кросс-энтропия, coverage error и тд. Однако при этом страдают такие метрики как среднее количество предсказаний на пользователя, полнота покрытия каталога и разнообразие предсказаний.

Нейронные сети

Нейросети хорошо зарекомендовали себя в решении самых разных задач – от классификации до обработки текста и изображений. Однако результаты нейросетей не интерпретируемые, и зачастую, чтобы найти оптимальное решение приходится перебрать большое количество архитектур. При этом затрачивается время как на обучение нейросетей так и время на перебор возможных гиперпараметров.

Многослойный персептрон

Для решения задачи мультиклассовой классификации подходит простая архитектура нейросети – многослойный персептрон. Это архитектура сети

прямого распространения, которая использует функцию обратного распространения ошибки.

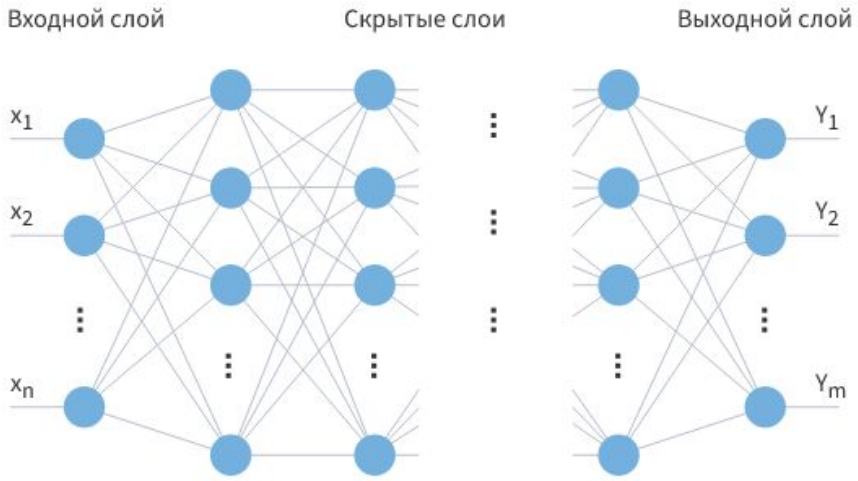

Иллюстрация 4 – Схема архитектуры многослойного персептрона

Для реализации многослойного персептрона мы использовали библиотеки tensordlow [6] и keras [7]. Мы перебрали несколько архитектур сети, постепенно добавляя Dense-слои (полносвязный слои, нейроны которого принимают на вход все выходы предыдущего слоя). Заметив, что в таком случае нейросеть очень быстро переобучается, мы добавили в комбинацию Dropout-слои . Это слои, случайно убирающие некоторые нейроны предыдущего слоя на разных эпохах обучения нейросети, уменьшая тем самым переобучение модели. Модель, показавшая лучшие результаты имеет четыре скрытых Dense слоя с Dropout слоем после каждых двух скрытых Dense слоев. В качестве функции активации мы использовали Leaky ReLU. В отличии от обычного ReLU эта функция не подвержена проблеме “смерти” нейронов (ситуация, при которой после обновления весов нейрон попадает в состояние, после которого он никогда не активируется и градиент, проходящий через данный нейрон, всегда будет равен нулю). Мы использовали функцию ошибки кросс энтропию, так как она дает наилучший результат, мы также пробовали использовать функцию

BP-MLL из библиотеки bpmll [40], но с такой функцией ошибки предсказания модели хуже по всем метрикам. Выходом нейросети стали n классов, и для того, чтобы получить вероятности каждого из класса, мы применяли не softmax, использующийся обычно для задач классификации, а брали сигмоиду от выхода для каждого класса. У полученной нейросети примерно 900 тысяч обучаемых параметров.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
digits (InputLayer)          (None, 26)                0
_________________________________________________________________
dense_1 (Dense)              (None, 75)                2025
_________________________________________________________________
dense_2 (Dense)              (None, 200)               15200
_________________________________________________________________
dropout_49 (Dropout)         (None, 200)               0
_________________________________________________________________
dense_3 (Dense)              (None, 500)               100500
_________________________________________________________________
dense_4 (Dense)              (None, 800)               400800
_________________________________________________________________
dropout_50 (Dropout)         (None, 800)               0
_________________________________________________________________
predictions (Dense)          (None, 476)               381276
=================================================================
Total params: 899,801
Trainable params: 899,801
Non-trainable params: 0
_________________________________________________________________
```

Иллюстрация 5 – Архитектура полученной нейросети

С помощью этой модели мы добились результатов, представленных в таблице ниже.

Таблица 2. Таблица метрик модели на основе многослойного персептрона

Кросс-энтропия	nDCG@k	coverage error	label ranking average precision score	label ranking loss

106.61	0.60	464.21	0.30	0.59
Разнообразие предсказаний	Среднее количество предсказаний на пользователя	Полнота покрытия каталога	Полнота покрытия пользователей	
3.0	5.0	0.42	1.0	

Значение кросс-энтропии, полученное при использовании описанной выше нейросети, оказалось приблизительно в 9 раз хуже, чем значение кросс-энтропии при использовании градиентного бустинга. Мы допускаем, что при более тщательной настройке гиперпараметров, значение кросс-энтропии при использовании нейросети можно приблизить к аналогичному показателю, рассчитанному для модели случайного леса, однако в рамках этой работы у нас не получилось достичь этого.

Результаты и выводы

В таблицах ниже отражен сравнительный анализ различных рассмотренных нами моделей.

Таблица 7. Сравнение времени работы алгоритмов

Алгоритм	Время обучения	Время предсказания
KNN	0.1 секунды	0.24 секунды
MLP	120 секунд	0.01 секунды
Случайный лес	28.1 секунды	0.92 секунды
Градиентный бустинг	7 минут	0.73 секунды

Таблица 8. Сравнение значений метрик на алгоритмах

Алгоритм	Полнота покрытия каталога	Полнота покрытия пользователей	Кросс-энтропия	Разнообразие предсказаний
KNN	12%	100%	-	3.24
Нейросеть	42%	100%	106.6	3.0

Случайный лес	4,6 %	100%	14.64	3.43
Градиентный бустинг	38%	93%	11.6	1.83

Лучшие результаты показали модели KNN и градиентный бустинг. Среди их отличий можно выделить время предсказания: градиентный бустинг предсказывает примерно в 3 раза дольше, чем метод ближайших соседей. Разнообразие категорий намного лучше у KNN модели, также с помощью данной модели мы покрываем всех возможных пользователей. Однако, градиентный бустинг выдает большее количество различных категорий, что видно из метрики “Полнота покрытия каталога”.

Реализации обеих моделей полностью готовы к использованию в боте. Какая модель будет работать лучше с точки зрения реальных пользователей можно будет сказать, проведя A/B-тестирование моделей [27]. Основной мы выбрали модель, основывающуюся на методе ближайших соседей за счет ее скорости и полного покрытия пользователей.

Дополнительное обучение модели

После запуска бота, мы планируем обновлять обучающую выборку с помощью данных, полученных ботом и дополнительно обучать построенную модель. Для этого мы добавили в функционал бота кнопки “посмотреть ссылку”, “получить новый подарок”. Будем считать, что человеку, который нажал на кнопку “получить новый подарок” подарок не подошел, а тому, кто нажал на кнопку “посмотреть ссылку” подошел.

Затем, мы будем добавлять в обучающую выборку людей и товары, понравившиеся им, и убирать постепенно людей, которых мы синтетически добавили в выборку. И каждые $N=100$ новых человек, будем заново обучать модель.

За счет этого планируется решить несколько проблем: на данный момент наша модель не учитывает социальный статус человека, которому дарится подарок, так как в собранных данных нет этого признака. Однако это планируется решить за счет новых поступающих данных из бота. Также, мы будем анализировать хобби, которых нет в модели, но которые вводят пользователи, и самые популярные добавлять в выборку. Такие доработки с течением времени должны постепенно исключить ошибки модели и расширить ее функционал.

Глава 3 – Интерфейс взаимодействия с пользователем и инфраструктура

Имея на руках модель, на этом этапе работы перед нами встала задача выбора платформы, или интерфейса взаимодействия с конечными пользователями. Саму по себе модель, так как она реализована на языке программирования Python, можно добавить к любому удобному интерфейсу взаимодействия с пользователем, будь то сайт, приложение или бот. Однако в рамках нашего учебного проекта мы остановили свой выбор на чат-боте в мессенджере Telegram. Такой выбор был сделан по ряду причин:

- Telegram – третий по популярности мессенджер в России в 2019 году по данным исследования компании Mediascope [12]. Первое и второе место в рейтинге занимают WhatsApp и Viber соответственно.
- В отличие от мессенджера WhatsApp, занимающего первое место по охвату аудитории в том же исследовании, Telegram предоставляет удобное API [14] для написания чат-ботов на большинстве популярных языков программирования. WhatsApp же официально продает инструменты для создания ботов только в качестве дорогостоящего продукта для корпоративных клиентов. Вдобавок к этому, боты в

WhatsApp обладают значительно меньшим функционалом, чем в Telegram. Viber – мессенджер, занимающий второе место по охвату аудитории в России, также с 2017 года предоставляет разработчикам инструменты для разработки ботов с похожим на Telegram функционалом [15], и готовый Telegram-бот может быть относительно легко адаптирован для работы с Viber.

- Telegram – мессенджер с молодой аудиторией. По данным исследования компании TGStat [13], больше всего в 2019 году в Telegram было представителей возрастных групп «25-34» (38% от общего числа пользователей) и «18-24» (27% от общего числа пользователей). В исследовании также указывается, что 60% аудитории Telegram имеют высшее образование, а 14% – находятся в процессе его получения. Все эти факторы, традиционно, указывают на бóльшую лояльность аудитории по отношению к новым технологиям и готовность пробовать новые сервисы.

Учитывая все указанные выше факты, мы решили использовать именно Telegram-бота в качестве интерфейса взаимодействия пользователя с моделью. Безусловно, у такого решения есть и свои минусы, главный из которых – продолжающаяся с октября 2017 блокировка Telegram Роскомнадзором. В первую очередь, это означает, что для обхода блокировки необходимо будет использовать proxy- или vpn-сервисы на стороне бота. Однако, это не должно значительно негативно сказаться на количестве пользователей сервиса – по данным уже упомянутого ранее исследования компании TGStat [13], около половины пользователей Telegram в России не испытывают проблем с использованием сервиса, 25.7% пользователей испытывают проблемы время от времени, и используют в таких случаях proxy и VPN, а остальные – успешно пользуются платными или бесплатными proxy- и VPN-сервисами.

Telegram-бот

Общий принцип работы

Для разработки Telegram-бота на языке программирования Python (мы выбрали именно его, так как все наши модели уже написаны на Python, и тогда подключить их для использования ботом не составляет труда) существует несколько подходов. Первый – это использовать REST API и HTTP-запросы, и реализовывать все необходимые методы, такие как отправка сообщений, обработка callback и другие, самостоятельно. Второй же вариант – это использовать уже готовый фреймворк-обработчик.

К плюсам первого варианта можно отнести более высокую отказоустойчивость, так как при его использовании мы не зависим от других сторонних разработчиков и можем более точно контролировать сценарии работы программы, а также – возможность использования всех инструментов актуальной версии API. Главный минус использования REST API – необходимость написания бóльшего количества кода и более сложный процесс разработки. Использование фреймворка, в свою очередь, позволяет использовать уже готовые методы и сильно упрощает процесс разработки. В нашем проекте мы остановились именно на этом варианте.

Готовых библиотек для разработки Telegram-ботов на языке программирования Python существует достаточно много. Выделим из них главным образом те, на которые ссылается в своих рекомендациях сам Telegram [16] – python-telegram-bot [17], pyTelegramBotAPI [18], AIOGram [19]. Эти проекты очень похожи между собой с силу того, что по сути представляют собой “обертку” одного и того же API. Python-telegram-bot – самый старый проект из приведенных, имеет подробную документацию и активно поддерживается сообществом программистов. AIOGram, в свою очередь, самый

молодой из перечисленных проектов. Особенностью этой библиотеки является то, что она написана на asyncio и aiohttp, что позволяет значительно повысить скорость и эффективность работы за счет использования асинхронных процессов. В нашем проекте мы будем использовать библиотеку telebot [20] вместе с pyTelegramBotAPI. Telebot – это небольшая библиотека, отличительной особенностью которой является легкость размещения проекта на облачном сервисе Google App Engine [21].

Заключительный принципиальный момент, с которым нам необходимо было определиться перед непосредственно созданием бота – какой тип связи с сервером будет использоваться: long polling или webhook.

При использовании подхода polling бот посылает запросы на получение обновлений на сервер, или же говоря неформально, как бы задает серверу вопрос: “Есть ли для меня новые сообщения?”. Такой подход считается неэффективным, ведь боту приходится несколько раз в секунду обращаться на сервер, и большую часть времени он будет получать от него неинформативный ответ из разряда: “Для вас пока нет новых сообщений”. Чтобы решить эту проблему и сэкономить ресурсы существует другой подход – long polling. Он работает аналогичным образом, за исключением того, что на стороне клиента устанавливается бóльший тайм аут (обычно – 30 или 60 секунд), и сервер отвечает только в двух случаях:

- Если появляется новое сообщение – отправляет ответ сразу, не дожидаясь тайм аута;
- Если за время тайм аута не появилось новых сообщений – отправляет ответ, что новых сообщений нет.

В противовес этим двум методам существует способ обращения webhook, который действует по обратному принципу. Теперь уже серверу известен адрес

клиента, и он отправляет ему запросы только в случае, когда это необходимо. Такой метод позволяет еще сильнее экономить ресурсы сети, исключая обмен незначащими сообщениями, а в случае с Telegram-ботом еще и позволяет значительно расширить возможности по асинхронной обработке сообщений нескольких клиентов одновременно. Также, как показывает практика, для ботов, которые работают постоянно на стороннем хостинге, невозможно использовать схему long polling запросов – после приблизительно 20 минут работы сервер возвращает 500 код ошибки, и для возобновления работы требуется перезапустить бота. Однако, при использовании webhook-запросов возникает потребность в дополнительной настройке инфраструктуры, в частности:

- Создание SSL сертификата для обмена сообщениями по защищенному HTTPS соединению;
- Статичный IP-адрес или доменное имя для устройства, на котором запущен бот;
- Дополнительная настройка правил firewall.

Учитывая все написанное выше, а также то, что при смене схемы обращения к серверу логика и остальные части бота остаются без изменений, для простоты тестирования нами был выбран вариант работы с long polling запросами.

С точки зрения логики работы, наш бот представляет собой своего рода “умную анкету”, в которую пользователь вносит необходимые данные, а в ответ получает информацию о подарке, который по собранной информации подобрала модель. Точкой входа является команда /start, после вызова которой бот начинает работу в соответствии со схемой, указанной на иллюстрации 2. После выбора действия “Выбрать подарок” пользователь в удобном ему

порядке по очереди вводит данные получателя подарка, и здесь же происходит проверка на корректность введенных данных, в частности:

- Пол – пользователю предлагается выбор из двух вариантов: “мужской” и “женский”. В случае, если введено иное, пользователю предлагается выбрать повторно;
- Возраст – пользователю предлагается ввести одно целое число. В случае, если введено иное, пользователю предлагается осуществить ввод повторно;
- Социальный статус – пользователю предлагается выбрать один из предложенных вариантов – “Знакомый”, “Друг”, “Коллега”, “Парень/Девушка” или “Родственник”. В случае, если введено иное, пользователю предлагается выбрать повторно;
- Хобби и увлечения – пользователю предлагается выбрать один или несколько из предложенных вариантов, а также ввести с клавиатуры один или несколько разделенных запятыми собственных вариантов, если необходимых хобби нет среди предложенных. Ввод заканчивается командой “Закончить”;
- Максимальная стоимость – пользователю предлагается ввести одно целое число. В случае, если введено иное, пользователю предлагается осуществить ввод повторно;
- Повод – пользователю предлагается выбрать один из предложенных вариантов – “День рождения“, “Новый год“, “23 февраля“, “8 марта“, “Годовщина отношений“, “День Святого Валентина“, “Другой повод“. В случае, если введено иное, пользователю предлагается выбрать повторно;

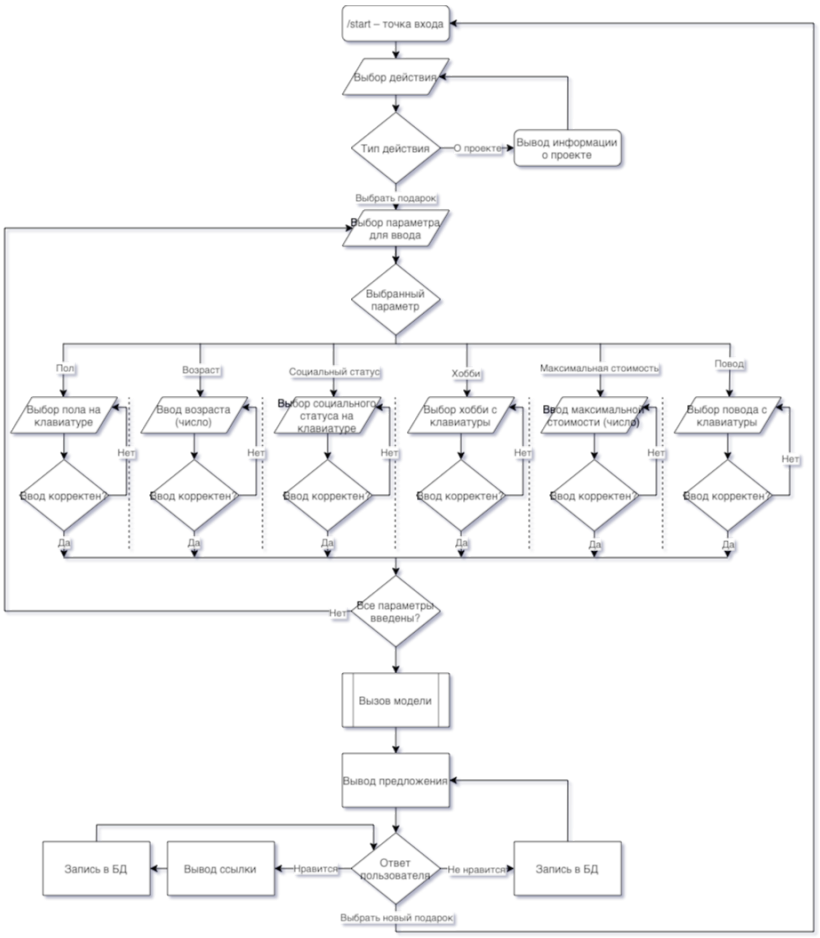

Иллюстрация 6 – Блок-схема чат-бота

Когда все необходимые данные собраны, происходит вызов функции get_gifts выбранной в настройках бота модели для получения товаров, которые будут предложены пользователю. С получением данных от модели, пользователю поочередно для всех выбранных моделью товаров выводятся название, описание и фотография каждого из них. На данном этапе пользователь может:

- перейти по ссылке на страницу товара на Яндекс.Маркет;
- перейти по ссылке на страницу магазина, предлагающего самую низкую цену на данный товар;
- перейти к следующему товару, выбранному моделью;
- выбрать новый подарок, заново введя данные получателя;

При этом, действия пользователя фиксируется и заносятся в базу данных с целью дальнейшего использования для обучения модели.

Сообщения и элементы взаимодействия с пользователем

Главными способами взаимодействия пользователя с ботом являются отправка текстовых сообщений, изображений, а также ответы с помощью ReplyKeyboardMarkup и InlineKeyboardMarkup.

ReplyKeyboardMarkup – это метод, благодаря которому пользователь видит в интерфейсе Telegram альтернативную клавиатуру с предложенными стандартными ответами (см. иллюстрацию 4). При нажатии на одну из кнопок от имени пользователя в чат происходит отправка сообщения с выбранным текстом. В нашем проекте ReplyKeyboardMarkup используется в стартовом меню, а также при выборе пола, хобби получателя подарка, типа социальной связи и повода для подарка.

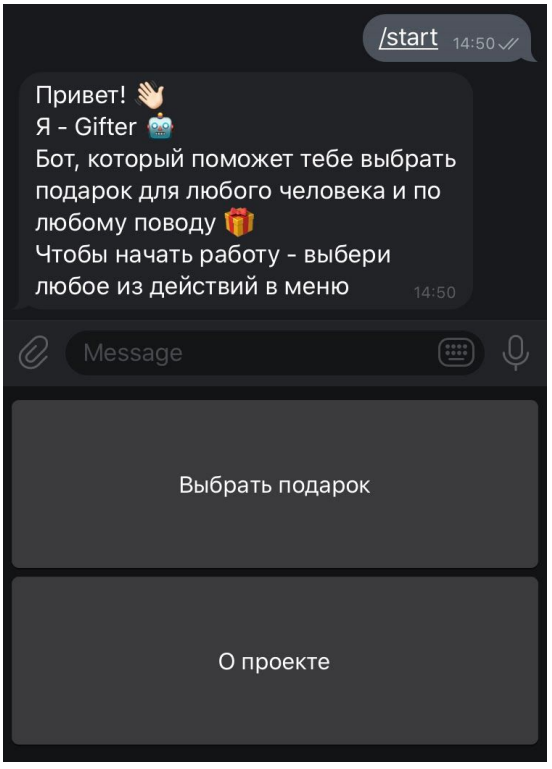

Иллюстрация 7 – Пример использования ReplyKeyboardMarkup в стартовом меню бота.

InlineKeyboardMarkup, в свою очередь – это метод, позволяющий закреплять интерактивные кнопки ниже отправленного сообщения. Сами кнопки могут быть следующих видов:

- Кнопки-ссылки, или URL-кнопки – при нажатии на нее пользователь переходит по ссылке, указанной как атрибут класса при создании кнопки в коде;
- Callback-кнопки позволяют взаимодействовать с пользователем по более сложным сценариям. При нажатии на такую кнопку боту отправляется объект типа CallbackQuery, содержащему поле data, в котором может быть:
    - записана некоторая строка, заложенная в кнопку на этапе создания;
    - объект Message, если сообщение отправлено ботом в обычном режиме;
    - поле inline_message_id, если сообщение отправлено в inline-режиме;
- Переключатели, или switch-кнопки. Такие кнопки в основном предназначены для обучения пользователей работе с ботом в inline-режиме. При нажатии на неё Telegram предложит выбрать чат, после чего подставит в поле ввода ник вашего бота и произвольный текст, если таковой был указан вами в аргументе switch_inline_query при создании кнопки.

В нашей работе в основном используются Callback-кнопки (см. пример на иллюстрации 8).

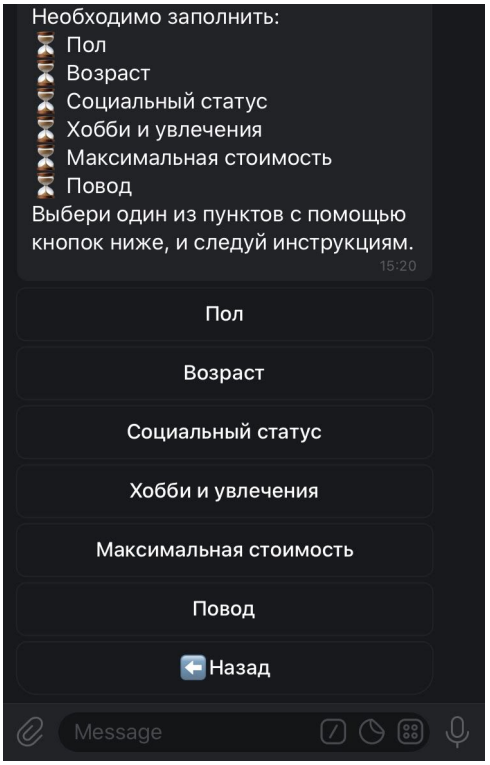

Иллюстрация 8 – Пример использования Callback-кнопок в меню выбора типа данных для ввода

Однако, нам не хватило стандартного функционала кнопок. При использовании стандартных URL-кнопок фиксировать факт перехода пользователя не предоставляется возможным. Для решения данной проблемы рассматривалось два варианта работы:

- Использование сервиса для сокращения ссылок, создание в каждом случае, был ли совершен переход. В нашем проекте мы предпочли не использовать такой вариант, так как он предполагает задействование дополнительного стороннего сервиса, что еще больше усложняет код и создает дополнительные риски в плане отказоустойчивости всей системы.
- Использовать Callback-кнопку, и при нажатии ей фиксировать факт заинтересованности пользователя в предложенном товаре, после чего редактировать сообщение, заменяя существующем InlineKeyboardMarkup Callback-кнопку на URL-кнопку. Мы остановились именно на данном варианте работы.

База данных

Для хранения данных о предпочтениях пользователей было принято решение использовать таблицу в базе данных PostgreSQL. Структура таблицы user_reactions, в которую происходит запись информации о действиях пользователя, выглядит следующим образом:

- ID – целое число, BIGINT – ID записи в таблице. В качестве значения по умолчанию используется максимальное значение ID, увеличенное на единицу;
- modifyDate – дата и время, TIMESTAMP – дата и время появления записи в таблице. В качестве значения текущая дата, полученная от функции *now* на момент записи в таблицу;
- telegram_id – целое число, NUMERIC – уникальный идентификатор пользователя Telegram;
- telegram_username – строка, TEXT – логин пользователя Telegram;
- telegram_firstname – строка, TEXT – имя пользователя Telegram;
- telegram_lastname – строка, TEXT – фамилия пользователя Telegram;
- telegram_language_code – строка, TEXT – код языка, используемого пользователем Telegram, согласно классификации IETF [8];
- recipient_sex – строка, TEXT – пол получателя подарка в формате M/F;
- recipient_age – целое число, INT – возраст получателя получателя подарка;
- recipient_status – строка, TEXT – тип социальной связи пользователя с получателем подарка;
- recipient_hobby – массив из строк, TEXT[] – список хобби получателя подарка;

- recipient_max_cost – целое число, INT – максимальная стоимость подарка;
- recipient_reason – строка, TEXT – повод для подарка;
- suggestion_model_id – число, NUMERIC – ID товара, предложенного моделью;
- suggestion_category_id – число, NUMERIC – ID категории, к которой относится предложенный моделью товар;
- user_reaction – строка, TEXT – действие, совершенное пользователем. Содержит одно из трех значений:
    - “next_item” – пользователь запросил отображение другого товара;
    - “goto_market” – пользователь перешел по ссылке на Яндекс.Маркет [35];
    - “goto_store” – пользователь перешел по ссылке на страницу интернет-магазина, предлагающего данный товар по самой низкой цене.

Пример данных, которые хранятся в таблице, можно посмотреть, выполнив запрос: select * from chatbot.dbo.user_reactions limit 10; Для записи данных в таблицу ботом используется библиотека sqlalchemy [41]. С помощью функции get_insert_command происходит получение необходимого запроса в формате строки, после чего с помощью функции execute_sql_safe модуля sql_utils осуществляется выполнение запроса.

Безусловно, такая структура данных не является самой оптимальной. Более того – база данных даже не приведена к первой нормальной форме (1НФ) [32], так как в ячейках столбца recipient_hobby хранятся списки значений, из-за чего нарушается условие атомарности, необходимое для приведения БД к 1НФ.

Однако для выполнения задач, поставленных перед учебным проектом, этого достаточно.

Дальнейшее усовершенствование структуры БД – одно из наиболее перспективных направлений работы по доработке проекта. В дальнейшем, при использовании сервиса подбора подарков большим количеством людей, можно будет собирать ценную аналитику о поведении и предпочтениях пользователей.

Заключение

В рамках данной работы перед нами была поставлена задача по разработке сервиса, позволяющего пользователю подобрать оптимальный подарок для любого человека.

Мы проделали работу, включающую в себя:

1) Анализ необходимых данных для построения предсказательной модели. Анализ и поиск источников данных, сбор информации из различных источников. Преобразование данных в нужный формат и синтетическое дублирование данных.

2) Исследование различных методов машинного обучения, подходящих под данную задачу. Сравнение работы моделей по различным метрикам и выбор наилучшего. Была разработана тактика дальнейшего улучшения качества моделей, и тактика выбора наилучшей модели из отобранных и показавших свою работоспособность.

3) Разработан сервис на основе Telegram бота, позволяющий максимально комфортно для пользователя ввести данные о человеке, которому необходимо подобрать подарок, и о стоимости подарка, а затем просмотреть возможные варианты подарка, а также при желании сразу приобрести понравившийся.

Работа была доведена до готового результата, при запуске бота в постоянном режиме были продуманы детали как со стороны удобства пользователей с самим сервисом, улучшение обучающей выборки с помощью данных, поступающих от самих пользователей так и улучшение моделей за счет обновления данных. А также выбор модели по “неявным” данным взаимодействия пользователя с сервисом.

К дальнейшим путям развития работы можно отнести более тщательное исследование моделей в части нейронных сетей, исследование юридических тонкостей работы с данными и развитие проекта в сторону увеличения количества информации, собираемой от пользователей, для дальнейшей более глубокой аналитики.

Все материалы проекта, исходные коды с комментариями и прочую информацию можно найти в репозитории проекта на GitHub [31].

Список источников

[1] Ca´ıque de Paula Pereira, Ruyther Parente da Costa and Edna Dias Canedo Mobile Gift Recommendation Algorithm // Faculdade UnB Gama, Universidade de Bras´ılia, Gama, DF, Brazil;

[2] Сервис онлайн подбора подарков по параметрам — на новый год, день рождения и другие праздники в Москве // «Деловые подарки» URL: https://www.delpodarki.ru/podbor/ (дата обращения: 25.05.2020);

[3] Подобрать подарок - Сервис подбора подарков «Сироп» // Сервис подбора подарков «Сироп» URL: https://bezpodarkov.net/find (дата обращения: 25.05.2020);

[4] Goody Bag - сервис подбора подарков с искусственным интеллектом // Goody Bag URL: https://goodybag.ru/#!/predictor (дата обращения: 25.05.2020);

[5] Rokach, L., Maimon, O. Data mining with decision trees: theory and applications. World Scientific Pub Co, 2008. 305 c. ;

[6] TensorFlow - An end-to-end open source machine learning platform // TensorFlow URL: https://www.tensorflow.org/ (дата обращения: 25.05.2020);

[7] Keras: the Python deep learning API // Keras Official Website URL: https://keras.io/ (дата обращения: 25.05.2020);

[8] Tags for Identifying Languages // IETF | Internet Engineering Task Force URL: https://tools.ietf.org/rfc/bcp/bcp47.txt (дата обращения: 25.05.2020);

[9] Min-Ling Zhang and Zhi-Hua Zhou, Senior Member // Multi-Label Neural Networks with Applications to Functional Genomics and Text Categorization, 2006;

[10] Schapire, R. E. and Singer, Y. // Boostexter: a boostingbased system for text categorization. Machine Learning, 39(2/3):135–168, 2000;

[11] WISHLIST.RU: мой вишлист // WISHLIST.RU URL: http://mywishlist.ru/ (дата обращения: 25.05.2020);

[12] За год блокировки Telegram остался в тройке самых популярных мессенджеров // РБК URL: https://www.rbc.ru/technology_and_media/13/04/2019/5cb19f339a794741a319f84d (дата обращения: 25.05.2020);

[13] Исследование аудитории Telegram 2019. Telegram Analytics // Telegram Analytics URL: https://tgstat.ru/research (дата обращения: 25.05.2020);

[14] Telegram Bot API // Telegram URL: https://core.telegram.org/bots/api (дата обращения: 25.05.2020);

[15] Releases | Viber Developers Hub // Viber URL: https://developers.viber.com/releases/ (дата обращения: 25.05.2020);

[16] Bot Code Examples // Telegram URL: https://core.telegram.org/bots/samples (дата обращения: 25.05.2020);

[17] Telegram Bot API // Telegram URL: https://core.telegram.org/bots/samples (дата обращения: 25.05.2020);

[18] eternnoir/pyTelegramBotAPI: Python Telegram bot api. // GitHub URL: https://github.com/eternnoir/pyTelegramBotAPI (дата обращения: 25.05.2020);

[19] aiogram/aiogram: Is a pretty simple and fully asynchronous framework for Telegram Bot API written in Python 3.7 with asyncio and aiohttp. // GitHub URL: https://github.com/aiogram/aiogram (дата обращения: 25.05.2020);

[20] yukuku/telebot: Telegram Bot starter kit. Very easy to install with Google App Engine. // GitHub URL: https://github.com/yukuku/telebot (дата обращения: 25.05.2020);

[21] App Engine | Google Cloud // Google Cloud URL: https://cloud.google.com/appengine (дата обращения: 25.05.2020);

[22] mitmproxy - an interactive HTTPS proxy // MITMPROXY URL: https://mitmproxy.org (дата обращения: 25.05.2020);

[23] Пользовательское соглашение сервисов Яндекса - Правовые документы // Яндекс URL: https://yandex.ru/legal/rules/ (дата обращения: 25.05.2020);

[24] Как попасть в правильную категорию - Маркет для магазинов // Яндекс.Справка URL: https://yandex.ru/support/partnermarket/guides/classification.html (дата обращения: 25.05.2020);

[25] Список категорий - Технологии Яндекса // Яндекс.Технологии URL: https://yandex.ru/dev/market/content-data/doc/dg-v2/reference/category-controller-v2-get-root-categories-docpage/ (дата обращения: 25.05.2020);

[26] Список подкатегорий - Технологии Яндекса // Яндекс.Технологии URL: https://yandex.ru/dev/market/content-data/doc/dg-v2/reference/category-controller-v2-get-children-categories-docpage/ (дата обращения: 25.05.2020);

[27] A/B тест — это просто // Хабр URL: https://habr.com/ru/post/233911/ (дата обращения: 25.05.2020);

[28] Информация о модели - Технологии Яндекса // Яндекс.Технологии URL: https://yandex.ru/dev/market/content-data/doc/dg-v2/reference/models-controller-v2-get-model-docpage/ (дата обращения: 25.05.2020);

[29] Список предложений на модель - Технологии Яндекса // Яндекс.Технологии URL: https://yandex.ru/dev/market/content-data/doc/dg-v2/reference/models-controller-v2-get-offers-docpage/ (дата обращения: 25.05.2020);

[30] Модели товаров - Технологии Яндекса // Яндекс.Технологии URL: https://yandex.ru/dev/market/content-data/doc/dg-v2/reference/models-controller-v2-docpage/ (дата обращения: 25.05.2020);

[31] mgcrp/hse_chatbot_2020: Курсовой проект "Чат бот рекомендует" // GitHub URL: https://github.com/mgcrp/hse_chatbot_2020 (дата обращения: 25.05.2020);

[32] К. Дж. Дейт. Введение в системы баз данных = Introduction to Database Systems. — 8-е изд. — М.: Вильямс, 2006. — С. 1328. — ISBN 5-8459-0788-8;

[33] Zhang, M. and Zhou, Z. // ML-kNN: A lazy learning approach to multi-label learning. Pattern Recognition, 40(7):2038–2048, 2007;

[34] Farbound Tai and Hsuan-Tien Lin // Multi-label Classification with Principle Label Space Transformation;

[35] Яндекс.Маркет — выбор и покупка товаров из проверенных интернет-магазинов // Яндекс.Маркет URL: https://market.yandex.ru (дата обращения: 25.05.2020);

[36] 1.12. Multiclass and multilabel algorithms — scikit-learn 0.23.1 documentation // scikit-learn: machine learning in Python URL: https://scikit-learn.org/stable/modules/multiclass.html (дата обращения: 25.05.2020);

[37] Python API Reference — xgboost 1.1.0-SNAPSHOT documentation // XGBoost Documentation URL: https://xgboost.readthedocs.io/en/latest/python/python_api.html (дата обращения: 25.05.2020);

[38] lightgbm.LGBMClassifier — LightGBM 2.3.2 documentation // LightGBM Documentation URL: https://lightgbm.readthedocs.io/en/latest/pythonapi/lightgbm.LGBMClassifier.html (дата обращения: 25.05.2020);

[39] CatBoostClassifier - CatBoost. Documentation // CatBoost - open-source gradient boosting library URL:

https://catboost.ai/docs/concepts/python-reference_catboostclassifier.html (дата обращения: 25.05.2020);

[40] bpmll · PyPI // Implementation of the BP-MLL loss function in Tensorflow URL: https://pypi.org/project/bpmll/ (дата обращения: 25.05.2020);

[41] SQLAlchemy - The Database Toolkit for Python // SQLAlchemy URL: https://www.sqlalchemy.org (дата обращения: 25.05.2020);

Приложение 1 – Скрипт для получения перечня товарных категорий сервиса Яндекс.Маркет

```
import json
import uuid
import numpy as np
import pandas as pd
import requests as rq

from pandas.io.json import json_normalize

df = pd.DataFrame(columns=['id', 'name', 'fullName', 'link', 'level', 'parent', 'childCount'])

for index, row in json_normalize(response.json()['categories']).iterrows():
    df = df.append(
        pd.Series(
            {
                'id':row.id,
                'name':row['name'],
                'fullName':row.fullName,
                'link':row.link,
                'level':0,
                'childCount':row.childCount
            }
        ), ignore_index=True
    )

for i in range(1, 5):
    for index, row in df[(df.level == i) & (df.childCount > 0)].iterrows():
        tmp_response = rq.get(
            f'https://mobile.market.yandex.net/market/white/v2.1.5/categories/{row.id}/children',
            params={'uuid':tmp_uuid, 'count':30},
            headers=headers
        )

        for index2, row2 in json_normalize(tmp_response.json()['categories']).iterrows():
            df = df.append(
                pd.Series(
                    {
                        'id':row2.id,
                        'name':row2['name'],
                        'fullName':row2.fullName,
                        'link':row2.link,
                        'level':i+1,
                        'childCount':row2.childCount,
                        'parent':row.id
                    }
                ), ignore_index=True
            )

        cur_page = 1
        max_page = tmp_response.json()['context']['page']['total']
        while cur_page < max_page:
            cur_page += 1
            tmp_response = rq.get(
                f'https://mobile.market.yandex.net/market/white/v2.1.5/categories/{row.id}/children',
                params={'uuid':tmp_uuid, 'count':30, 'page':cur_page},
                headers=headers
            )

            for index2, row2 in json_normalize(tmp_response.json()['categories']).iterrows():
                df = df.append(
                    pd.Series(
                        {
                            'id':row2.id,
                            'name':row2['name'],
                            'fullName':row2.fullName,
                            'link':row2.link,
                            'level':i+1,
                            'childCount':row2.childCount,
                            'parent':row.id
                        }
                    ), ignore_index=True
                )
```

Приложение 2 – Функция для получения заданного количества популярных товаров в категории

```
def getTopInCategory(n, category_id, max_price, min_price=0, page=1):
    """
    Возвращает `n` первых товаров в категории `category_id` заданым
    ограничением по цене (сортировка по популярности на Яндекс.Маркет)

    Параметры:
        n          - int - количество товаров
        category_id - int - ID категории товара
        max_price   - int - максимальная цена товара
        min_price   - int - минимальная цена товара
        page       - int - номер страницы в выдаче (пока что оставил как технический параметр)
    """

    _headers = {
        "Host" : "ipa.touch.market.yandex.ru",
        "X-Test-Id" : "",
        "Connection" : "keep-alive",
        "Accept" : "*/*",
        "Accept-Language" : "en-us",
        "Accept-Encoding" : "gzip, deflate, br",
        "Content-Type" : "application/json",
        "Api-Platform" : "IOS",
        "User-Agent" : "WhiteMarket/600.19.1 (ru.yandex.ymarket; build:1172; iOS 13.3.1; iPhone)",
        "X-Platform" : "IOS",
        "Content-Length" : "314",
        "X-App-Version" : "600.19.1",
        "X-Device-Type" : "SMARTPHONE"
    }

    _uuid = uuid.uuid4().hex

    _payload = json.dumps(
        {
            "params": [
                {
                    "filters": {
                        "allow-collapsing": "1",
                        "glfilter": [],
                        "grhow": "shop",
                        "onstock": "1",
                        "pricefrom": min_price,
                        "priceto": max_price,
                        "show-cutprice": "0"
                    },
                    "hid": category_id,
                    "local-offers-first": 0,
                    "numdoc": n,
                    "page": page,
                    "pp": "SEARCH",
                    "rearr-factors": "search_offline_offers=1",
                    "show-shops": "all",
                    "show-vendors": "all",
                    "text": "",
                    "use-default-offers": "1"
                }
            ]
        }
    )

    _response = rq.post(
        "https://ipa.touch.market.yandex.ru/api/v2",
        params={
            "name" : "resolveSearch",
            "rearr_factors" : "market_disable_parametric_search_for_white_except_parametric_specification=0",
            "sections" : "MEDICINE",
            "uuid" : tmp_uuid
        },
        headers=_headers,
        data=_payload
    )

    _goods = pd.DataFrame(columns=[
        'id', 'raw_title', 'description',
        'offersCount', 'reviewsCount', 'min_price',
        'max_price', 'avg_price', 'rating', 'ratingCount'
    ])
```

```
for item in _response.json()['collections']['product']:
    _goods = _goods.append(
        pd.Series(
            {
                'id'           : item['id']            if 'id' in item.keys() else pd.np.nan,
                'description'  : item['description']   if 'description' in item.keys() else pd.np.nan,
                'offersCount'  : item['offersCount']   if 'offersCount' in item.keys() else pd.np.nan,
                'reviewsCount' : item['reviewsCount']  if 'reviewsCount' in item.keys() else pd.np.nan,
                'min_price'    : item['prices']['min'] if 'prices' in item.keys() else pd.np.nan,
                'max_price'    : item['prices']['max'] if 'prices' in item.keys() else pd.np.nan,
                'avg_price'    : item['prices']['avg'] if 'prices' in item.keys() else pd.np.nan,
                'rating'       : item['rating']        if 'rating' in item.keys() else pd.np.nan,
                'ratingCount'  : item['ratingCount']   if 'ratingCount' in item.keys() else pd.np.nan,
                'raw_title'    : item['titles']['raw'] if 'titles' in item.keys() else pd.np.nan
            }
        ), ignore_index=True
    )

return _goods
```

Приложение 3 – Функция для получения информации о товаре по ID

```
def getDataByYandexID(yandex_id):
    output = {}

    # Part 1 - get basic information, such as name and description

    _uuid = uuid.uuid4().hex
    _baseUrl = f'https://mobile.market.yandex.net/market/white/v2.1.5/models/{yandex_id}'
    _params = {
        'count':'10',
'fields':'SHOP_RATING,OFFER_OFFERS_LINK,OFFER_SHOP,OFFER_DELIVERY,OFFER_DISCOUNT,FILTERS,FILTER_FOUND,FILTER_PHOTO_PICKER,FILTER_SORTS,OFFER_AC
TIVE_FILTERS,PHOTO',
        'groupBy':'SHOP',
        'local_offers_first':'0',
        'onstock':'1',
        'page':'1',
        'pp':'531',
        'sort':'RELEVANCY',
        'rearr_factors':'market_disable_parametric_search_for_white_except_parametric_specification=0',
        'sections':'MEDICINE',
        'uuid':_uuid
    }
    _headers = {
        'Host':       'mobile.market.yandex.net',
        'Accept':     '*/*',
        'Accept-Language':       'en-us',
        'Accept-Encoding':       'gzip, deflate, br',
        'X-Test-Id':  '',
        'Api-Platform':          'IOS',
        'User-Agent': 'WhiteMarket/600.19.1 (ru.yandex.ymarket; build:1172; iOS 13.3.1; iPhone)',
        'X-Platform': 'IOS',
        'X-App-Version':         '600.19.1',
        'Connection': 'keep-alive',
        'X-Device-Type':         'SMARTPHONE'
    }

    _request = rq.get(
        url=_baseUrl,
        headers=_headers,
        params=_params
    )

    output['name'] = _request.json()['model']['name']
    output['description'] = _request.json()['model']['description']
    output['market_link'] = _request.json()['model']['link']
    output['photo'] = _request.json()['model']['photo']['url']

    # Part 2 - get offers

    _params = {
        'count':'10',
'fields':'SHOP_RATING,OFFER_OFFERS_LINK,OFFER_SHOP,OFFER_DELIVERY,OFFER_DISCOUNT,FILTERS,FILTER_FOUND,FILTER_PHOTO_PICKER,FILTER_SORTS,OFFER_AC
TIVE_FILTERS',
        'groupBy':'SHOP',
        'local_offers_first':'0',
        'onstock':'1',
        'page':'1',
        'pp':'531',
        'sort':'RELEVANCY',
        'rearr_factors':'market_disable_parametric_search_for_white_except_parametric_specification=0',
        'sections':'MEDICINE',
        'uuid':_uuid
    }

    _offers = []
    _baseUrl = f'https://mobile.market.yandex.net/market/white/v2.1.5/models/{yandex_id}/offers'

    _response = rq.get(
        url=_baseUrl,
        headers=_headers,
        params=_params
    )

    _offers.extend(_response.json()['offers'])
    _curPage = 2
    _maxPage = _response.json()['context']['page']['total']
```

```
while _curPage < _maxPage:
    _params['page'] = _curPage
    print(_params)
    _response = rq.get(
        url=_baseUrl,
        headers=_headers,
        params=_params
    )
    _offers.extend(_response.json()['offers'])
    _curPage += 1

_prices = [float(offer['price']['value']) for offer in _offers]

output['max_price'] = max(_prices)
output['min_price'] = min(_prices)
output['count_offers'] = len(_offers)
output['best_offer'] = _offers[np.argmin(_prices)]

# Part 3 - get specifications

_baseUrl = f'https://mobile.market.yandex.net/market/white/v2.1.5/models/{yandex_id}/specification'
_params = {
    'rearr_factors': 'market_disable_parametric_search_for_white_except_parametric_specification=0',
    'sections': 'MEDICINE',
    'uuid': _uuid
}

_response = rq.get(
    url=_baseUrl,
    headers=_headers,
    params=_params
)
output['specification'] = _response.json()['specification'][0]

return output
```