МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ФАКУЛЬТЕТ ПРИКЛАДНОЙ МАТЕМАТИКИ И ИНФОРМАТИКИ

КАФЕДРА ДИСКРЕТНОЙ МАТЕМАТИКИ И АЛГОРИТМИКИ

ГАНКОВИЧ Святослав Сергеевич

РАЗРАБОТКА И РЕАЛИЗАЦИЯ МЕТОДОВ ОБРАБОТКИ ЕСТЕСТВЕННЫХ ЯЗЫКОВ НА ОСНОВЕ МЕТОДОВ МАШИННОГО ОБУЧЕНИЯ

Магистерская диссертация

специальность 1-31 81 09 «Алгоритмы и системы обработки больших объемов информации»

Научный руководитель
Свирид Юрий Владимирович
кандидат технических наук,
доцент кафедры биомедицинской
информатики ФПМИ

Допущена к защите

«___» _____ 2020 г.

Зав. кафедрой дискретной математики и алгоритмики
_____ В.М. Котов
доктор физико-математических наук, профессор

Минск, 2020

СОДЕРЖАНИЕ

ОБЩАЯ ХАРАКТЕРИСТИКА РАБОТЫ

Магистерская диссертация, 54 с., 18 рис., 5 табл., 11 источников**.**

Ключевые слова: РЕКОМЕНДАТЕЛЬНАЯ СИСТЕМА; ИНФОРМАЦИОННЫЙ ПОИСК; ОБРАБОТКА ЕСТЕСТВЕННОГО ЯЗЫКА; МАШИННОЕ ОБУЧЕНИЕ; НЕЙРОННАЯ СЕТЬ.

Объектом исследования магистерской диссертации являются задачи построения рекомендательных систем. Исследуются, в частности, вопросы адаптации алгоритмов машинного обучения, которые используются в задачах информационного поиска, бинарной классификации и генерации текстов на естественных языках для их использования в рекомендательных системах.

Цель работы – исследовать способы построения рекомендательных систем, описать подходы к построению рекомендаций, проанализировать основные проблемы, возникающие в таких системах; разработать рекомендательную систему на примере музыкального сервиса.

Результаты исследования:

- Изучены основные подходы к построению рекомендательных систем,
- Предложено несколько способов построения рекомендательных систем, исследованы их достоинства и недостатки,
- Описаны способы получения данных, сами данные подготовлены,
- Получено несколько моделей рекомендательных систем, по результатам их анализа построена гибридная система,
- Разработано и реализовано мобильное приложение для работы с построенной рекомендательной системой.

Областями применения полученных моделей являются прикладные задачи, возникающие в системах, предоставляющих пользователям доступ к контенту. Описанные алгоритмы и основные достижения могут быть расширены и на другие классы систем, а не только на музыкальные сервисы.

Работа состоит из трёх глав. В первой главе даны основные понятия, подробно изложена задача и представлены способы её решения с помощью аппаратов информационного поиска, машинного обучения и алгоритмов обработки текстов. Во второй главе описываются проблемы получение данных, а также вычислительные эксперименты, проведённые в результате применения алгоритмов из первой главы, строются несколько моделей системы и на их основе затем создаётся гибридная система. В третьей главе даны обоснования необходимости построения клиентской части системы, проведён выбор технологий, описаны сложности, возникшие при практической реализации.

АГУЛЬНАЯ ХАРАКТАРЫСТЫКА ПРАЦЫ

Магістарская дысертацыя, 54 с., 18 мал., 5 табл., 11 крыніц**.**

Ключавыя словы: РЭКАМЕНДАЦЫЙНАЯ СІСТЭМА; ІНФАРМАЦЫЙНЫ ПОШУК; АПРАЦОЎКА НАТУРАЛЬНАЙ МОВЫ; МАШЫННАЕ НАВУЧАННЕ; НЕЙРОННАЯ СЕТКА.

Аб'ектам даследавання магістарскай дысертацыі з'яўляюцца задачы пабудовы рэкамендацыйных сістэм. Даследуюцца, у прыватнасці, пытанні адаптацыі алгарытмаў машыннага навучання, якія выкарыстоўваюцца ў задачах інфармацыйнага пошуку, бінарнай класіфікацыі і генерацыі тэкстаў на натуральных мовах.

Мэта працы - даследаваць спосабы пабудовы рэкамендацыйных сістэм, апісаць падыходы да пабудовы рэкамендацый, прааналізаваць асноўныя праблемы, якія ўзнікаюць у такіх сістэмах; распрацаваць рэкамендацыйную сістэму на прыкладзе музычнага сэрвісу.

Вынікі даследавання:

- Вывучаны асноўныя падыходы да пабудовы рэкамендацыйных сістэм,
- Прапанавана некалькі спосабаў пабудовы рэкамендацыйных сістэм, даследаваны іх добрыя якасці і недахопы,
- Апісаны спосабы атрымання дадзеных, самі дадзеныя падрыхтаваны,
- Атрымана некалькі мадэляў рэкамендацыйных сістэм, па выніках іх аналізу пабудавана гібрыдная сістэма,
- Распрацавана і рэалізавана мабільнае прыкладанне для працы з пабудаванай рэкамендацыйнай сістэмай.

Абласцямі прымянення атрыманых мадэляў з'яўляюцца прыкладныя задачы, якія ўзнікаюць у сістэмах, якія прадстаўляюць карыстачам доступ да кантэнту. Апісаныя алгарытмы і асноўныя дасягненні могуць быць пашыраныя і на іншыя класы сістэм, а не толькі на музычныя сэрвісы.

Праца складаецца з трох раздзелаў. У першай чале дадзены асноўныя паняцці, выкладзена задача і прадстаўлены спосабы яе вырашэнн. У другой чале апісваюцца праблемы атрыманне даных, а таксама вылічальныя эксперыменты, праведзеныя ў выніку прымянення алгарытмаў з першай кіраўніка, строюцтся некалькі мадэляў сістэмы і на іх аснове затым ствараецца гібрыдная сістэма. У трэцяй чале дадзены абгрунтавання неабходнасці пабудовы кліенцкай часткі сістэмы, праведзены выбар тэхналогій, апісаны складанасці, якія ўзніклі пры практычнай рэалізацыі.

ABSTRACT

Master's thesis, 54 p., 18 fig., 5 tab., 11 sources.

Keywords: RECOMMENDATION SYSTEM; INFORMATION SEARCH; PROCESSING NATURAL LANGUAGE; MACHINE TRAINING; NEURAL NETWORK.

The object of study is building recommendation systems. In particular, the problems of adaptation of machine learning algorithms that are used in the problems of information retrieval, binary classification, and natural language generation.

The purpose of the work is to investigate methods for building recommendation systems, describe approaches to prepare recommendations, and analyze the main problems that exist in such systems; to develop a recommendation system for a music streaming service.

The results of the study:

- The main approaches to the construction of recommendation systems are studied,
- Several methods for constructing recommender systems are proposed, their advantages and disadvantages are described,
- Data obtaining problems are solved, the data itself is prepared,
- Several models of recommendation systems are built. A hybrid system has been built based on the results of their analysis,
- A mobile application for the built recommendation system is implemented.

The fields of application of the created models are problems that arise in systems that provide users with access to content. The described algorithms and main results can be extended to other classes of systems, and not just to music services.

The thesis consists of three chapters. The first chapter gives basic concepts, sets out the problem in detail, and presents methods for solving it using information search, machine learning, and NLP algorithms. The second chapter describes the problems of obtaining data, as well as the computational experiments carried out as a result of applying the algorithms from the first chapter, several system models are built and a hybrid system is then created on their basis. Third chapter describes why is important to have our own client application, a selection of its technologies is made, and the difficulties faced during the implementation are described.

ВВЕДЕНИЕ

В несколько последних лет можно наблюдать рост популярности сервисов, предоставляющих доступ к каталогу контента и возможности взаимодействовать с этим контентом. Это и интернет-магазины, и сервисы доставки, и сервисы для потребления цифрового контента. Среди последних особенно популярны видео- и аудио-сервисы. Они, как правило, обладают большой фонотекой, ориентированной на широкий круг пользователей, и предоставляют доступ к такой фонотеке. Но пользователи технически не могут знать всего находящегося там контента и поэтому сталкиваются с трудностью нахождения нового для них материала.

Поэтому такие сервисы предоставляют широкий спектр рекомендаций, стараясь предложить пользователю тот контент, который будет ему потенциально интересен. Таким образом, от качества рекомендаций, от их релевантности сильно зависит удовлетворённость пользователей сервисом, а, следовательно, и количество времени, которые они будут на нём проводить.

Достаточно часто сама система рекомендаций является одной из самых наукоёмких частей сервисов. В ней находят своё применение и информационный поиск, и машинное обучение, и обработка естественных языков, и много других смежных областей.

В рамках магистерской диссертации как раз была выполнена работа по созданию такой рекомендательной системы для пользователей сервиса Яндекс.Музыка. В следующих главах будут даны некоторые теоретические сведения о рекомендательных системах и информационном поиске, изучены конкретные подходы к построению систем для рекомендаций, проанализированы все компоненты системы, описаны полученные знания о специфике сервиса Яндекс.Музыка, а также изложен практический опыт, полученный в результате проектирования и разработки рекомендательной системы.

ГЛАВА 1. ТЕОРЕТИЧЕСКИЕ ОСНОВЫ ПОСТРОЕНИЯ РЕКОМЕНДАТЕЛЬНЫХ СИСТЕМ

В этой главе будут даны теоретические сведения, необходимые для создания рекомендательной системы для аудиоматериалов, представлены некоторые алгоритмы информационного поиска и машинного обучения, которые будут использованы для построения рекомендательной системы, а также проанализированы средства, предлагаемые аппаратом NLP, и изложены способы использования этого аппарата в контексте построения систем рекомендаций.

1.1. Основные подходы к рекомендациям

Преследуя цель рекомендации релевантного для пользователей контента стоит дать более формальные определения терминам.

Под контентом далее будем понимать аудиоматериалы, доступные для прослушивания в системе. Под релевантным контентом будет понимать рекомендуемый контент, соответствующий интересам пользователя.

При рекомендации новых материалов пользователям можно отталкиваться от следующих идей:

1. (Content-based filtering) Можно предлагать контент, похожий на тот, который им уже понравился, т.е. базируясь исключительно на знаниях об похожести контента и не вовлекая никаких персонализированных рекомендаций (пользователю понравился контент А, контент А похож на контент Б, рекомендуем пользователю контент Б).
2. (Collaborative filtering) Можно предлагать контент, который оценили другие, похожие пользователи. Здесь вопрос об нахождении похожего контента не ставится, задача состоит в том, чтобы оценить похожесть пользователя и порекомендовать то, что похожему пользователю уже понравилось (Пользователь А похож на Пользователя Б, Пользователю Б понравился контент, порекомендуем контент и Пользователю А)
3. Комбинированный подход, где оценивается как похожесть пользователей, так и похожесть контента.

Заметим, что у всех пунктов есть свои плюсы и минусы и попробуем их перечислить.

Заметным плюсом первого подхода с точки зрения построения рекомендаций является его независимость от пользователя, т.е. для нового пользователя не требуется обработки его интересов. В случае предрасчитанной

похожести контента можно почти сразу рекомендовать пользователю что-то новое, т.е. система легко масштабируется по количеству пользователей.

Из минусов можно заметить сложность формализации похожести между разным контентом и чувствительность качества рекомендаций к функции похожести. При неверном её выборе мы получим очень низкое качество рекомендаций. Также возникает следующая проблема: функция похожести коммутативна (контент А похож на контент Б ровно настолько, насколько контент Б похож на контент А), поэтому мы можем испытывать проблемы с зацикливаемостью рекомендаций.

Несомненным плюсом второго подхода является его подстраиваемость под каждого конкретного пользователя и легкая расширяемость для нового контента (не требуется делать настолько сильного и потенциально энергоёмкого сравнения контента, как в первом пункте) и отсутствия минусов первого пункта.

Минусами можно назвать необходимость нахождения похожих пользователей, что, на первый взгляд, является затратной операцией и плохо работает онлайн (т.е. пользователю требуется потребить некоторое, возможно немалое, число контента, чтобы найти похожих на него пользователей). Также, очевидно, для того чтобы для каждого нового пользователя можно было найти пользователя с релевантными интересами, в системе должно уже быть некоторое критическое число активных пользователей. На первых порах в системе больше контента, чем пользователей (в случае музыкальных сервисов не является разумным иметь в фонотеке лишь несколько сотен композиций), что делает невозможным нахождение очень похожих пользователей.

Таким образом, основными проблемами, с которыми сталкиваются рекомендательные системы - новые пользователи, которые не успели потребить достаточное число контента, и новый либо редкий контент, на который пока не нашёлся похожий существующий. И если вторая проблема не мешает нам строить хорошие рекомендации на основе уже предобработанного контента (мы просто не будем рекомендовать новый до его полной обработки), то с первой проблемой всё равно нужно бороться.

Также уже было вскользь упомянуто, что системы могут столкнуться с чисто вычислительно сложными задачами как при росте количества контента, так и при увеличении числа пользователей, т.е. присутствует уязвимость к т.н. “проклятью размерности”.

Что касается выбора способов рекомендации, то оптимальным будет являться комбинированный подход, когда, например, на начальных этапах пользователю будут даваться либо рекомендации под среднего пользователя,

либо на очень похожий на уже одобренный контент. В дальнейшем, когда будут более понятны предпочтения пользователя, можно будет сделать рекомендации персонализированными.

Ошибки в рекомендациях нежелательны, а ошибки в рекомендациях новым пользователям даже опасны, т.к. могут сразу убедить их перейти на пользования неким конкурирующим сервисом. В целом, проблема рекомендации контента свежим пользователям несколько выходит за рамки данной диссертации и подробно рассматриваться не будет.

1.2. Методы оценки подобия контента

Разумеется, при оценке подобия контента требуется учитывать его структуру и извлекать пользу из её особенностей, т.е. при попытке нахождения максимально похожих композиций смотреть на жанр, количество ударов в минуту и прочие параметры аудиодорожки. Такими подходами, однако, подобие не ограничивается. Например, непосредственно с контентом в сопровождении идёт большое количество метаданных, на основе которых тоже можно сделать выводы о похожести.

Разумно рекомендовать следующую серию сериала, а не какую-то случайную в случае видео-сервиса, можно посоветовать другие песни любимого исполнителя в случае аудио-контента. Но при работе с аудио из-за специфики структуры песен (недлинные записи, часто с текстом), можно извлечь дополнительную выгоду при обработке текстов песен при их наличии.

Сфера обработки текстов на естественных языках является достаточно продвинутой и может сильно помочь при оценке подобия песен. Например, можно рекомендовать песни, соответствующие по настроению тем песням, которые сейчас слушает пользователь, или песни, связанные общей темой с текущими. Такое подобие положительно скажется на релевантности рекомендаций.

Можно заметить, что стратегия рекомендации самой похожей песни не является выигрышной, т.к. самыми похожими, очевидно, являются одинаковые песни (например, разные версии исполнения разными артистами одной и той же песни), т.е. необходимо решать более сложную задачу и дополнительно отфильтровывать чересчур похожий контент. Рекомендации также не должны ставить целью предоставления только похожего контента, важной чертой хорошей системы является способность “предугадывать” вкус пользователя и предоставлять ему новый контент, который до этого был ему незнаком. Т.е. рекомендованные материалы должны быть похожи не в буквальном смысле, а в

смысле похожести на ожидания пользователя, т.е., по факту, быть релевантными его запросам.

Рассмотрим подробнее способы нахождения релевантных песен для рекомендации пользователю и определения похожести тексов песен.

1.3. Задачи информационного поиска в рекомендательных системах

Задачу рекомендации контента пользователю можно рассмотреть, как задачу поиска контента из множества, где поисковым запросом являются предпочтения пользователя. Обработать непосредственно предпочтения не представляется возможным, мы можем работать с ними только по косвенным признакам, таким как уже одобренный контент. Таким образом, мы можем воспользоваться аппаратом информационного поиска для нахождения подобия.

Как уже было сказано, мы будем искать подобные песни, в частности, по их текстам. Эта информация не будет возвращена непосредственно пользователю как рекомендация, но будет положена в основу, и, будучи скомбинированной с другими характеристиками, которые будут описаны позднее, повлияет на итоговый рекомендуемый контент.

Сравнивать непосредственно тесты песен и как-то работать с текстами как с единым целым не представляется разумным, поэтому для начала нужно разбить текст песни на слова, что может являться нетривиальной задачей. В рассматриваемой системе мы не будем уделять внимание языкам с иероглификой, ограничимся лишь языками, использующими латиницу и кириллицу. Для таких языков достаточным будет деление на токены по пробельным символам и знакам препинания, а также исключение символов, которых нет в алфавите (для текстов на кириллице можно оставлять также латинские символы). Стоит отменить, что можно всё-таки оставить некоторые спецсимволы, например, дефисы в середине слов.

Далее требуется привести все слова к единой форме. Опять же, эта проблема не стоит остро в, например, английском языке, где все формы, по сути, близки к инфинитиву. В русском же языке к основам добавляются приставки, суффиксы и окончания, которые нам, как разработчикам системы, требуется удалить. Это можно достичь, прибегнув к лемматизации (т.е. процессу приведения словоформы к лемме), однако лемматизация подразумевает наличие словаря и априорных знаний правил языка, создание качественной лемматизирующей системы выходит за рамки данной диссертации.

Однако, нам не нужно, чтобы все формы были приведены к инфинитиву, наша задача состоит в том, чтобы все формы привести к какому-то одному виду, возможно даже не существующему с точки зрения морфологии. Как раз такую задачу выполняет стеммер, примерами которого могут служить стеммер Портера и MyStem.

Стеммеры, в отличие от лемматизаторов, не обладают обширным словарём и знаниями о многочисленных граматических правилах, об сфере их применимости и исключениях. Наоборот, стеммеры представляют собой достаточно простые автоматы, состоящие их некоторого числа переходов.

Например, в упомянутом стеммере Портера для английского языка есть такие правила:

1. Если слово заканчивается на SSES, то удалить последние 4 буквы и добавить SS в конец.
2. Если слово заканчивается на IES, то удалить последние 3 буквы и добавить I в конец.
3. Если слово заканчивается на SS, то удалить последние 2 буквы.
4. Если слово заканчивается на S, то удалить последнюю букву.

Для русского языка тоже есть свои стеммеры, например, алгоритмы стеммера Портера адаптированы для русских слов. Точность и корректность таких алгоритмов зачастую невысока, т.к. обилие форм, времён и исключений делают задачу качественного стемминга трудновыполнимой. Банально, слова “идёт” и “шёл” должны были бы быть приведены к одной и той же форме, но у них только одна общая буква (гласная, причём). Хорошие попытки улучшить качество стемминга для русских слов предпринимаются компанией Яндекс, они предлагают воспользоваться их наработками в этой области через использование их стеммера MyStem (доступны только собранные версии, ни алгоритм, ни исходные коды в общем доступе не найдены).

Очевидным плюсом простых алгоритмов стемминга является скорость их работы. В отличие от лемматизации, стеммеру не нужен контекст, поэтому процесс смемминга можно еще и распараллеливать.

Следующим пунктом стоит избавление от стоп-слов, т.е. слов, которые встречаются почти во всех документах, но не несут смысловой нагрузки. Это всевозможные связующие слова, предлоги, частицы. В случае с текстами песен это могут быть также специфические слова типа “Припев” или расставленные над строчками аккорды. В зависимости от задачи, можно считать стоп-словами и числительные.

Для естественных языков доступны списки соответствующих стоп-слов, что может быть удобно при работе со специфическими текстами, например, в

которых естественные стоп-слова встречаются не так часто. Полезно заранее из рассмотрения исключить все слова, которые заведомо являются стоп-словами. Хотя это не исключает того, что всё равно необходимо очистить тексты от актуальных стоп-слов (как было упомянуто, слово “привев” в контесте всего русского языка не является стоп-словом, и, соответственно, не входит ни в один список стоп-слов, однако оно может являться стоп-словом в контексте корпуса текстов песен).

Итак, в результате токенизации и нормализации текста у нас теперь тексты представлены массивами слов, для сравнения которых на похожесть мы можем применить tf-idf.

1.4. Мера tf-idf и её применение при оценке подобия текстов

В задачах информационного поиска часто встречается необходимость определить уровень подобия некоторого числа текстов. Для решения этой задачи применяют механизмы, построенные на tf-idf.

Идея, которая стоит за этой мерой, достаточно проста: для каждого слова (терма) в документе можно оценить его важность в этом документе следующим образом:

$$tf(\text{терм}, \text{документ}) = \frac{\text{число вхождений терма в документ}}{\text{общее число термов в документе}}$$

$$idf(\text{терм}, \text{документы}) = \log\left(\frac{\text{общее число документов}}{\text{число документов, включающих терм}}\right)$$

Для каждого терма в каждом документе нас интересует произведение tf * idf. Технически мы выходим из предположения, что чем чаще слово встречается в тексте, тем более оно значимо для определения темы этого текста. Этого, однако, недостаточно, некоторые слова встречаются во всех текстах и больших количествах, но не несут смысловой нагрузки, поэтому мы штрафуем за слова, которые встречаются уж слишком часто.

На практике бывает выгодно рассматривать не все термы из всех документов (получившаяся матрица оказывается слишком большой, ведь каждое редкое слово или опечатка приводит к появлению дополнительного столбца), а только полученные из какого-либо подмножества документов. Можно использовать альтернативный подход: сгруппировать все термы из всех документов, но взять только те термы, которые встречаются больше некоторого порогового числа раз (например, 100), но не более, чем в неком пороговом числе документов (из тех же соображений, что и при удалении стоп-слов).

Далее для каждого документа его строку-представление tf-idf матрицы считать вектором в некотором пространстве. Тогда мы можем подсчитать угол между этими векторами и на основе угла сделать вывод о похожести. Чем меньше угол, тем более похожи тексты. На практике удобно считать не сам угол, а его косинус (чем ближе косинус угла к 1, тем тексты более похожи).

Таким образом, для корпуса тексов песен можно преобразовать их вектора и через попарные углы между векторами оценить попарную схожесть текстов, а на основе схожести основывать дальнейшие рекомендации.

Тексты большинства песен соответствуют схеме ”куплет-припев”, т.е. имеют большое число повторений припева. Часто и сам привет состоит из небольшого числа часто повторяющихся слов. Получается, что при описанном подходе значимыми в каждом документы могут стать только повторяющиеся слова припева, т.к. их количество в песне будет нивелировать все остальные слова. Для того, чтобы сбалансировать значимость слов, мы будем в качестве tf брать не количество вхождений слова в текст, а логарифм этого числа.

Обратим внимание, что в большом корпусе текстов обязательно найдутся дубликаты либо различные версии одних и тех же документов. Очевидно, что дубликаты будут максимально похожи между собой. Таким образом, при построении рекомендаций требуется отсекать слишком похожие документы (те, например, у которых косинус угла больше 0.7).

1.5. Задачи двоичной классификации в рекомендательных системах

Если зайти с другой стороны, то задачу рекомендации пользователю контента можно рассмотреть и как задачу двоичной классификации контента для каждого пользователя: каждый контент попадает либо в группу релевантных, либо в группу нерелевантных.

Для решения указанной задачи можно применить метод опорных векторов с комбинации с вышеупомянутой мерой tf-idf. Для этого мы сначала строим матрицу tf-idf, потом берём список просмотренного пользователем контента и преобразовываем его к вектору из 0 и 1 (где 1 означает, что контент понравился, а 0 – что контент не был одобрен). Далее берётся некоторое число случайного контента и ему проставляются случайные отметки из {0, 1}. Т.к. уверенность в оценках разная, то оценки непосредственно от пользователя берутся с большим весом, а случайно добавленные – с меньшим.

Далее на этих данных мы обучаем метод опорных векторов, а затем классифицируем весь имеющийся контент. SVM отнесёт каждый контент к

какому-то классу с некоторой вероятностью. Эти значения мы преобразовываем в вероятность попадания контента в класс 1, сортируем контент в порядке убывания этой вероятности и можем использовать эти данные для построения финальных рекомендаций (разумеется, необходимо также предварительно удалить или минимизировать рекомендацию того контента, который пользователь уже видел).

Такой способ требует больше вычислительных усилий (предрасчёт должен выполняться для каждого пользователя), зато позволяет получить персонализированные рекомендации. Заметим также, что можно подбирать параметры метода опорных векторов через деление информации о одобренном контенте на обучающую (train) и валидационную (validation) выборки, что позволит улучшить показатели и уменьшить вероятность переобучения.

Тот факт, что разные оценки имеют разный вес, позволит более гибко управлять рекомендациями. Например, пользователь может априорно уменьшить вес для всех композиций какого-либо исполнителя. Также можно ввести разные градации оценивания (подробнее об этом речь пойдёт в следующей главе), которые не только сгладят тот факт, что музыкальная составляющая не берётся в расчёт, но и позволят получить качественные и релевантные рекомендации.

Итак, было рассмотрено несколько способов построения рекомендаций исключительно на основе схожести текста контента. Рассмотрим другие возможные подходы.

1.6. Задачи генерации текстов в рекомендательных системах

Итак, до сих пор рассматривался лишь один пользователь и во внимание бралась лишь его история потреблённого контента. На практике системами пользуются большое число людей (достигающее десятков и сотен миллионов), что позволяет основывать рекомендации в том числе и на действиях других пользователей.

Например, обладая историей просмотра контента другими пользователями, можно найти для текущего наиболее вероятный следующий контент. Т.о. можно говорить об генерации текстов на некоем фиктивном языке, где единицей (словом) является единица контента, а текстом – история потребления. Корпусом текстов будем называть множество текстов, т.е. группу очередей потребления.

Задача генерации текстов (NLG) является одной из задач класса NLP, в нашем случае мы можем привлечь механизмы обработки текстов на

естественных языках для обработки текстов (и последующей генерации новых) на описанном выше языке.

В рамках NLP язык представляется языковой моделью, т.е. машино-понятной моделью грамматики языка, описывающей вероятности появления конкретных языковых конструкций. Дальнейшие алгоритмы оперируют уже в терминах языковых моделей, т.о. для возможности применения аппарата NLP требуется вначале построить языковую модель.

Самой примитивной языковой моделью может служить униграммная модель, где каждому слову в языке ставится в соответствие его вероятность его появления в этом языке. Сумма всех вероятностей, очевидно, составляет 1.

Обобщением униграммной является n-граммная модель, где появляется некоторая информация о контексте, т.е. для каждой возможной последовательности из n слов языка мы знаем вероятность появления такой последовательности.

Близкой по смыслу к n-граммной модели является skip-gram, в которой речь идёт не про n слов языка, встречающихся подряд, а про n слов, встречающихся последовательно с некоторыми пропусками. Такая модель хорошо подходит для задач нахождения слова по его контексту (ведь она как раз и описывает контекст пропущенного слова).

В нашей задаче мы будем применять т.н. “мешок слов” (bag-of-words), в которой текст представим в виде пар ключ-значение, где ключом является слово, а значением – количество повторений его в тексте. Обычно такие модели применяют для определения слова по заданному контексту.

Такая модель в наших условиях действительно имеет смысл, т.к. контент непосредственно друг с другом не связан (грамматика у нашего языка теоретически позволяет любые последовательности). Нам не важно различать тексты, полученные лишь перестановкой слов, нам важны только входящие слова.

1.7. word2vec и его применение

Пользуясь ранее подходами, использующими tf-idf, мы уже отмечали один из важных минусов – размерность векторов, с которыми приходится работать. Размерность каждого вектора равна числу слов в словаре, что превышает десятки тысяч. Даже пользуясь для хранения числами с плавающей точкой одинарной точности, матрицы с такими векторами занимают много места и требуют крупных мощностей для оперирования с ними (при нахождении максимально похожих контентов мы, например, попарно

умножали все вектора). Заметим, что значительную часть этих матриц составляют нулевые элементы, что логично, ведь контент не может быть связан каждым значащим словом с каждым значащим словом каждого другого контента.

Подход на основе word embeddings позволяет снизить размерность векторов до приемлемого уровня (до сотен вместо десятков тысяч). Т.е. мы по-прежнему каждому слову ставим в соответствие вектор, но куда меньшей длины, чем в tf-idf. Более того, т.к. word embeddings тоже получены на основе корпуса текстов, то вектора будут обладать некоторыми удобными для нас характеристиками, в частности, они и будут представлять нашу языковую модель и, в дальнейшем, мы будет оперировать в терминах word embedding (т.е. в нашей модели каждому слову языка будет поставлен свой вектор небольшой размерности).

Алгоритм word2vec является одним из способов получения word embeddings. Он представляет из себя нейронную сеть с одним скрытым слоем. После обучения веса с этого слоя лягут в основу word embeddings. Полученные вектора будут чем ближе друг к другу, чем чаще соответствующие им слова входят в один контекст.

Word2vec позволяет получить результат на основе одной из двух архитектур (continuous bag of words либо skip-gram). Т.к. наша задача состоит в предсказании следующего слова по предыдущим, то в нашем случае стоит использовать первый вариант.

Рассчитав word embeddings, можно приступить к генерации текстов. Самым примитивным алгоритмом будет такой: для последнего потребленного контента ищем наиболее близкий (т.е. с наименьшим расстоянием между соответствующим им word embeddings). Такой алгоритм, однако, на практике не очень хорош, т.к. достаточно быстро начинает зацикливаться и рекомендовать контент, который уже был недавно просмотрен. Чтобы не загонять пользователя в такой “пузырь рекомендаций” требуется выбирать не наиболее близкий, а случайный из какого-то числа наиболее близких. Этот подход будет подробнее описан далее.

1.8. Использование LSTM для генерации текстов

LSTM (Long shot-term memory, долгая краткосрочная память) – архитектура рекуррентных нейронных сетей (RNN, recurrent neural network), используемая в т.ч. при генерации текстов, основанная на LSTM-ячейках. Плюсом этой архитектуры являются хорошие показатели в работе с последовательностями данных.

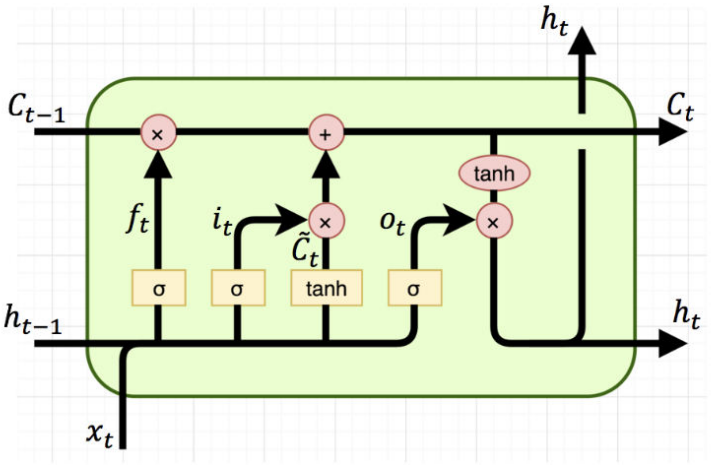

Рисунок 1.1. Структура LSTM-ячейки

Как было сказано, удобно использовать LSTM при генерации текстов. Для этого в нейронной сети достаточно всего нескольких слоёв:

1. Слой с предобученными word embeddings,
2. LSTM-слой,
3. Dropout-слой для противодействия переобучению,
4. Полносвязный Dense-слой для преобразовывания в n-мерный вектор вероятностей.

Далее будет использоваться похожая, но несколько усовершенствованная архитектура сети.

1.9. Выводы

В данной главе были проведены следующие исследования:

- рассмотрены подходы к построению рекомендательных систем,
- дано описание аппарата информационного поиска и обработки естественных языков, описаны грани применимости таких алгоритмов,
- изложено несколько способов построения рекомендательных систем для аудио-сервисов,
- проведён анализ этих способов, выявлены их положительные и отрицательные стороны.

ГЛАВА 2. ПОСТРОЕНИЕ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ ДЛЯ АУДИОСЕРВИСА

В этой главе будет дано описание сервиса Яндекс.Музыка, рассмотрена задача построения рекомендательной системы для этого сервиса, изложен практический опыт сбора и анализа данных, описаны реализации рассмотренных в первой главе концепций по построению рекомендательных систем, а также описаны проблемы, с которыми приходилось сталкиваться по мере реализации этих систем.

2.1. Сервис “Яндекс.Музыка”

“Яндекс.Музыка” – один из сервисов компании “Яндекс”, он, среди прочего, предоставляет пользователям доступ к фонотеке, содержащей аудиоматериалы (как песни, так и подкасты, новости и прочее) и позволяет их прослушивать, делиться и составлять их них подборки (плейлисты). Пользование сервисом осуществляется на основе модели по подписке, некоторые функции доступны и без подписки, но только на территории 11 стран (включая Россию и Беларусь), по подписке же сервисом можно пользоваться по всему миру.

Сервис появился в 2010 году, на конец 2019-го года насчитывает более 3 миллионов пользователей, имеющих платную подписку. В фонотеке сервиса на начало 2020-го года представлены 63 миллиона композиций.

Далее будут подробно изучены возможности сервиса по предоставлению метаданных к композициям и будет построена рекомендательная системы для пользователей сервиса.

2.2. Способы получения данных о композициях

Как было показано в первой главе, для начала построения рекомендательной системы первым, что необходимо получить, являются данные о контенте. Т.к. мы разрабатываем системы для стороннего поставщика контента, то у нас нет доступа непосредственно к данным сервиса и нам придётся собирать их самим. Для начала можно оценить то, что доступно через официальные приложения и клиенты, например, через веб-версию.

Заметим, что при клике на композицию открывается страница, где доступны текст песни, альбомы, в которых она встречается, а также перечислены некоторые похожие композиции. При наличии также указаны другие версии композиции (исполненные, например, другими исполнителями или исполненные вживую).

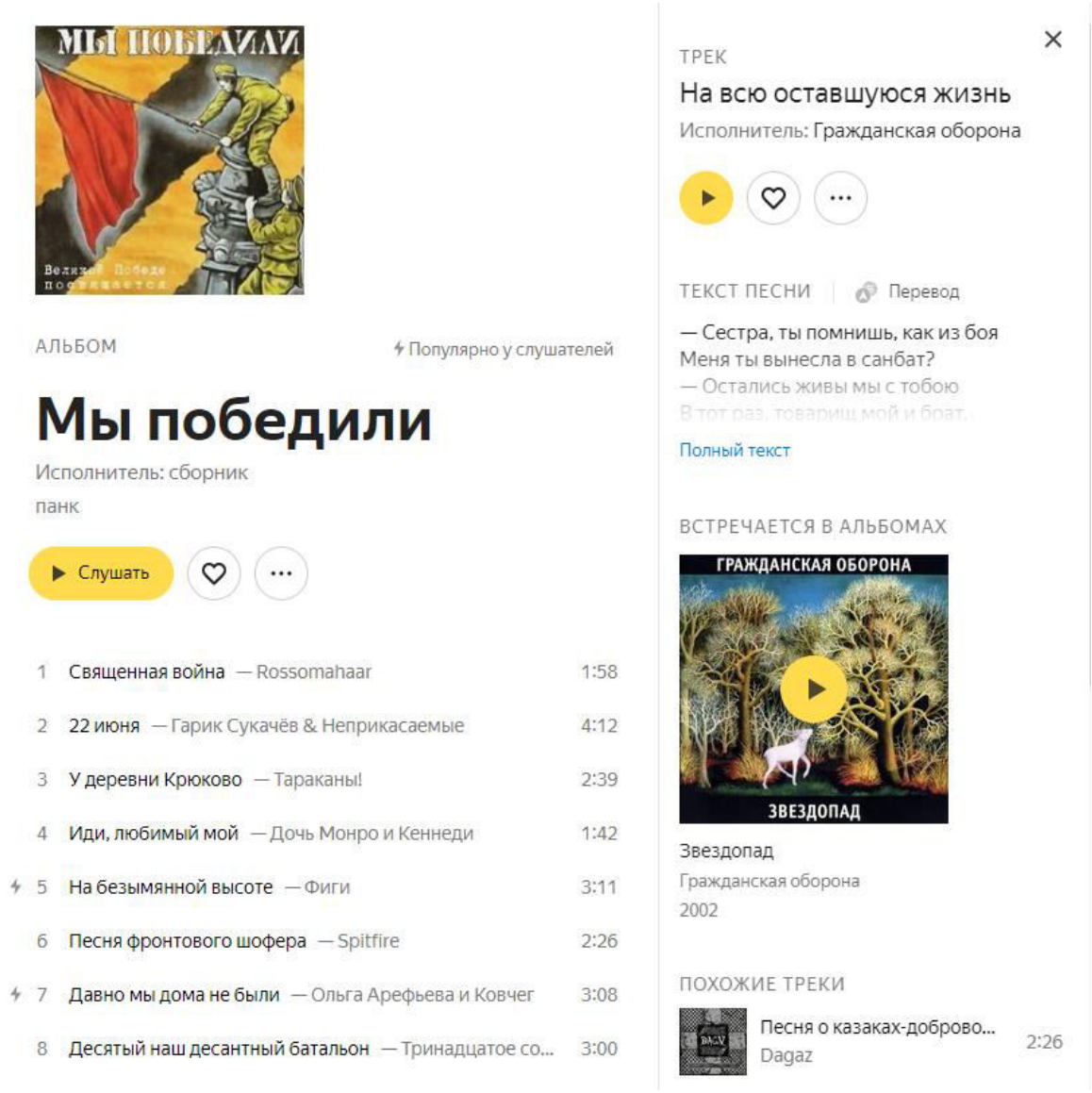

Рисунок 2.1. Пример информации о композиции

Воспользовавшись каким-нибудь программных обеспечением для изучения сетевого траффика (в нашем случае использовался fiddler), можно получить формат конечной точки HTTP, на которую можно обратиться для получения композиции с заданным идентификатором (`https://music.yandex.ru/handlers/track.jsx?track={id}`). В качестве ответа на GET-запрос на эту точку будет получен json-документ, содержащий всю информацию о композиции, визуализированную на рисунке 2.1, а именно информацию об исполнителях, альбомах, похожих композициях, версиях и прочей информацией, в т.ч. содержащей флаг наличия текста у композиции и самого текста (с указанием языка текста).

Проблемы, с которыми сразу приходится столкнуться, очевидны: во-первых, необходимо знание о структуре идентификатора композиции, во-вторых, нужно знать все идентификаторы всех композиций, чтобы их получить.

Было замечено, что полный идентификатор композиции имеет вид “идентификатор_композиции:идентификатор_альбома”, однако, при отсутствии последнего сегмента (т.е. при указании лишь идентификатора композиции) ответ всё равно возвращается (“идентификатор_альбома” влияет в большей степени на изображение для композиции, шаблон для ссылки на которую также возвращается в ответе). Также обнаружилось, что “идентификатор_композиции” – положительное число, а эти числа для разных композиций выделяются подряд. Таким образом, технически можно сделать какое-то количество запросов информации о композициях, просто итерируя счётчик от 1 до максимально возможного значения.

На момент написания отчета экспериментально было замечено, что в фонотеке не более 63 миллионов композиций (запросы композиций с несуществующими идентификаторами возвращали код ответа 200 OK, но внутри содержали информацию о том, что композиция не найдена). Это число подтверждалось также некоторыми косвенными признаками (например, среди самых свежих добавленных на сайт композиций ни у одной идентификатор не превышал 63-х миллионов).

Полученной таким образом информации достаточно, чтобы написать программный компонент, который сделает 63 миллиона запросов на публичные конечные точки, разберёт ответы и сложит информацию о композициях в базу данных.

Единственным моментом, на который хотелось бы дополнительно обратить внимание – ограничение API Яндекс.Музыки на количество запросов в секунду. Т.к. мы используем недокументированное API, то открытой информации об ограничениях не имеется. Было замечено, что некоторые запросы возвращаются с кодом ответа 522 Bad Gateway (вместо 200 OK), а некоторые – коды ответа 3XX и перенаправление на страницу для ввода captcha. Идентификаторы, для которых были получены такие ответы, были помещены в отдельную очередь, и были впоследствии запрошены повторно.

Запросы слались пачками по 10000 в параллели, заголовки Set-Cookie, посылаемые сервером, игнорировались, т.к. в противном случае возникали проблемы с ограничениями на количество запросов, описанные выше. Также были проведены эксперименты по использованию сжатия (API умеет возвращать сжатые gzip данные при указании специального заголовка), но в итоговой версии было решено от отказаться от сжатого контента, т.к. бутылочное горлышко было в потреблении CPU, а не в пропускной способности сети, а необходимость разжимать контент требовала дополнительных нагрузок на CPU.

Описанным образом данные об всех композициях были получены и сохранены, на все запросы были затрачено суммарно около 50 часов.

Было также обнаружено, что некоторые идентификаторы не выделены никаким композициям, информация о несуществующих композициях, очевидно, игнорировалась.

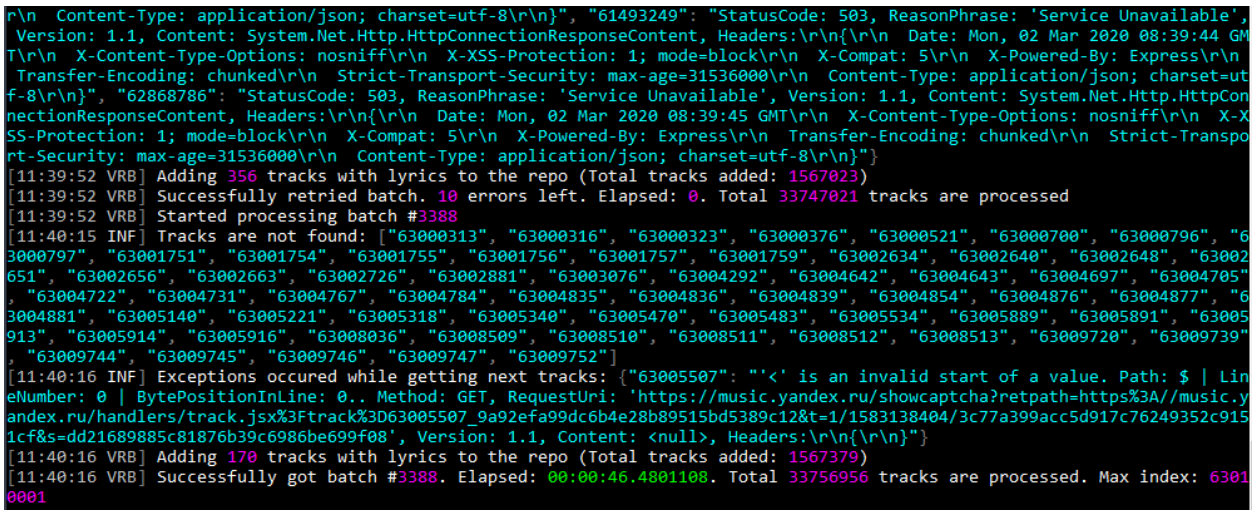

Рисунок 2.2. Фрагмент вывода во время сбора информации о композициях

Оказалось, что из всех композиций информацией о текстах обладают только 4603880, что составляет 7.3%. Интересно, что среди первого миллиона запрошенных композиций этот показатель был около 40%. Впрочем, неудивительно, что не все композиции размечены. Во-первых, не все являются песнями со словами, во-вторых, не все они являются песнями – как уже было сказано, сервис также поддерживает подкасты.

Причем, несмотря на то, что в json-объекте ответа ключу “lyric” соответствует массив, на практике ни у одной композиции не встретилось отличное от одного число текстов.

2.3. Исследование полученных данных

Тексты представлены на 85 разных языках. Количество текстов на конкретном языке варьируется от 1 (узбекский и тальмильский) до 3537607 (английский). График, визуализирующий количество композиций для семи следующих за английским языков представлен на рисунке 2.3. Заметим, что для около 33000 композиций указанный текст не размечен никаким языком. Такие тесты были исключены из дальнейшего рассмотрения и в статистику не попали.

В дальнейшем примеры будут приводиться на русском и на английском как на самых распространенных в русскоговорящем сегменте пользователей

языках композиций. Описанные в главе 1 алгоритмы будут одинаково применимы и для других языков (на которых наберется критическая масса размеченных текстов).

Имея такое количество текстов, сложно следить за уровнем их качества, поэтому встречаются тексты с неправильно промаркированным языком (песня на английском, текст на английском, а указано, что он на русском) или не совсем правильные тексты (песня на английском, вместо оригинального текста представлен перевод на русский, итоговая песня помечена как русская), а также достаточно много других артефактов (указания “Не забыть вставить сюда текст песни”, например).

После того, как мы скачали все тексты песен себе, можно построить некоторые статистики и найти, например, топ-10 исполнителей с самым большим числом текстов (таблица 2.1).

Русскоязычный исполнитель и количество проаннотированных текстов	**Исполнитель и количество проаннотированных текстов**
Владимир Высоцкий (985)	Frank Sinatra (40063)
Михаил Шуфутинский (866)	Elvis Presley (38380)
Аквариум (832)	Ella Fitzgerald (26532)
Александр Розенбаум (736)	Billie Holiday (25876)
Noize MC (606)	Dean Martin (20885)
Крематорий (507)	Johnny Cash (19787)
Пурген (484)	Nat King Cole (19170)
Филипп Киркоров (454)	Ray Charles (17645)
КИНО (445)	Louis Armstrong (15424)
Юрий Визбор (444)	Perry Como (12659)

Таблица 2.1. Топ-10 исполнителей по количеству текстов песен среди русскоязычных (слева) и мировых (справа)

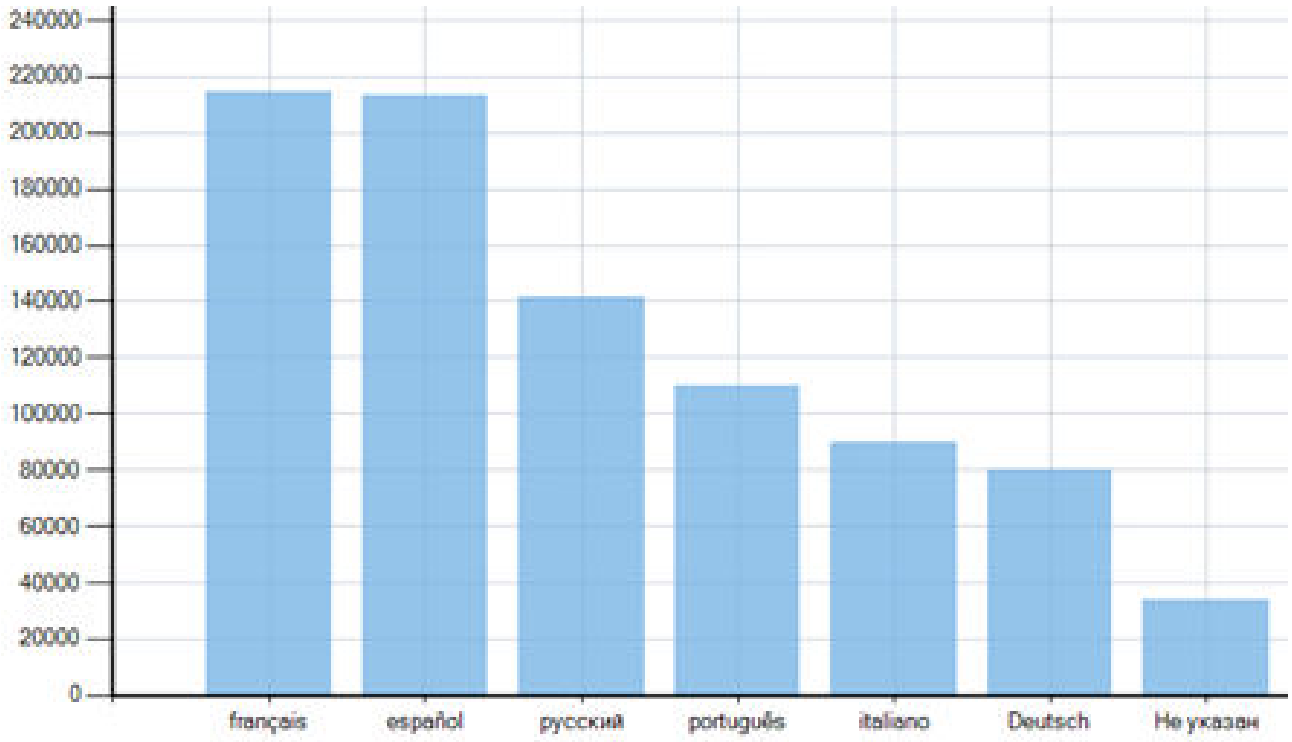

Рисунок 2.3. График зависимости количества текстов от языка

Заметим, что у самого популярного по количеству текстов песен исполнителя (Фрэнк Синатра) присутствует более 40 тысяч композиций, что кажется очень большим количеством. В действительности, конечно, у него нет такого числа композиций, это ошибки контент-менеджеров, которые многократно загружали одни и те же композиции вместо того, чтобы ссылаться на уже существующие. Вероятно, что это также связано с разными издателями (label) и имеет отношение к авторским правам.

При загрузке новых композиций на Яндекс.Музыку контент-менеджер можно указать, что загружаемый трек является версией уже существующего. Поддерживаются следующие категории версий: “remix”, “версия в живом исполнении”, “инструментальная версия” и “иное”. Многие композиции, однако, загружено без указания связи с уже загруженными композициями, что позволяет их считать разными, хотя и это не соответствует действительности. Таким образом у указанных в таблице 2.1 исполнителей и набралось такое число композиций.

Можно попробовать оценить количество точных дубликатов.

Количество вхождений текста	**Эпизодов для русского языка**	**Эпизодов для английского языка**
2	17364	184790
3	6652	50280
4	2974	20878

5	1400	10676
6	780	6585
7	444	4606
8	260	3320
9	188	2561
10	134	2018

Таблица 2.2. Количество дубликатов с 2 – 10 эпизодами для русского и английского языков

Также можно оценить самые часто встречающиеся композиции.

Песня, помеченная русской, и число её вхождений	**Песня, помеченная английской, и число её вхождений**
Louis Armstrong - La Vie En Rose (189)	The Ink Spots - Stardust (3662)
Elvis Presley - I Need Your Love Tonight (173)	Oscar Peterson - Blue Moon (2814)
Avichi - 3 da Hard Away (69)	Wayne Newton - Jingle Bell (2748)
The Barry Sisters - Chiribim Chiribom (57)	John Denver - Silent Night (2578)
Серёга (Группа ПМ) - К Элизе (50)	Judy Garland - Over the Rainbow (2515)
The Barry Sisters - Papirossen (46)	Joe Williams - How High The Moon (2146)
New Day - Тинто Брасс (46)	Bing Crosby And The Andrews Sisters - Santa Claus Is Comin' to Town (2132)
Лев Лещенко - День Победы (33)	Nat King Cole - Slow Down (1999)
Smash - Moscow Never Sleeps (31)	Artie Shaw - Begin The Beguine (1950)
Вячеслав Добрынин - Где же ты была (28)	Louis Armstrong - Ain't Misbehavin' (1927)

Таблица 2.3. Топ-10 самых частых композиций и количество их вхождений для композиций с текстом на русском (слева) и английском (справа)

Обратим внимание, что композиция Stardust встречается 3662 раза! Имена исполнителей условны и представляют лишь одни из возможных значений. У композиции Jingle Bell, например, достаточно много различных исполнителей, перечисление их всех затруднительно.

Можно заметить, что четыре самые часто встречающиеся композиции на русском на самом деле не имеют ни одного русского слова, что представляет

собой типичную ошибку контент-менеджера (некоторые примеры ошибок уже были описаны выше). В дальнейшем из рассмотрения (и, соответственно, из рекомендуемых как композиции на русском) такие композиции были исключены. Т.е. если в тексте рассматриваемой композиции нет ни одного символа из кириллического алфавита, то он в нашей системе перестает считаться текстом на русском языке.

При дальнейших операциях указанные четкие дубликаты (ровно, как и различные версии одних и тех же композиций) были сгруппированы по тексту и считались одной композицией. Если требовалось порекомендовать пользователю композицию, которая на деле оказывается такой сгруппированной, то каждый раз из группы случайным образом выбирался один представитель и именно он шёл в итоговые рекомендации.

Заметим, что если из текстов на русском, которых изначально насчитывалось 141358, убрать чёткие дубликаты, т.е. те композиции, тексты которых посимвольно совпадают, то их останется лишь чуть больше 100000. В дальнейшем мы еще несколько уменьшим это число (при обнаружении несколько отличающихся, нечётких дубликатов).

2.4. Поиск похожих композиций

Мы в первой главе уже рассмотрели меру tf-idf и способы её применения для оценки подобия текстов. Попробуем теперь применить эти знания на практике.

Для начала все тексты были токенизированы методом простого разбития по пробельным символам, из полученных слов были исключены цифры и спецсимволы (кроме дефиса), затем из них были исключены априорные стоп-слова русского языка (список таких слов был взят из https://github.com/stopwords-iso/stopwords-ru).

Процесс стемминга был проделан несколько раз для разных стеммеров. Стеммер Портера, хоть и работал очень быстро, показал худшее качество, чем MyStem.

2.4.1. Практический опыт использования MyStem

В отличие от стеммера Портера, алгоритмы и реализация которого есть в свободном доступе, MyStem поставляется в виде проприетарной консольной утилиты с закрытым исходным кодом и специальной лицензией. Касательно самого алгоритма есть некоторые пометки, но все подробности неизвестны.

Консольный интерфейс MyStem и так не очень удобно использовать, а т.к. мы хотим интегрировать MyStem в нашу систему, то пришлось написать обвязку вокруг него. Из-за ограничений системы, требовалось проводить процедуру стемминга для текста каждой песни отдельно, но оказалось, что MyStem не обладает поточным вводом-выводом, т.е. он начинает свою работу только после того, как входной поток был закрыт на запись, что приводит к необходимости создавать новый процесс с MyStem для каждой композиции. Альтернативно, можно создать процесс один раз и передать ему адрес папки с текстами композиций, но создание такого большого числа временных файлов не лучшим образом сказывается на производительности.

Следующая проблема, с которой пришлось столкнуться – предел потока ввода MyStem в 4 Кб, т.е. после создания процесса MyStem, когда мы начинаем писать на переопределённый поток ввода, то получается записать только 4 Кб данных, а потом поток блокируется до тех пор, пока MyStem не прочитает записанные данные. Напомним, что MyStem начинает читать данные только после явного закрытия входного потока. Таким образом, попытки обработать вышеописанным способом песни с текстами длиной, превышающей 4 Кб, приводят к дедлоку. Оказалось, что число таких композиций крайне невелико, для русского языка их всего 14. Терять такие композиции, однако, не хочется. В качестве обхода указанной проблемы тексты песен делились на блоки, которые не превышают 4 Кб. Деление на блоки происходило по пробельным символам, т.е. ситуации, когда начало слова попало в один блок, а конец – в следующий, исключались.

Полученные на данном этапе тексты были сохранены, т.к. вызовы MyStem и сама операция стемминга достаточно тяжеловесны.

2.5. Рекомендации на основе tf-idf

После предыдущего шага для каждой песни мы получили массив слов, приведённых к одной форме и без априорных стоп-слов соответствующего языка. Количество разных слов в полученных массивах достаточно велико, оно приближается к 60000. Большинство этих слов встречаются всего лишь в нескольких текстах, но влияют на размерность tf-idf матрицы, поэтому разумно ограничивать число слов в словаре и строить матрицу только для словарных слов.

Если взять в рассмотрение только те слова, которые встречаются более чем в 100 композициях и не более, чем в 50% композиций (те, что встречаются чаще, будем считать нашими апостериорными стоп-словами), то число уменьшится до 5734, что является приемлемым показателем. Таким образом, мы можем из массива слов каждой песни исключить слова, не встречающиеся в

словаре, существенно сократив длину песен, не потерял при этом в качестве данных.

Далее тексты следует сгруппировать, т.к. среди текстов имеется большое число дубликатов. Тексты будем считать идентичными, если массивы их слов поэлементно совпадают. Также есть возможность использовать чего-то типа расстояния Левенштайна для более точного определения дубликатов (чтобы допускать возможность некоторого небольшого числа несовпадающих элементов), но такой подход сложнее в реализации и требует большего числа вычислительных ресурсов, в текущем подходе при разном числе слов в текстах тексы сразу же признаются разными.

Убрав таким образом нечеткие дубликаты (но, как было описано, они имеют шанс попасть в итоговую рекомендацию), мы получили 70408 уникальных текста, для которых посчитали tf-idf. Из-за того, что слова в песне повторяются, а припев песни может звучать много раз, может случиться сдвиг фокуса в сторону слов из припева, поэтому в качестве tf было взято не число вхождений слова в текст, а логарифм этого значения. Это позволило усреднить важность слов и привело к более качественным рекомендациям (если брать не логарифм, а само значение, то в некоторых вырожденных случаях весь текст песни может представляться одним словом, которое наиболее часто звучит в припеве).

Для композиций было вычислено косинусное подобие по описанному в первой главе способу. Т.к. в данном случае нет зависимости по данным, этот алгоритм хорошо распараллеливается (функция нахождения косинусного подобия коммутативна, т.е. вектор А косинусно подобен на вектор Б ровно настолько, насколько вектор Б - на вектор А, то считать можно не для каждого вектора с каждым, а только для половины).

Самыми похожими, очевидно, являются одинаковые композиции, поэтому при построении рекомендаций требуется удалить те, косинусное подобие которых превышает какое-то пороговое значение (в данной системе это 0.7). Итак, после фильтрации для каждой композиции были рассчитаны 10 самых близких к ней, эти данные были сохранены в хранилище.

Была написана консольная утилита, позволяющая для каждой композиции найти 10 похожих на неё.

```
Noize MC - Кантемировская

ST - Метрополитен
Blondrock - Чувства и гитары
Чиж & Co - Попутная песня
Анимация - Метро
Корни - Что-то ещё
Мастер - дайте свет!
GUF - Metropolitan Mail
Arinna - Наверное ты
Виктор Королев — Базар-вокзал
```

Рисунок 2.4. Рекомендации для композиции “Кантемировская”

```
Чиж & Co - Вечная молодость

Разные Люди и Чиж & Co - Вечная молодость
Кино - Восьмиклассница
Юлия Началова - Ах, школа, школа!
Леонид Фёдоров - Было
Алсу - Последний звонок
Chelsea (Челси) - Последний звонок
ВИА «Лейся, песня!» - Конопатая девчонка
SchoolRadio - девушка на миллион
Коррозия металла - Он не любил учителей
```

Рисунок 2.5. Рекомендации для композиции “Вечная молодость”

Как видно из наиболее похожих, такой простой, казалось бы, подход способен получить действительно похожие композиции. В первом случае тематика связана с метро, во втором – со школой.

Сам вывод утилиты не является рекомендацией, отправляемой клиенту. Из всех рекомендуемых случайным образом выбирается одна и уже идёт непосредственно на прослушивание (дополнительно у сервера присутствует информация об истории прослушанных композиций, чтобы не рекомендовать уже прослушанный контент и не зацикливаться).

Попытаемся обозначить плюсы и минусы построенной системы. Начнем с минусов:

- Размерность корпуса английских текстов не позволяет применить этот подход на них, т.е. система неустойчива к масштабированию,
- Из прошлого пункта вытекает, что система требует много ресурсов и для своей работы. Даже при хранении чисел с плавающей точкой одинарной точности матрица tf-idf занимаем много места в оперативной памяти,

- Рекомендации кажется релевантными, если у текста есть какие-то центральные слова, в противном случае схожесть даже с топ-10 похожих может быть около нулевым,
- Система не учитывает музыкального содержания песен,
- Рекомендации распространяются только на песни с текстом,
- (Опционально) Никак не учитывается опыт других пользователей.

Плюсами системы будут:

- Это действительно работающая система, позволяющая рекомендовать свежие композиции, до этого неизвестные пользователю,
- Процесс построения рекомендаций очень быстрый, не требуется переобучать модель с течением времени,
- Модель устойчива к росту числа пользователей,
- Модель служит хорошим base line для других систем рекомендаций.

Как понятно по построениям, рекомендации, которые будут получены на основе похожести текстов композиций, не будут персонализированными. Для получения персонализированных рекомендаций требуется делать вывод на основе уже одобренных пользователем композиций. Т.к. эти данные в общем случае не являются публичными и не могут быть получены от авторов сервиса “Яндекс.Музыка”, то эксперименты проводились для автора данной работы на основе его личной истории прослушивания.

2.6. Рекомендации на основе метода опорных векторов

Как было обосновано в первой главе, задачу рекомендации можно рассмотреть, как задачу бинарной классификации контента, где в один из классов попадает релевантный для пользователя контент, а в другой – нерелевантный. Такую задачу можно решать через метод опорных векторов (SVM) с линейным ядром.

Для этого мы берём уже просмотренный пользователем контент, делим его на два класса, преобразовываем в вектор из 0 и 1. Т.к. этот контент был классифицирован непосредственно пользователем, то в такой оценке можно быть уверенным и поставить ей больший вес. Далее, мы из не просмотренного пользователем контента выбираем некоторое число контента и ставим ему метки случайно. Уверенность в поставленных метках невелика, поэтому вес у таких оценок будет гораздо ниже, чем у пользовательских.

На полученных таким образом данных мы можем обучить SVM. Для этого в качестве X мы берём вектор tf-idf (согласно способу, описанному в

предыдущих пунктах), а в качестве Y – метку, причём с каким-то весом. Затем мы используем модель, чтобы вычислить метки (и их вероятности) для всей матрицы tf-idf и сортируем их по убыванию вероятности оказаться понравившимися.

Полученный вектор вероятностей идёт в основу построения рекомендаций. Стоит заметить, что из-за большого веса у одобренного пользователем контента у него, очевидно, будут и высокие вероятности попадания в класс 1, поэтому уже знакомый пользователю контент требуется отфильтровывать.

2.6.1. Проведение вычислительного эксперимента. Особенности практической реализации

Как и у многих алгоритмов, у SVM есть немалое число параметров, которые можно варьировать. В нашей системе использовалась реализация алгоритма из библиотеки машинного обучения Accord.Net, все дальнейшие рассуждения будут вестись в терминах этой библиотеки.

Одним из основных параметров, которые требуется выбрать, является Complexity (также C), который отвечает за обобщающие способности модели. Чем C ниже, тем более “осторожной” будет модель, и тем меньше контента будет отнесено в класс потенциально пригодных для рекомендаций. В утрированном случае модель будет настолько консервативной, что будет одобрять только априорно одобренный контент. В случае слишком большого C модель, наоборот, можем слишком мало внимания уделять оценкам пользователя. В данной системе было эмпирически выяснено, что число 100 позволяет сохранить баланс между консервативностью и излишней обобщающей способностью. При таком значении получившиеся рекомендации для пользователя будут новыми, но не будут чересчур неожиданными.

Еще одним значением параметра является отношение числа элементов в одном классе к элементам в другом классе. В нашей системе было выдвинуто предположение, что количество потенциально одобренного контента гораздо ниже, чем отвергнутого. На практике оказалось, что значение параметра 0.01 (разница на два порядка между мощностями представителей понравившихся и нет в сторону преобладания последних) отлично подходит и отражает реальную картину.

Заметим, что мы апеллировали к наличию у нас уже одобренного и неодобренного пользователем контента, но не было сделано никаких пояснений по поводу источника такого деления. В большинстве систем с подсистемами рекомендаций у пользователя присутствует возможность явно проставить эту

оценку. Во-первых, однако, такое действие не является обязательным, т.е. далеко не весь просмотренный контент будет размечен, во-вторых, эти данные могут быть недостоверными, в-третьих, это далеко не единственный источник данных.

Что касается упомянутой недостоверности, то тут возникает интересная уловка с точки зрения поведенческой психологии. С одной стороны, у нас есть непосредственные отметки от пользователя, а с другой стороны - есть реальная картина того контента, что он просмотрел. Т.е. может так оказаться, что пользователь в некотором смысле сам не знает того, что хочет. Получается, что если руководствоваться только теми оценками, что он лично проставил, то может так статься, что наша модель будет отражать не пользователя, а тот образ, который он транслирует. Образ, который может отличаться от реальных действий пользователя.

В любом случае, помимо явных оценок можно извлекать еще неявные данные. Например, отличным показателем будет длина прослушанной части композиции. Очевидно, что если пользователь пропустил композицию в течении её первых 20 секунд (при этом сама композиция имеет гораздо большую длину), то рекомендация не релевантная. С другой стороны, при прослушивании “от и до” можно смело говорить о положительной оценке рекомендации.

Можно также на основе истории прослушивания спекулятивно поднимать вес у других песен исполнителя, несколько композиций которого были положительно оценены. Вес композиций тех исполнителей, которые были явно негативно оценены, требуется тоже понизить (очень важно минимизировать число нежелательных рекомендаций, поэтому явные негативные оценки будут иметь больший вес, чем позитивные).

Вопрос изначального выбора контента для рекомендаций (т.е. способы рекомендации для совершенно новых пользователей) пока обсуждать не будем, затронем эту тему позже. Заметим, что на данный момент не удалось найти способа получить из Яндекс.Музыки подробной информации об истории прослушивания. Те композиции, которые появляются в разделе “История”, были прослушаны пользователем полностью, но нельзя найти те, которые не были дослушаны до конца. Такая информация, однако, сохраняется, на сервер отправляются все действия, совершаемые пользователем. Для аутентифицированного пользователя доступно лишь получение информации об композициях, которые он пометил как “мне нравится” или “не рекомендовать”.

Была написана консольная утилита, которая на вход принимала очередь прослушанных пользователем композиций и обучалась на них. По описанным выше причинам предполагалось, что эта очередь была прослушана

пользователем полностью, поэтому все композиции попали в положительный класс. Далее весь контент классифицировался и из 50 самых релевантных выбирались 5 композиций для рекомендации случайным образом. Эти композиции и рекомендовались пользователю. У пользователя есть возможность каждой композиции поставить одну из 4-х оценок: “не рекомендовать”, “пропустить”, “прослушать полностью” и “поставить отметку “мне нравится”. Получив от пользователя оценки для каждой рекомендации, модель доучивается и по описанной схеме генерируем еще 5 рекомендаций. Действия повторяется, пока пользователь не захочет выйти.

Количество контента, которое подмешивалось со случайными оценками превышало количество оценённого контента в 20 раз, в расчёт брались только последние 200 одобренных композиций, веса были такие:

- Если композиции явно была поставлена отметка “Не рекомендовать” – 1.2, при этом топ самых похожих на неё композиций попадает в специальную группу dislike hits,
- Если композиции явно была поставлена отметка “Мне нравится” – 1.0, при этом топ самых похожих на неё композиций и некоторое число композиций того же исполнителя попадают в специальную группу like hits,
- Если композиция попадает в группу dislike hits, то ставим ей негативную оценку с весом 0.6,
- Если композиция попадает в группу like hits, то ставим ей позитивную оценку с весом 0.25,
- Если композиция была полностью пропущена, то ставим ей вес 0.3 и негативную оценку,
- Если композиция была полностью прослушана, то ставим ей вес 0.25 и позитивную оценку,
- Если композиция была отобрано случайно, то ставим ей вес 0.05 и относим её равномерно к случайному классу.

Также принудительно не рекомендовались те композиции, которые недавно встречались в рекомендациях, а также была добавлена функциональность, которая ограничивала композиции одного и того же исполнителя и не давала им встречаться слишком часто.

```
Пикник - Телефон: Liked!
Браво - DJ Фантомас: Skipped!
Аквариум - Пока не начался джаз: Liked!
Полюса - Танцы войны: Skipped!
Аквариум - Не трать время: Liked!

Preparing new recommendations
Recommendations prepared in 00:00:17.3239899

Ленинград - Группа крови: Listened!
Сплин - Свет горел всю ночь: Liked!
масло черного тмина - 83.55:  Skipped!
Виктор Цой - Закрой за мной дверь, я ухожу: Liked!
Браво - Сон - обман:  Listened!

Preparing new recommendations
Recommendations prepared in 00:00:16.8347361
```

Рисунок 2.6. Пример вывода системы, которой на вход изначально была дана очередь из песен группы “КИНО”

Попытаемся дать оценку построенной системе и обозначить её сильные и слабы стороны. Начнём с минусов:

- Процедура выбора весов для оценок нетривиальна. В случае описываемой системы был проведён подбор этих гиперпараметров сеткой и кросс-валидацией. Т.е. был подготовлен достаточно большой пласт контента, который априорно нравится автору (построенный на основе его личной истории, топа песен всех исполнителей, чьи композиции представлены в истории, а также лучших песен некоторых жанров (по версии Яндекс.Музыки)), все эксперименты проводились на этом датасете и заняли длительное время. Однако сложно оценить качество весов, ведь существует, вероятно, гораздо лучшие наборы гиперпараметров для этой задачи.
- Не решена проблема рекомендации новым пользователям.
- Необходимо часто доучивать модель, что может сказываться на производительности. Это не является критичным, но стоит это отметить.

Плюсы системы:

- По качеству рекомендаций значительно превосходит систему на tf-idf. В случае первой системы рекомендации проводились только на основе подобия контента, в данном случае рекомендации регулируются ещё и за счёт пользовательских реакций.
- Выбор весов и описанные правила спекулятивного подъёма и штрафов позволяют получить качественные рекомендации, релевантные и с точки зрения музыкального вкуса.
- Система позволяет для каждого пользователя строить персональные рекомендации, учитывающие конкретно его опыт и предпочтения.

- Система устойчива к добавлению нового контента. Расширение фонотеки не приводит к квадратичному увеличению требуемых мощностей.
- Система лояльна к новому либо к уникальному контенту, который не похож на существующие. За счёт реакций и набора дополнительных правил у каждой композиции есть шанс попасть в рекомендованные.

Теперь, когда мы имеем две системы, мы можем сделать их сравнение и проверить те показатели, которые удалось улучшить. Даже по примерам рекомендаций видно, что их качество и релевантность существенно увеличились. Планомерно возросла и сложность системы, теперь на серверной стороне дополнительно нужно хранить не только историю прослушивания, а еще и реакции, и обученную модель для каждого пользователя. С другой стороны, из-за меньших размеров используемых матриц (мы хранили связь каждого контента с каждым в первом случае, а теперь для каждого контента хранится лишь его tf-idf вектор, т.е. это число на 1-2 порядка меньше) это может не сильно сказаться на вычислительных ресурсах. У нас по-прежнему никак не используется опыт других пользователей, зато мы решили проблему с новым контентом и получили возможность создания персональных рекомендаций!

2.7. Получение данных о других пользователях

Несколько раз был подчёркнут тот факт, что построенные рекомендательные системы не использует опыт других пользователей и строят рекомендации для каждого пользователя в изоляции от других. Разумеется, такой подход проще, он не требует хранить и анализировать большое число данных, однако дополнительные данные позволят построить систему с использованием подхода collaborative filtering, описанного в первой главе.

Не весь просмотренный контент был получен через систему рекомендаций, ведь у пользователя еще есть возможность находить контент через поиск, смотреть его по прямым ссылкам (полученным, например, от других пользователей через социальные сети) или перекрёстным ссылкам (переход от к другому альбому того же исполнителя и т.д.). Таким образом, мы можем учитывать естественное поведение пользователей и анализировать на основе их очередей прослушивания максимально подходящую следующую композицию для текущего пользователя.

В сервисе Яндекс.Музыка нет способа для текущего пользователя получить подробную историю, соответственно, невозможно получить подробную историю и для произвольного пользователя. Однако, есть возможность для произвольного пользователя по его идентификатору получить

список композиций с отметкой “мне нравится”, отсортированный по дате добавления композиции в этот список.

На первый взгляд кажется, что неверно будет использовать данные о понравившихся композициях в качестве очереди, но из-за того, что, во-первых, эти данные отсортированы в порядке добавления (это важно, т.к. эти списки могут быть большими и охватывать большой временной промежуток), и, во-вторых, т.к. эти композиции пользователь специально отметил, получается, что список “мне нравится” представляет собой подпоследовательность очереди пользователя, состоящую только из одобренных композиций. Получается, что такие данные даже более значимы, чем просто история прослушивания (конечно, в лучшем случае мы бы еще имели список из непонравившихся композиций, но получение такой информации непрозрачно и для текущего пользователя).

Итак, как было сказано, можно по идентификатору пользователя получить его фонотеку в случае, если он пожелал оставить её открытой (эта опция включена по умолчанию и не изменяется большинством пользователей). В отличие от идентификаторов композиций, которые представляют из себя монотонно увеличивающийся счётчик, что делает лёгким сбор данных о всех композиция, идентификатором пользователя является его логин, который он сам выбрал при регистрации.

Было замечено, что пользователь может подписаться на другого пользователя, тогда тот, кто подписался, будет отображаться в подписчиках у того, на которого подписались, и наоборот (подписавшийся будет иметь у себя ссылки на тех, на кого он подписан). Эта информация позволяет написать программный компонент, который будет осуществлять сбор логинов пользователей методом обхода в ширину и сохранять эти логины в хранилище. В качестве seed можно взять некоторые популярные аккаунты (например, у официального аккаунта “Яндекс” достаточно много подписчиков).

Оказалось, что данные о пользователях и их связях защищены лучше, чем данные о композициях. Был реализован crawler по описанному алгоритму (из соображений производительности он выполнял запросы пачками по 10000 за раз в несколько потоков), на запросы достаточно быстро начали приходить ответы с кодами 3XX, перенаправлявшими на страницу ввода каптчи. Из-за того, что все такие перенаправления игнорировались, IP-адреса попали в список заблокированных и достаточно быстро действие программы было парализовано. При любом запросе на любую конечную точку любого имеющего отношение к Яндекс.Музыке хоста возвращался код ответа 403 и страница следующего содержания:

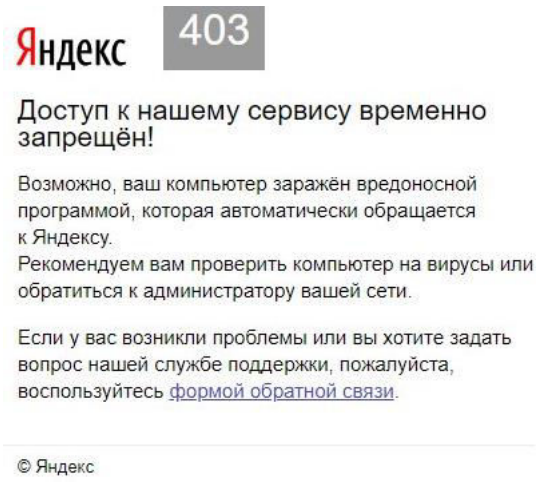

Рисунок 2.7. Страница с запретом дальнейших действий

Экспериментально выяснилось, что IP-адрес блокируется на около 25 часов, потом можно делать новые запросы. За несколько запусков программы удалось собрать логины 482306 активных пользователей с открытой фонотекой.

Был написан еще один программный компонент, который для каждого пользователя запросил его фонотеку и сохранил эти данные в хранилище.

```
[01:05:18 VRB] 480600 libraries processed (49 failed, 63 not found). 100 processed in 00:00:02.9668178
[01:05:23 VRB] 480700 libraries processed (49 failed, 63 not found). 100 processed in 00:00:05.4122938
[01:05:28 VRB] 480800 libraries processed (49 failed, 63 not found). 100 processed in 00:00:05.2752182
[01:05:31 VRB] 480900 libraries processed (49 failed, 63 not found). 100 processed in 00:00:02.9820399
[01:05:35 VRB] 481000 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.7758781
[01:05:39 VRB] 481100 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.7804661
[01:05:43 VRB] 481200 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.6564009
[01:05:46 VRB] 481300 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.0302740
[01:05:48 VRB] 481400 libraries processed (49 failed, 63 not found). 100 processed in 00:00:02.4823419
[01:05:51 VRB] 481500 libraries processed (49 failed, 63 not found). 100 processed in 00:00:02.7097679
[01:05:55 VRB] 481600 libraries processed (49 failed, 63 not found). 100 processed in 00:00:04.0025912
[01:05:58 VRB] 481700 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.2081463
[01:06:01 VRB] 481800 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.1774185
[01:06:07 VRB] 481900 libraries processed (49 failed, 63 not found). 100 processed in 00:00:05.9267338
[01:06:14 VRB] 482000 libraries processed (49 failed, 63 not found). 100 processed in 00:00:06.7913148
[01:06:17 VRB] 482100 libraries processed (49 failed, 63 not found). 100 processed in 00:00:03.0056734
[01:06:19 INF] All logins processed! Total: 482188, Not found: 63, Exceptions: 49. Retrying 49 errors
[01:06:22 VRB] 482200 libraries processed (49 failed, 63 not found). 100 processed in 00:00:05.0442183
```

Рисунок 2.8. Фрагмент вывода во время сбора информации о фонотеках

Данные об фонотеках пользователей защищены не так строго, как данные о связях между пользователями, вышеописанных проблем не возникало. Местами возвращались кода 3XX с переадресацией на страницы каптчи, но такие ответы игнорировались, а соответствующие логины добавлялись обратно в очередь для последующего перезапроса.

Итого были запрошены фонотеки 482306 пользователей. Сумма длин их списков композиций составила 75445871, т.е. в среднем у пользователя больше 150 одобренных композиций. У самого активного пользователя набралось 110691 одобренных композиций.

2.8. Анализ данных о композициях

Среди всех очередей всех пользователей оказалось 1012480 уникальных композиций, которые встречаются 5 и более раз. Рассмотрим самые популярные композиции:

Исполнитель и название композиции	Количество пользователей, одобривших композицию
Sting - Shape Of My Heart	23293
The White Stripes - Seven Nation Army	22289
Red Hot Chili Peppers - Snow (Hey Oh)	20696
Nirvana - Smells Like Teen Spirit	20226
Red Hot Chili Peppers - Can't Stop	20023
Limp Bizkit - Behind Blue Eyes	18740
Hozier - Take Me To Church	17662
Red Hot Chili Peppers – Californication	17649
The Cranberries – Zombie	17328
Linkin Park - Numb	17252

Таблица 2.4. Топ-10 самых частых композиций среди полученных списков

Исполнитель и название композиции	Количество пользователей, одобривших композицию
Океан Ельзи - Без бою	15381
Земфира - Искала	13654
Сплин - Романс	13408
Сплин - Моё сердце	13122
Би-2 - Молитва	12371
Мумий Тролль - Владивосток 2000	12121
Земфира - Хочешь?	11764
ДДТ - Это всё…	11666
Ляпис Трубецкой - Я верю	11580
Lumen - Гореть	11277

Таблица 2.5. Топ-10 самых частых композиций на русском языке среди полученных списков

2.9. Решение проблемы построения рекомендаций для новых пользователей системы

При построении обеих рассмотренных ранее систем были отмечены проблемы рекомендации для новых пользователей. Нам было нечего

предложить новым пользователям до тех пор, пока не наберётся критический минимум размеченных непосредственно ими композиций. Теперь, когда мы обладаем знаниями о частоте встречи некоторых композиций в очередях прослушивания пользователей, мы можем среди всех композиций те, которые являются очевидными хитами и которые можно сразу рекомендовать пользователю, т.к. с большой долей вероятности они не будут идти вразрез с его интересами. Вероятность того, что ему не нравятся абсолютно все хиты, крайне низка, но даже в таком случае мы можем на основе прослушивания им некоторого стартового числа композиций построить профиль пользователя и начать строить уже персональные рекомендации, а не рекомендации для “среднего” пользователя.

Также ранее уже несколько раз было упомянут факт случайного выбора композиций из некоторого топа рекомендаций, но этот вопрос широко не рассматривался. Было сказано, что рекомендация самой вероятной композиции (как и самого популярного хита) не представляется хорошим выбором, т.к. может способствовать зацикливанию или попаданию пользователя в “пузырь” рекомендаций, когда система будет вести себя достаточно консервативно и не будет рекомендовать ничего принципиально нового и свежего.

Рассмотрим подробнее процесс выбора следующей песни из некоторого списка композиций, где каждой композиции поставлена в соответствие некоторая вероятность p_i (и сумма вероятностей для всех композиций списка равна единице). Введём параметр t, который будем называть температурой (он будет лежать в промежутке от 0 до 1) и преобразуем вектор из p_i по следующему правилу:

$$q_i = \frac{exp\left(\frac{p_i}{t}\right)}{\sum_{j=1}^{N} exp\left(\frac{p_j}{t}\right)}$$

Далее мы моделируем 1 эксперимент с распределением по мульти-модальному закону. Композиция с полученным таким образом индексом идёт в рекомендации.

Температура в данном случае помогает сглаживать разницу в вероятностях исходов. При температуре = 1 исходы будут консервативными и ожидаемыми, температуры ниже будут приводить к более интересным результатам. Слишком низкий показатель температуры ставить нежелательно, результаты получаются слишком непредсказуемыми. В рамках построенных систем использовалось значение температуры 0.7.

2.10. Рекомендации на основе word embeddings и LSTM

Теперь, когда обладаем данными об опыте других пользователей, мы можем интегрировать эти знания в нашу систему рекомендаций. Если рассмотреть список одобренных композиций каждого пользователя как последовательность подряд идущих идентификаторов, то можно каждый идентификатор назвать словом, а саму очередь – текстом. Получается, что у нас есть корпус текстов на таком псевдоязыке, где слова являются идентификаторами композиций. Тогда задачу рекомендации можно рассмотреть, как задачу генерации текста по некоторым начальным словам (т.е. по предыдущим прослушанным композициям).

В первой главе были рассмотрены способы решения этой задачи аппаратом NLP. Для этого можно обучить word2vec модель на базе корпуса текстов, взять оттуда претренированные word embeddings и установить их в начальный слой нейронной сети.

Использовалась реализация word2vec из библиотеки Word2Vec.Net. В процессе работы было выяснено, что у библиотеки есть баги, поэтому пришлось склонировать и изменить её исходный код для получения верных векторов. В качестве фреймворка для построения нейронных сетей использовался Keras.Net с бекендом TensorFlow. Keras.Net – порт фреймворка Keras на платформу .NET, который в своей реализации всё равно вызывает python код из Keras, но делает это через систему обёрток, чтобы можно было синхронизировать среды выполнение python и .NET (для работы таких мезанизмов, как исключения или сборка мусора). Т.о. можно использовать абстракции из Keras в строго-типизированном стиле. Для того, чтобы узнать о наличии, имени или назначении параметра не нужно лезть в документацию, эта информация сразу доступна через систему типов. Это же касается и возвращаемых значений.

Для работы Keras.Net требует установленную версию python 3.7, Keras и TensorFlow.

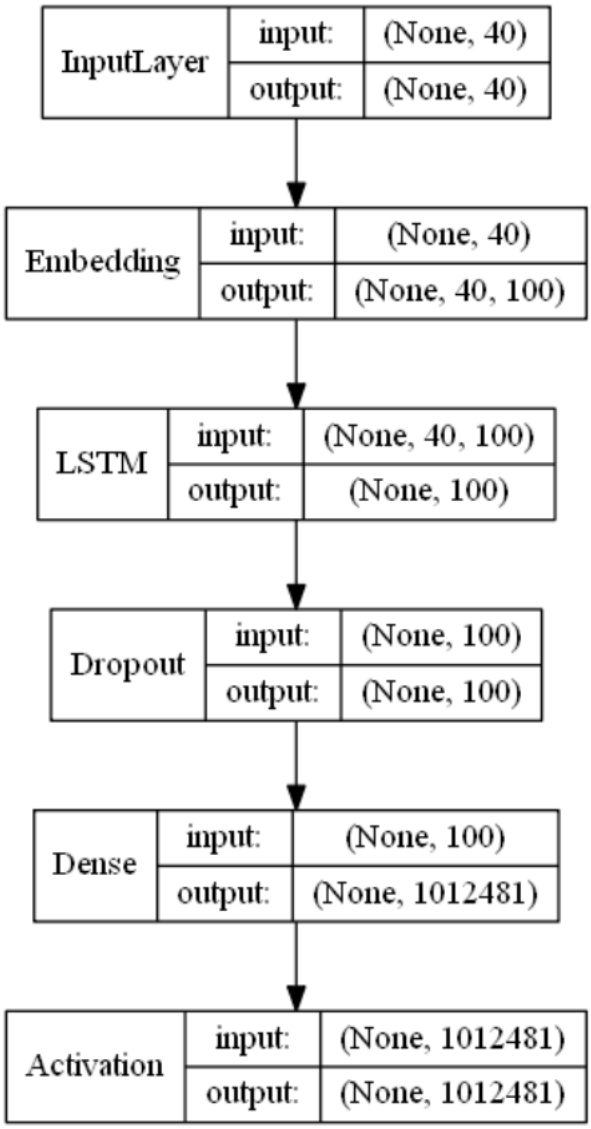

Рисунок 2.9. Архитектура используемой нейронной сети

В качестве X тренировочной выборки были взяты последовательности по 40 композиций, а в качестве Y – следующая за ними песня. Было обучено 30 эпох сети.

Для производства рекомендаций был использован способ с температурой, в качестве её значения было выбрано 0.7. За раз пользователь получает 10 рекомендаций.

```
7Б - Песни мои
КИНО - Печаль
Евгений Маргулис - Когда ты уйдёшь
Ария - Улица роз
Nautilus Pompilius - Дыхание
7Б - Жди
Гарик Сукачёв - Нулевой километр
КИНО - Красно - жёлтые дни
Nautilus Pompilius - Христос (Мне снилось, что...)
ЧайФ – Ошейник
```

Рисунок 2.10. Построенные рекомендации для пользователя, прослушавшего топ-5 композиций группы "Чиж и Со"

Дадим оценки построенной системе. Начнём с минусов:

- В рекомендации попадает только тот контент, который уже присутствует в очередях других пользователей, система не толерантна к новому или непопулярному контенту,
- Построенные рекомендации нельзя в полной мере назвать персонализированными, не используются пользовательские реакции,
- Из-за специфики пользовательских очередей, в которых зачастую подряд идут композиции одних и тех же исполнителей (иногда даже просто подряд идут все композиции одного альбома), в рекомендациях тоже могут встречаться несколько подряд идущих композиций одного исполнителя.

Плюсами такой системы будут:

- Хорошее качество рекомендаций из коробки, с очень большой долей вероятности рекомендованные композиции пользователю понравятся,
- Из-за того, что система базируется на реальных очередях, минимизировано появление неожиданных композиций (например, металла в очереди детских песен),
- Решена проблема рекомендации новым пользователям, получена модель, настроенная под среднего пользователя,
- Модель позволяет быстро получить рекомендации, не требуется частого переучивания.

Из указанных минусов один можно исправить достаточно просто: нужно установить лимиты на количество композиций одного исполнителя подряд (или, что ещё лучше, добавить правило, что один исполнитель должен появляться не чаще, чем 1 раз из 10). Также можно заметить, что предыдущая модель (на основе SVM) не имела соответствующих минусов, а плюсы новой модели закрывают минусы модели на SVM. Это приводит к мысли, что возможно построить гибридную модель, которая возьмёт в себя все плюсы обеих систем и закроет минусы системы на word embeddings и LSTM.

2.11. Создание гибридной модели

Важной частью персонализированных рекомендаций является тот факт, что не будут рекомендованы те композиции, которые пользователю не понравятся с высокой долей вероятности. Наша система на LSTM не получает никакой обратной информации от пользователя о реакции на рекомендации, т.е. можно сказать, что модель считает, что все рекомендации пользователю заведомо понравятся. Из этого следует, что в систему нужно добавить какой-то компонент-оракул, с которым модель будет “консультироваться” перед отправкой рекомендаций пользователю. Этот компонент знает пользовательские предпочтения и может с некоторой вероятностью сказать, что предложенная композиция пользователю не понравится. Выступая в качестве фильтра, этот компонент также будет получать обратную связь от пользователя с реакциями на рекомендации, что позволит ему подстраиваться к пользовательским предпочтениям.

Ранее мы уже создали систему с поддержкой пользовательских реакций, т.о. система на основе SVM и станет этим компонентом-оракулом. Более того, этот компонент не будет просто удалять заведомо нерелевантные рекомендации, он будет их заменять заведомо релевантными. Заметим, что модель на основе SVM также умеет рекомендовать новый и уникальный контент, который может не встречаться в очередях других пользователей. Получается, что построенная гибридная модель решает проблемы обеих построенных систем и собирает их плюсы вместе.

Итоговая схема построенной системы продемонстрирована на рисунке 2.11. Видно, что последним шагом на ней отображён пост-процессинг. В процессе создания рекомендации генерируется с профицитом, на шаге пост-процессинга часть из них отфильтровывается, часть заменяется, например:

- Среди созданных рекомендаций могут быть дубли, их требуется удалить. Под дублями понимаются не только те, что полностью идентичны, но ещё разные версии одной и той же композиции. Такие дубли мы научились искать еще при построении системы на tf-idf,
- Могут быть дубли с тем, что пользователь уже недавно прослушал, также их удаляем,
- Может так статься, что в рекомендации проникло много композиций от одних и тех же исполнителей. Композициям одного исполнителя разрешено появляться не чаще, чем раз в 10 рекомендаций, поэтому удаляем лишние,
- За время с последнего сбора данные могли измениться, композиции могли быть отозваны правообладателем или у них мог смениться идентификатор, поэтому проверяем, что все композиции из

рекомендаций доступны пользователю. Если нет, то оставляем только доступные.

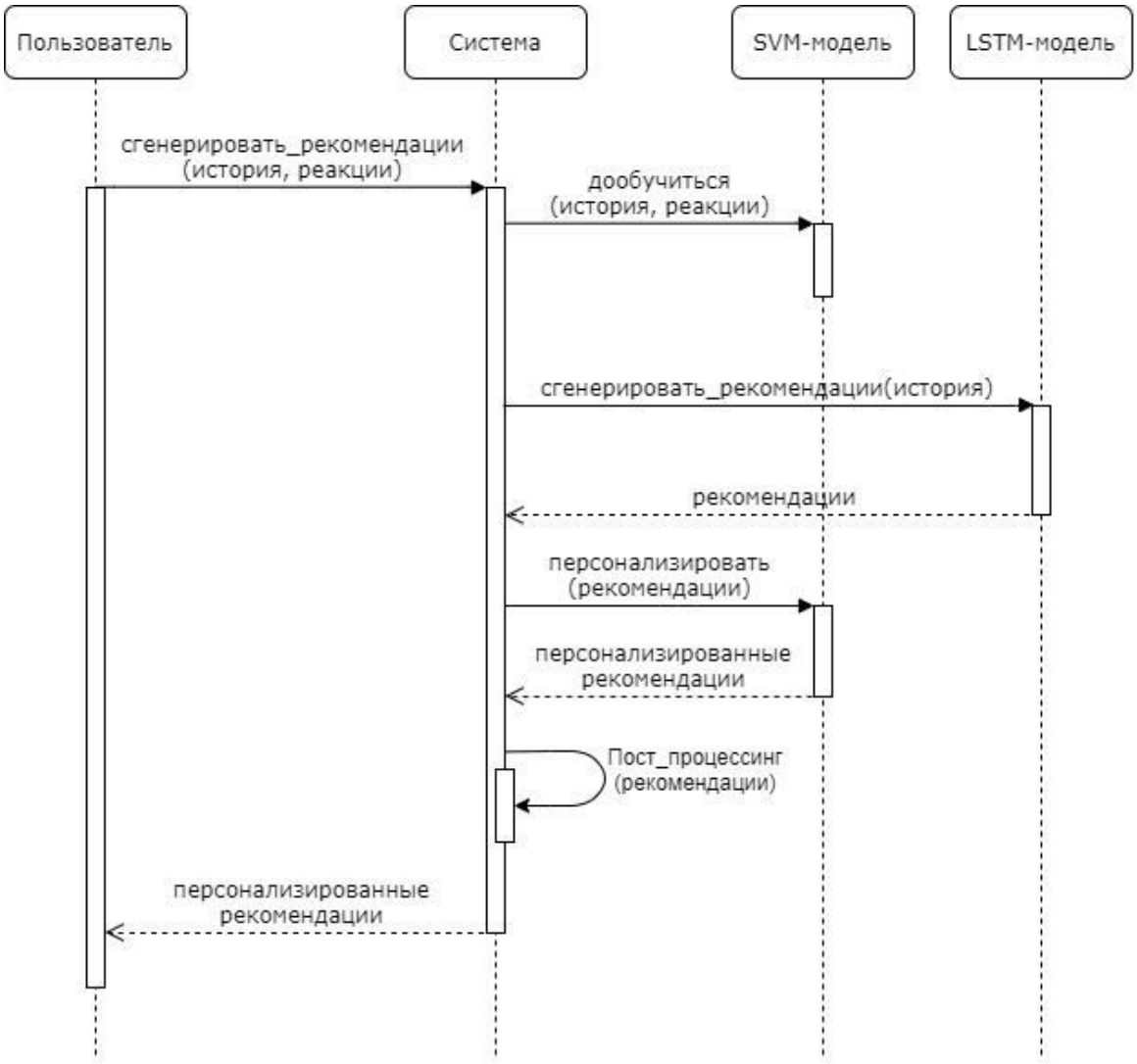

Рисунок 2.11. Схема гибридной модели

2.12. Выводы

В данной главе были проведены следующие исследования, получены следующие результаты:

- Дано описание сервиса Яндекс.Музыка, изложены способы получения информации обо всех представленных на нём композициях, эти данные собраны и проведён их анализ,
- Построена и проанализирована модель на основе tf-idf,
- Построена и проанализирована модель на основе метода опорных векторов, позволяющая получить рекомендации в т.ч. на основе

пользовательских реакций, изложен опыт оптимизации весов и обучения модели,

- Описан способ сбора данных о других пользователях, изложен опыт такого сбора, получены и проанализированы сами данные,
- Использованы механизмы NLP, построена модель на основе word embeddings и LSTM, использующая полученные данные. Модель проанализирована, выделены её слабые и сильные стороны,
- Построена гибридная модель, комбинирующая в себе модели на основе SVM и LSTM, приведена её схема, описан полный цикл работы.

ГЛАВА 3. СОЗДАНИЕ МОБИЛЬНОГО ПРИЛОЖЕНИЯ-КЛИЕНТА РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

3.1. Необходимость в собственном клиентском приложении

Можно заметить, что при росте сложности выстраиваемых рекомендательных систем планомерно рос также требуемый уровень контроля за приложением-клиентом, с помощью которого можно пользоваться рекомендательной системой.

В случае tf-idf ничего специфического не требовалось, хватало базовой информации об очереди прослушивания, но в рекомендательной системе на основе SVM ситуация изменилась, для построения персонализированных рекомендаций нам стала критически важна обратная связь от пользователя, выражаемая как очевидными для него отметками “Мне нравится” и “Не рекомендовать”, так и менее очевидными данными для сбора: информации, что какая-то композиция была пропущена, какая-то была промотана, а третья – прослушана полностью.

Помимо обратной связи на рекомендации, клиентское приложение должно собирать данные о пользователе даже в то время, когда он не пользуется непосредственно рекомендациями, т.е. когда он набирает что-то в поиске, переходит между разными исполнителями, открывает что-то по прямым ссылкам и, конечно, когда он сам слушает какой-то контент не из рекомендаций.

Вся эта функциональность присутствует в официальных приложениях-клиентах Яндекс.Музыки и используется для улучшения качества рекомендаций, но эта информация недоступна даже о самом авторизированном пользователе, речь даже не идёт о том, чтобы это было открытой информацией, доступной для автоматизированного сбора, подобно информации о композициях и примитивной информации о предпочтениях пользователей, сбор которой был подробно описан в главе 2.

Помимо функции сбора интересующей нас информации, впоследствии мы бы хотели использовать приложение-клиент для непосредственно прослушивания построенных рекомендаций.

Как понятно из описания, ни один из официальных приложений-клиентов не соответствует выдвигаемым требованиям, поэтому в рамках работы над диссертацией была также проведена работа по созданию собственного клиентского приложения.

3.2. Выбор технологии для клиента

Первый выбор, который встаёт перед началом создания клиентского приложения – это выбор платформы, где оно будет работать:

- Веб-приложение, работающее в браузере,
- Настольное приложение, работающее под управлением операционной системы, которую тоже надо выбрать,
- Мобильное приложение-клиент, дополнительно нужно выбрать платформу.

Исходя из реалий современного мира, а также специфики решаемой задачи, выбор был однозначно сделан в пользу мобильного приложения. Веб-версия кажется слишком громоздкой, а настольное приложение в задачах прослушивания музыки явно проигрывает мобильному, тем более, что всё у большего числа людей есть смартфоны и доступ к мобильному интернету. Очевидным следующим выбором является выбор платформы:

- Нативное Android-приложение на языке, компилируемом в Java-байт-код,
- Нативное iOS приложение, написанное на Objective C или Swift,
- Кроссплатформенное приложение на основе WebView (написанное, например, на React Native)
- Кроссплатформенное приложение, не основанное на WebView.

При разработке приложений под управлением iOS требуется иметь другую продукцию от компании Apple, их проприетарный компилятор не позволит скомпилировать приложение на компьютере под управлением Windows. Поскольку у автора нет ни одного устройства от компании Apple, вопрос кроссплатформенности остро не стоял, хотя, несомненно, кроссплатформенность была бы плюсом. Дополнительно, т.к. вся рекомендательная система написана на языке C# в строго-типизированном стиле, хотелось бы тоже использовать C# и на клиенте, чтобы, во-первых, использовать уже известный и крайне удобный инструмент, во-вторых, для задач переиспользования уже написанного кода.

Платформа .NET динамично развивается и предоставляет несколько способов написания Android-приложений:

- Написание нативных Android-приложений с использованием всех Java классов и объектов за счёт механизма Interop, но также с возможностью использовать любой другой .NET код (Xamarin.Android),
- Написание нативное iOS приложения со всеми теми же свойствами (Xamarin.iOS),
- Написание кроссплатформенного приложения, для которого будет возможность собрать его и под Android и под iOS, где будет

возможно переиспользование любого .NET кода, но не будет прямой возможности использования нативных компонентов. При этом подходе пользовательский интерфейс создаётся на основе абстракций, предоставляемых фреймворком Xamarin.Forms, а при реальной отрисовке эти абстракции уже преобразуются непосредственно в нативные компоненты. Очевидным недостатком является возможность использования только тех компонентов и только тех их свойств, которые поддерживаются в обеих системах.

Изначально был сделан выбор в пользу Xamarin.Forms, т.к. это тот путь, которого компания Microsoft рекомендует придерживаться.

3.3. Опыт использования Xamarin.Forms

Xamarin.Forms действительно позволил за достаточно небольшой промежуток времени начать разработку приложения и неплохо продвинуться в этом. Будучи построенных для использования шаблона MVVM, удавалось отделить логику отображения от бизнес-логики приложения, поэтому сама архитектура выглядела чистой и всё приложение в целом было модульным и юнит-тестируемым.

Однако по мере усложнения функциональности стало понятно, что возможности по настройке нативных компонентов скудны и требуют излишних усилий, а также то, что наборы готовых компонентов не покрывают все поставленные цели в области функционирования пользовательского интерфейса.

Более того, производительность решения оставляла желать лучшего. Время открытия каждой страницы исчислялась несколькими секундами, что не соответствовало хорошим практикам с точки зрения UX. Для начала была проведена работа по оптимизации приложения, все действия, не имеющие непосредственного отношения к обновлению пользовательского интерфейса, были делегированы background потокам, чтобы максимально очистить UI поток.

Эти действия позволили достичь скорости загрузки страницы в полторы секунды, но не привели к отзывчивости приложения. Во время некоторых действий приложение не отвечало и не отображало никакого прогресса. С помощью профилировщика Xamarin Profiler, который интегрирован в IDE Visual Studio Enterprise, было выяснено, что UI поток полностью сосредоточен на выполнении кода Xamarin.Forms, т.е. библиотечного кода, на который у нас нет возможности повлиять.

Эти два факта привели к тому, что было принято решение пожертвовать кроссплатформенностью и переписать приложение на Xamarin.Android, получив при этом контроль за всеми процессами, как отвечающими за производительность, так и за всестороннюю настройку UI/UX.

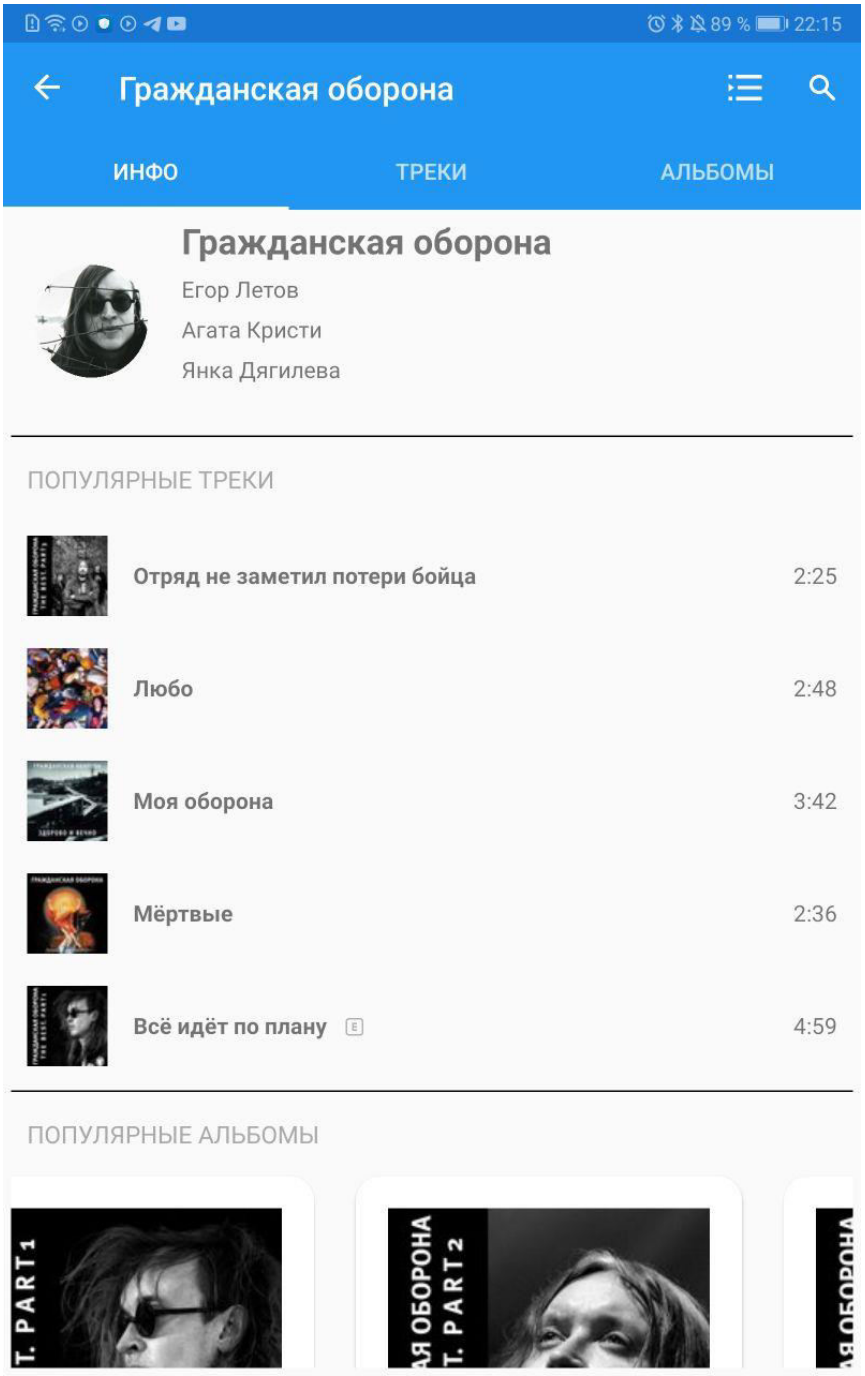

Рисунок 3.1. Снимок экрана приложения на Xamarin.Forms

3.4. Опыт использования Xamarin.Android

Как было сказано, у Xamarin.Android нет недостатков Xamarin.Forms, поэтому приложение было переписано с использованием этой технологии. Из-за родственности технологий некоторое количество кода получилось переиспользовать.

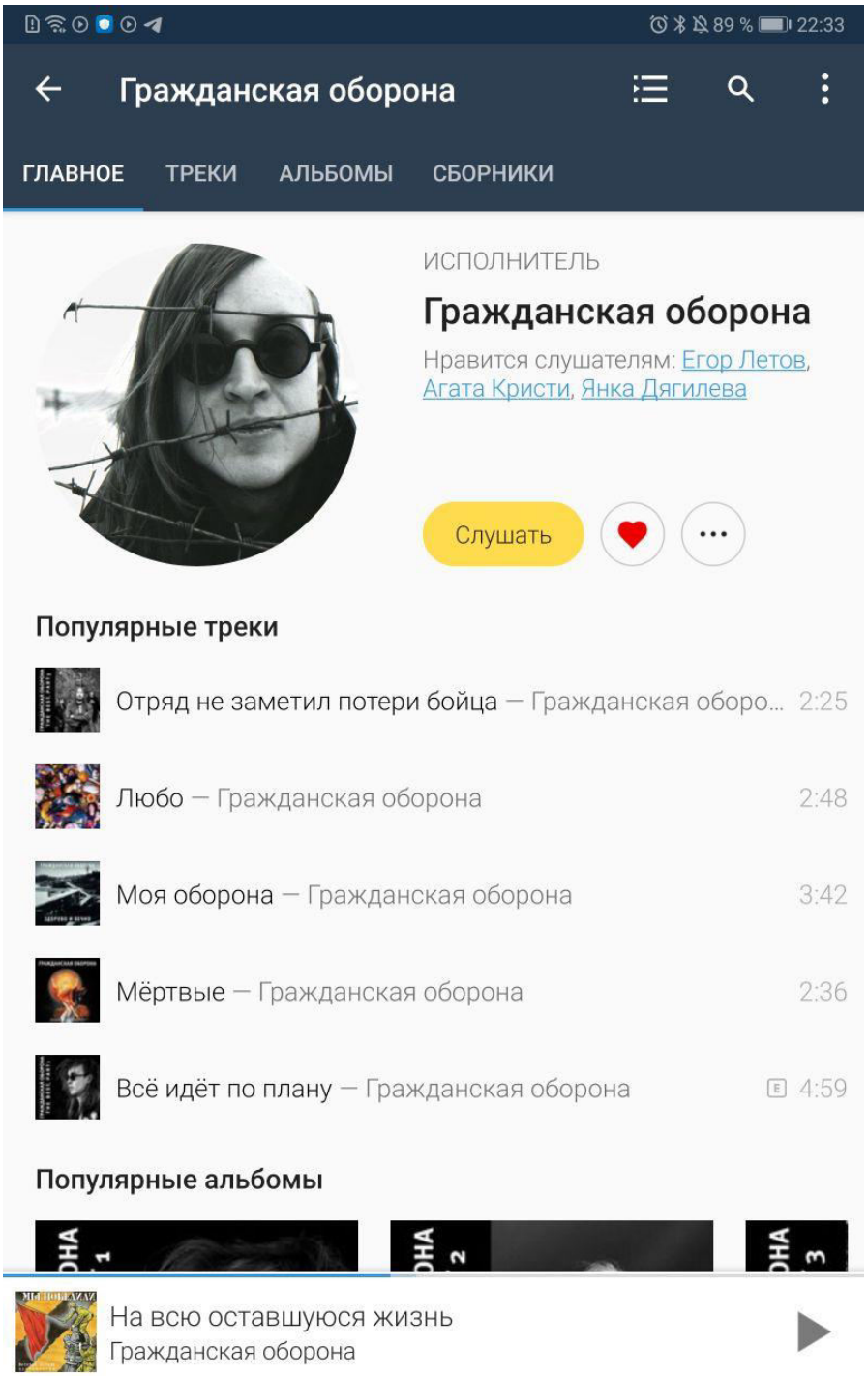

Рисунок 3.2. Снимок экрана приложения на Xamarin.Android

При разработке под Android шаблон MVVM не так популярен, как при разработке на основанных на XAML-разметке технологиях (где он, собственно, и был придуман), поэтому пришлось использовать сторонний фреймфорк для работы с MVVM (с поддержкой binding, command и converters) – Mvvm Cross. В процессе работы даже поучаствовать в разработке Mvvm Cross (это проект с открытым исходным кодом).

В качестве библиотеки для проигрывания композиций использовался ExoPlayer от компании Google (эта библиотека позволяет гораздо больше, чем стандартные способы воспроизведения звука на Android).

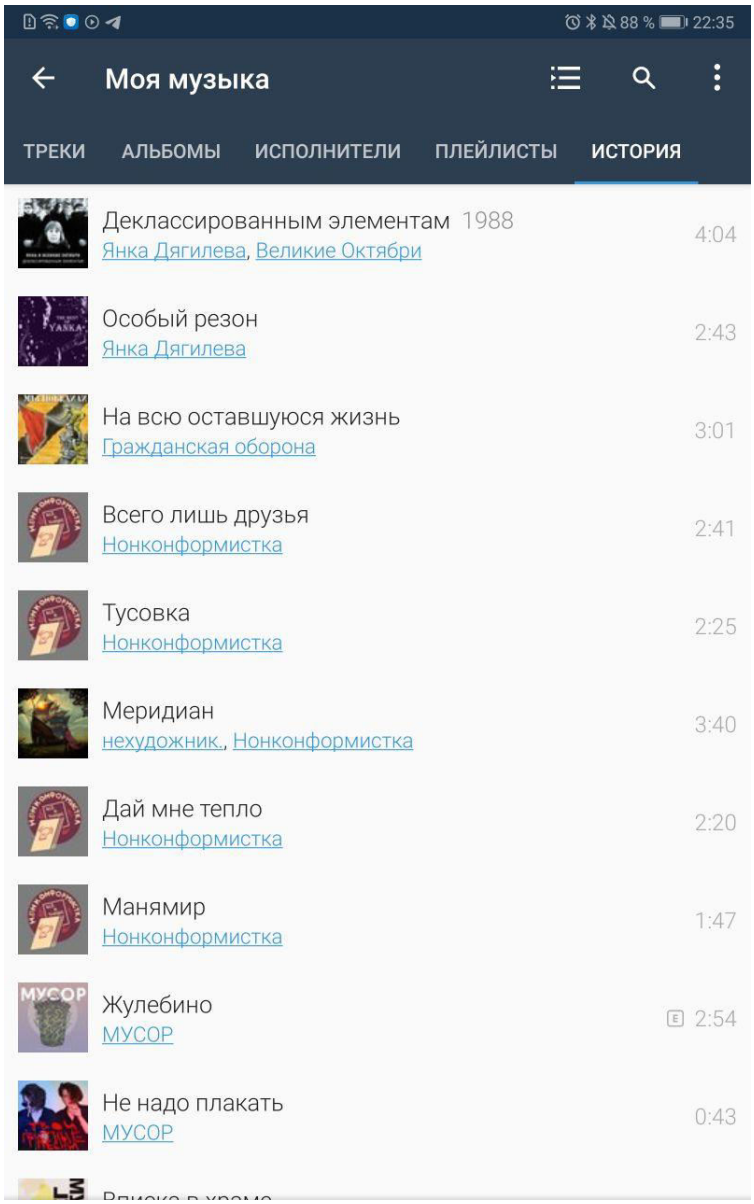

Рисунок 2.6. Снимок экрана приложения

3.5 Разработка мобильного клиента

В процессе разработки клиента для выяснения адресов конечных точек HTTP и форматов входных и выходных данных использовался следующий подход: с помощью инструмента для отладки и поиска уязвимостей в веб-приложениях Fiddler было организовано прослушивания траффика от веб-версии Яндекс.Музыки к серверу. Далее, открывались интересующие страницы в веб-версии и записывались форматы данных и адреса конечных точек. Затем предпринимались попытки интерпретации этих результатов (цели и содержания некоторых полей неочевидны).

По мере получения этих данных, в клиент добавлялась соответствующая функциональность. Однако, как было указано при формулировании требований, дополнительно перед отправкой запросов от клиента все данные дополнительно журналируются в локальную базу данных клиента.

При запросе рекомендаций клиент дополнительно передаёт сервису некоторые дополнительные данные из числа собранных.

3.6 Выводы

В данной главе были проведены следующие исследования, получены следующие результаты:

- Дано обоснование необходимости построения собственного приложения-клиента,
- Обосновал и сделан выбор в пользу мобильного приложения,
- Сделан осмотр возможных технологий, обоснован выбор решений, основанных на Xamarin,
- Изложен практический опыт построения приложения на основе Xamarin.Forms, описаны возникшие проблемы и предпринятые попытки по их решению
- Описана миграция на Xamarin.Android,
- Дано описание метода построения клиента,
- Спроектирован и реализован полноценный мобильный клиент для смартфонов под управлением ОС Android, реализующий функции навигации по исполнителям, альбомам, плейлистам и композициям; поиска по фонотеке, возможности поставить оценке композициям и предоставляющий возможность их прослушивания. Дополнительно клиент собирает данные, которые используются для улучшения качества рекомендаций.

ЗАКЛЮЧЕНИЕ

Таким образом, в рамках магистерской диссертации на основе изученной литературы рассмотрены и описаны подходы к разработке рекомендательных систем. Были изучены некоторые алгоритмы информационного поиска и методы машинного обучения.

Проведены эксперименты и создан программный компонент, который собрал все доступные тексты композиций сервиса “Яндекс.Музыка” и сохранил их в локальное хранилище. Был проведён анализ этих текстов, собраны некоторые статистики. Впоследствии этот компонент был расширен и собрал также персональную информацию о пользователях сервиса, касающуюся их очередей прослушивания.

Основные результаты работы состоят в следующем:

- Были предложены базовые модели построения рекомендательных систем для аудио-сервисов,
- Создана программная реализация предложенных моделей, описан практический опыт и указаны трудности, с которыми пришлось столкнуться,
- Проведён полный анализ построенных систем, для каждой модели выделены сильные и слабые стороны,
- Проведены вычислительные эксперименты, целью которых является сравнение построенных систем и демонстрация их работы,
- На базе этой информации предложена и реализована гибридная модель, которая лишена указанных недостатков,
- Для удобства сбора данных создано приложение-клиент для ОС Android, проанализированы возможные технологии, обоснован выбор Xamarin, описан процесс его создания приложения, изложен практический опыт и трудности, решённые при построении.

Дальнейшие направления исследования включают в себя применение различны архитектур нейронных сетей для улучшения качества рекомендаций, построение и обоснование методологии оптимизации гиперпараметров в модели на основе SVM, а также развитие мобильного клиента, добавление в него новой функциональности.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. François Chollet, Deep Learning with Python – Manning Publications, 2017. – 384 с.

2. Николенко С. Глубокое обучение. Погружение в мир нейронных сетей / С. Николенко, А. Кадурин, Е. Архангельская; под ред. С. Николенко. – Санкт-Петербург: Питер, 2018. – 481 с.

3. Грас Дж., Data Science. Наука о данных с нуля – O'Reilly, 2017. – 336 с.

4. Кристофер Д. Маннинг, Введение в информационный поиск – Вильямс, 2014. – 528 с.

5. J. Ramos, Using TF-IDF to Determine Word Relevance in Document Queries – 2003. – 4 с. Tech. rep.

6. T. Mikolov, I. Sutskever, K. Chen, G. Corrado, J. Dean, Distributed Representations of Words and Phrases and their Compositionality - 2013. – ACL, стр. 746–751.

7. Accord.Net Documentation. 2017. http://accord-framework.net

8. Keras.NET Documentation. 2020. https://scisharp.github.io/Keras.NET/

9. Документация MyStem. 2020. https://yandex.ru/dev/mystem/

10. Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, Yejin Choi, The curious case of Neural Text DeGeneration. – 2020. – ICLR 2020 Conference, 16 с.

11. Mvvm Cross Documentation. 2020. https://www.mvvmcross.com/