МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
им. Н.Э. Баумана

Факультет «Информатика и системы управления»
Кафедра «Систем обработки информации и управления»

Домашнее задание

по дисциплине «НИР по обработке и анализу данных»

Тема: «Рекомендательная система по подбору музыки для пользователей»

ИСПОЛНИТЕЛЬ: _Бабин В. Е._____
ФИО

группа ИУ5-32 _____
подпись

"__"_____2020 г.

ПРЕПОДАВАТЕЛЬ: __Гапанюк Ю. Е._
ФИО

_____
подпись

"__"_____2020 г.

Москва - 2020

Оглавление

Аннотация

Большинство рекомендательных систем используют методы фильтрации на основе содержания (данный метод также называют «content-based filtering») и коллаборативной фильтрации (данный метод также называют «collaborative filtering») для прогнозирования новых элементов, представляющих интерес для пользователя. Хотя оба метода имеют свои преимущества, по отдельности во многих случаях они не дают хороших рекомендаций. Объединяя сильные стороны каждого из методов, гибридная рекомендательная система может попытаться преодолеть их отдельные недостатки.

В НИР представлен один из способов объединения методов фильтрации на основе содержания и коллаборативной фильтрации. Первоначально будет использоваться метод фильтрации на основе содержания для улучшения существующих пользовательских данных (матрица пользователь–объект будет становится менее разреженной), а после использования метода коллаборативной фильтрации будут производиться точные прогнозы, на основе которых пользователям будут предоставляться рекомендации.

Введение

Научно-исследовательская работа (НИР) проводится в рамках выпускной квалификационной работы магистра на тему «Рекомендательная система по подбору музыки для пользователей» (далее просто рекомендательная система). Так как на момент написания НИР ВКРМ ещё не реализована польностью, будут приведены уже имеющиеся результаты и те способы, при помощи которых они были достигнуты, а также приведен план дальнейшей работы.

Системы рекомендаций помогают преодолеть информационную перегрузку, предоставляя персонализированные предложения, основанные на истории симпатий и антипатий пользователя. Многие интернет-магазины предоставляют рекомендуемые услуги, например, Amazon, CDNOW, BarnesAndNoble, IMDb и так далее.

Существует два распространенных подхода к созданию рекомендательных систем - коллаборативная фильтрация (CF) и рекомендации на основе содержимого (CB). Системы CF работают, собирая отзывы пользователей в форме оценок для элементов в данной области и используют сходства и различия между профилями нескольких пользователей, чтобы определить, как рекомендовать элемент. С другой стороны, методы, основанные на фильтрации содержания, предоставляют рекомендации путем сравнения представлений содержимого, содержащегося в элементе, с представлениями содержимого, которое интересует пользователя.

CB методы могут однозначно характеризовать каждого пользователя, но CF все же имеет некоторые ключевые преимущества перед ними. Во-первых, CF может работать в тех областях, где не так много контента, связанного с элементами, или где контент трудно анализировать компьютеру - идеи, мнения и так далее. Во-вторых, CF-система может предоставлять случайные рекомендации, т.е. - исправить элементы, которые имеют отношение к пользователю, но не имеют информации о профиле пользователя. По этим причинам CF-системы довольно успешно использовались для создания рекомендательных систем в различных областях. Однако они страдают от двух основных проблем:

1) *Разреженность*

   Проще говоря, большинство пользователей не оценивают большинство элементов, и, следовательно, матрица рейтингов пользовательских элементов обычно очень разреженная. Поэтому вероятность найти группу пользователей со значительно схожими рейтингами обычно невысока. Это часто случается, когда в системах очень высокое соотношение элементов и пользователей. Эта проблема также очень важна, когда система находится на начальной стадии использования.

2) *Проблема первого оценщика*
   Товар нельзя рекомендовать, если пользователь не оценил его ранее. Эта проблема относится к новым предметам, а также к непрозрачным предметам и особенно вредна для пользователей с эклектичными вкусами.

Гибридный метод оценки позволит преодолеть эти недостатки систем CF, используя информацию о содержании уже оцененных элементов. В подходе текущей рекомендательной системы будут использоваться прогнозы на основе содержимого для преобразования разреженной матрицы оценок пользователей в полную матрицу оценок; затем будет использоваться CF для предоставления рекомендаций. Текущая рекомендательная система будет предназначена для подбора музыки для пользователей.

Поиск подходящего набора данных

Первая задача, с которой придется столкнуться при создании подобного рода рекомендательной системы – это, конечно, поиск качественного набора данных, на основе которого впоследствии будет обучаться нейронная сеть для реализации CB метода.

При поиске набора данных для текущей рекомендательной системы были рассмотерны несколько вариантов:

1) Различные наборы данных на сайте https://www.kaggle.com/
2) Наборы данных на сайте https://data.world/datasets/music
3) The Million Song Dataset с http://millionsongdataset.com/
4) Набор данных, предоставляемый LastFM с http://ocelma.net/MusicRecommendationDataset/lastfm-1K.html

Наборы данных, которые были рассмотрены на сайтах из первого и второго варинатов имеют довольно малый объем данных – они пригодны для задач машинного обучения, но не глубокого обучения, который нужно использовать в нашем случае для обучения нейронной сети.

Третий вариант, The Million Song Dataset, имеет достаточное количество данных, но у него есть другая проблема: в нем предоставляется информация о том, какое количество пользователей любит и слушает определенные аудиозаписи и группы, но нет никакой связи между конкретными объектами и пользователями. Без конкретных связей не получится реализовать CB метод.

Последний вариант, набор данных с LastFM, имеет достаточное количество объектов, а также связи между конкретными объектами и пользователями. Он отлично подходит для текущей задачи. Однако, есть трудность, с которой пришлось столкнуться при работе с этим набором данных: у него довольно мало данных о характеристиках самих объектов – есть только названия треков и артистов, которые их исполняют. Для разрешения данной задачи есть возможность воспользоваться API, к которой LastFM предоставляет свободный доступ. При помощи запросов к API получится найти все необходимые характеристики для объектов – описание аудиозаписи, из которого можно получить жанр, альбом, год исполнения, а также тег. При наличии достаточного количества объектов, достаточного количества характеристик у этих объектов, характиристик пользователей и связей между пользователями и объектами будет возможно создание нейронной сети для метода CB. С помощью сети впоследствии будет упраздняться разреженость матрицы пользователей-объектов.

Описание подхода

Схема проектирования системы показан на рисунке 1.

После того, как был найден необходимый набор данных, его необходимо сохранить в БД. Полученные данные будут представлять из себя матрицу предпочтений пользователей, которая представляет собой матрицу пользователей и объектов, где на их пересечении будет находится рейтинг, присвоенный пользователем элементу. Рейтинг в нашем случае будет 0 или 1, то есть нравится трек пользователю, или нет.

Каждую строку этой матрицы можно назвать вектором оценок пользователей. Матрица оценок пользователей очень разреженная, поскольку большинство элементов не оцениваются большинством пользователей.

Нейронная сеть будет обучаться на основе данных о пользователях и данных о тех треках, которые им нравятся. В результате мы получим модель, которая сможет предсказать, понравится ли конкретный трек тому или иному пользователю, или нет.

С помощью этой модели создается вектор псевдо-оценок пользователей. Вектор псевдопользовательских оценок содержит фактические оценки пользователя и прогнозы на основе содержимого для элементов тех элементов, которые пользователи ещё не оценили. Все векторы псевдопользовательских оценок вместе образуют матрицу псевдорейтингов, которая представляет собой полную матрицу. И теперь, учитывая оценки активного пользователя и прогнозы для ещё не прослушанных треков, появляется возможность использовать CF метод: мы будем находить коэффициенты корреляции между векторами матрицы псевдо-оценок и определять ближайшие, то есть те, для которых значение коэффициента корреляции наибольшее. Из ближайших векторов пользователей мы сможем находить треки с положительной оценкой и которые пользователь, для которого делается прогноз, ещё не слышал. Такие треки можно будет порекомендовать пользователю.

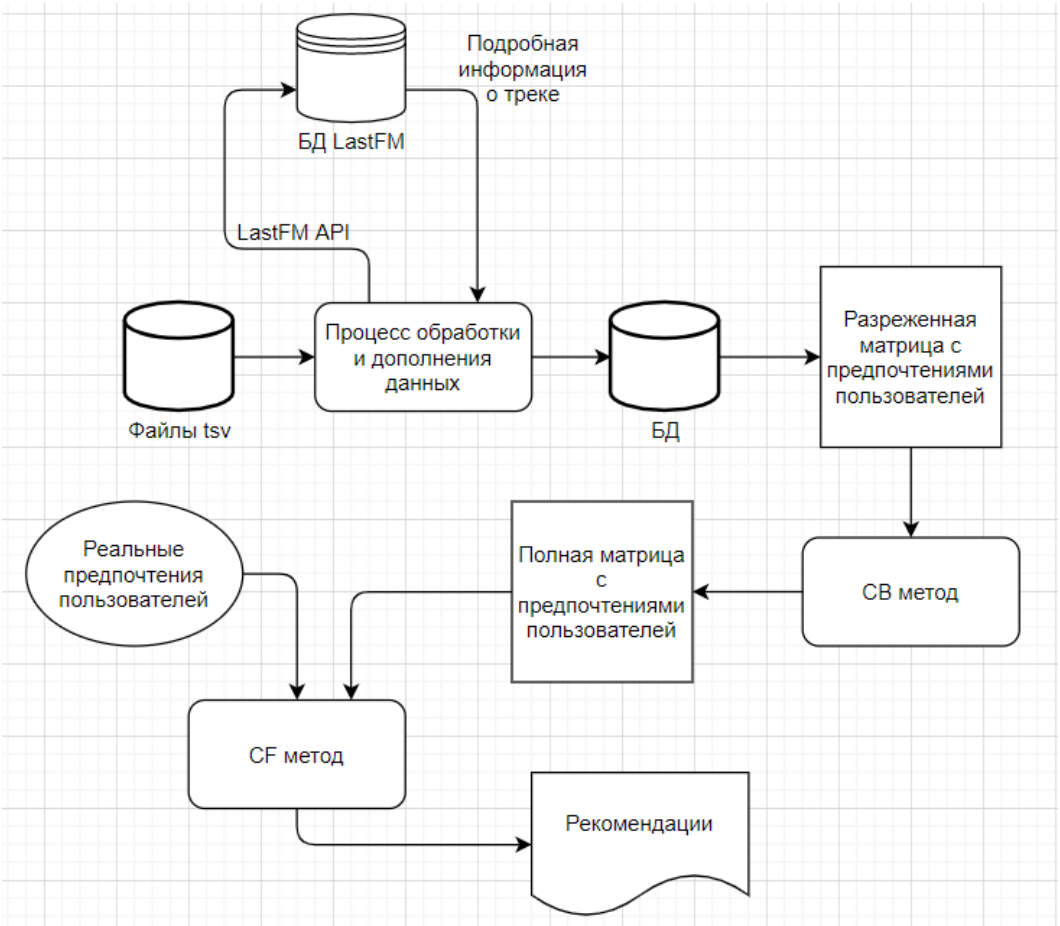

Рисунок 1. Общая схема проектирования системы

Обработка и дополнение набора данных

В наборе данных содержатся два файла с расширением tsv:

1) Первый - с информацией только о пользователя. Строка в файле содержит следующую информацию:
   <id пользователя - пол - возраст – страна – дата регистрации>
   Нам потребуются данные: id, пол, возраст и страна.

2) Второй – представляет из себя набор строк типа:
   < id пользователя – временная метка – id артиста – название артиста – id трека – название трека>
   Нам потребуются все данные.

Статистика данных файлов:

- Всего строк - 19,150,868
- Уникальных пользователей – 992
- Артистов с id представлено – 107,528
- Артистов без id представлено - 69,420

На этапе обработки и дополнения данных, учитывая большой объем, будет гораздо удобнее перенести информацию в БД. В качестве СУБД будем использовать MySQL. Структура БД показана на рисунке 2:

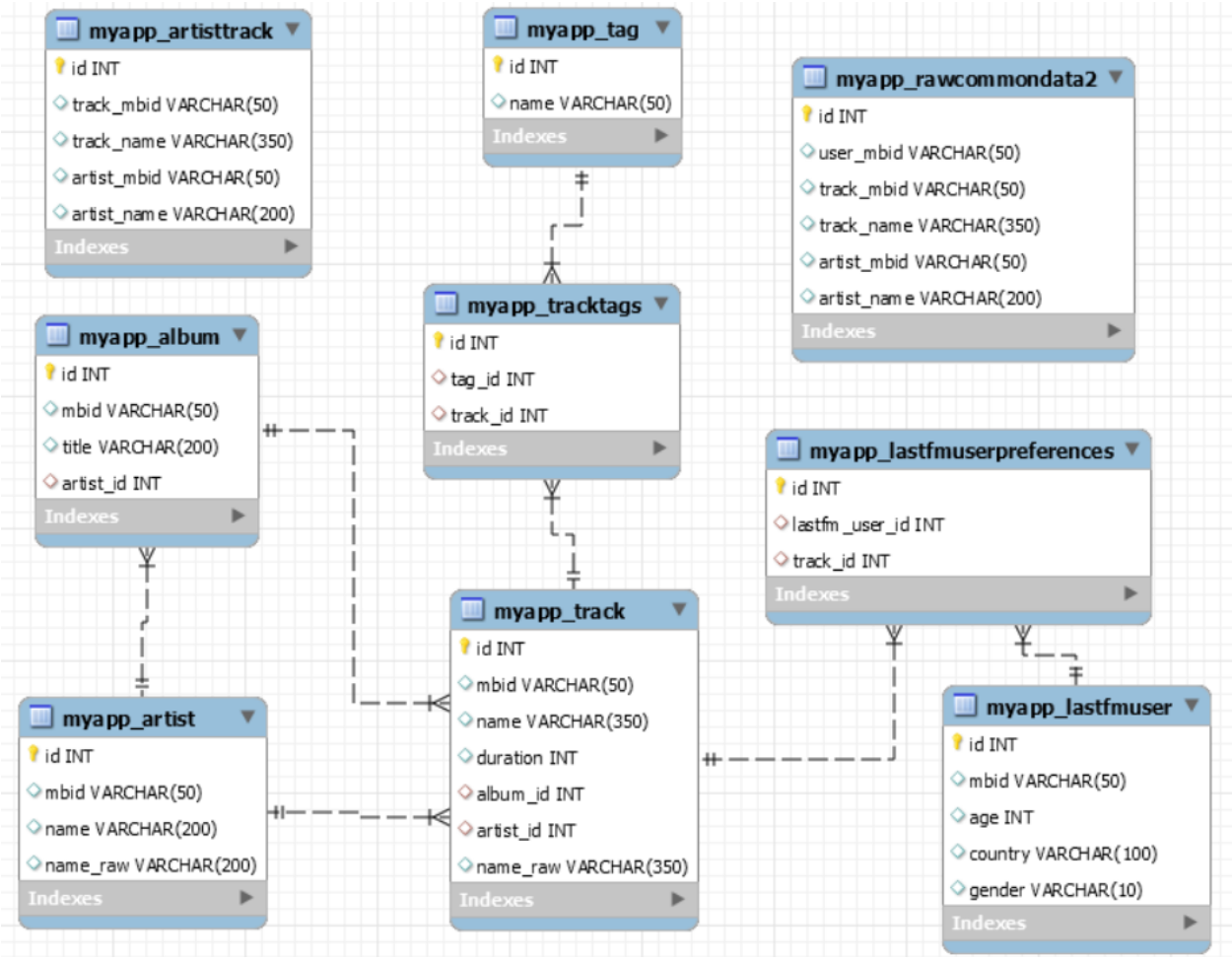

Рисунок 2. Структура БД

В первую очередь перенесем данные о пользователях в таблицу myapp_lastfmuser:

```
def get_user_data_from_row(row):
    data = dict()

    if is_empty_list(row):
        return data
    try:
        data['mbid'] = row[0]
        data['gender'] = row[1]
        data['age'] = get_value_of_type(row[2], int)
        data['country'] = row[3]

    except IndexError as e:
        print("Error: {}; \n Row: {}".format(e, row))

    return data
def save_user_if_not_exist(row):
    data = get_user_data_from_row(row)

    if not common_utils.is_empty_dict(data):
        user = LastFmUser(
            mbid=data['mbid'],
            gender=data['gender'],
            age=data['age'],
            country=data['country']
        )

        if not is_user_exists(user):
            user.save()


def main():
    user_info_file = open('userid-profile.tsv')
    read_file = csv.reader(user_info_file, delimiter='\t')

    line_number = 0

    for row in read_file:
        line_number += 1
        if line_number != 1:  # header with column names
            save_user_if_not_exist(row)

    user_info_file.close()


if __name__ == '__main__':
    main()
```

После этого перенесем данные из второго файла в таблицу myapp_rawcommondata2. При этом нужно учесть тот факт, что записи будут повторять комбинацию <id пользователя – название артиста – название трека>, потому что в файле присутствует временная метка. Одни и те же комбинации могут иметь разную временную метку, так как пользователь мог добавлять один и тот же трек в разные плейлисты в разное время или каким-либо другим способом взаимодействовать с треком.

```
def main():

    csv.field_size_limit(92233720)
    file_name = 'userid-timestamp-artid-artname-traid-traname.tsv'
    user_preferences_file = open(file_name, encoding='utf8')
    read_file = csv.reader(user_preferences_file, delimiter='\t')

    start_time = datetime.datetime.now()

    row_number = 0
    start_with_row = 0
    end_with_row = 20000000

    for row in read_file:
        row_number += 1

        if row_number >= start_with_row:
            try:
                data = get_user_preferences_data_from_row(row)

                process_raw_common_data(data)
            except Exception as e:
                if int(e.args[0]) == 1062:  # duplicate entry error number
                    pass
                else:
                    print("FAILED AT raw common data filling WITH ERROR: {}".format(e))
                    print("ROW_NUMBER: {}; ROW: {}".format(row_number, row))

        if row_number % 10000 == 0:
            print("Обработано записей: {}".format(row_number))
        if row_number >= end_with_row:
            break

    user_preferences_file.close()

    end_time = datetime.datetime.now()
    print("\n\n Времени всего прошло: {}".format(end_time - start_time))


if __name__ == '__main__':
    main()
def get_user_preferences_data_from_row(row):
    data = dict()

    if is_empty_list(row):
        return data

    try:
        data['user_mbid'] = get_value_of_type(row[0], str)
        data['artist_mbid'] = get_value_of_type(row[2], str)
        data['artist_name'] = get_value_of_type(row[3], str)
        data['track_mbid'] = get_value_of_type(row[4], str)
        data['track_name'] = get_value_of_type(row[5], str)

    except IndexError as e:
        print("Error: {}. \n Row: {}".format(e, row))

    return data
```

```
def process_raw_common_data(data):
    obj_raw = RawCommonData2(
        user_mbid=data['user_mbid'],
        artist_mbid=data['artist_mbid'],
        artist_name=data['artist_name'],
        track_mbid=data['track_mbid'],
        track_name=data['track_name']
    )

    obj_raw.save()
```

После выполнения этих двух скриптов, мы получим готовую таблицу с пользователями и ещё «сырую» таблицу с данными о предпочтениях пользователей и самими аудиозаписями.

Теперь нужно получить подробную информацию о каждом треке, чтобы впоследствии обучить на этих данных нейронную сеть. Чтобы получить подробную информацию о треке, будем использовать LastFM API (https://www.last.fm/api). При этом нужно учесть тот факт, что у API есть ограничение на количество запросов в определенный период времени. При превышении порога интенсивности запросов, сервис будет блокировать пользователя по его уникальному ключу, предаставленному при регистрации учётной записи.

На момент написания НИР было обработано 10 миллионов строк из файла с предпочтениями (из имеющихся примерно 19 миллионов). Из этих 10 миллионов было получено примерно 2.5 миллиона записей с уникальной комбинацией <id пользователя – название артиста – название трека>. Порог интенсивности запросов – 4 запроса в секунду. Зная это, можно рассчитать, что для того, чтобы для каждой записи из 2.5 миллионов сделать запрос на получение подробной информации о треке, нужно будет потратить около 7 дней. Это подсчёты без учёта того, что при увеличении количества объектов в БД, сами операции по добавлению объектов будут занимать все больше и больше времени.

Поэтому, было принято решение минимизировать число запросов к API: из имеющихся 2.5 миллионов записей понадобятся уникальные в разрезе <название артиста – название трека>. Таким образом, нужно также извлечь уникальные треки из таблицы myapp_rawcommondata2 и поместить их в таблицу myapp_artisttrack.

```python
def add_row_to_database(raw_object: RawCommonData2):
    artist_track_obj = ArtistTrack(
        artist_mbid=raw_object.artist_mbid,
        artist_name=raw_object.artist_name,
        track_mbid=raw_object.track_mbid,
        track_name=raw_object.track_name
    )

    artist_track_obj.save()


def main():
    row_number = 0
    start_with_row_number = 0

    for raw_object in RawCommonData2.objects.all():
        row_number += 1

        if start_with_row_number <= row_number:
            try:
                add_row_to_database(raw_object)

            except Exception as e:
                if int(e.args[0]) == 1062:  # duplicate entry error number
                    pass
                else:
                    print("FAILED WITH ERROR: {}".format(e))
                    print("ROW_NUMBER: {}; OBJ: {}".format(row_number, raw_object))

        if row_number % 10000 == 0 and row_number != 0:
            print("Обработано записей: {}; ".format(row_number))


if __name__ == '__main__':
    main()
```

После выполнения последнего скрипта в таблицу было занесено 1000265 записей. Их обработка займёт около 3 полных дней.

Теперь будем проходиться по каждому объекту таблицы myapp_artisttrack, делать API-запрос, чтобы получить подробную информацию о треке, и добавлять в таблицы myapp_artist, myapp_album, myapp_track, myapp_tag информацию об артисте, альбоме, треке и тегах трека соответственно.

```
def main():

    row_number = 0
    full_time_start = datetime.now()
    start_with_row_number = 0

    for raw_object in ArtistTrack.objects.all():
        row_number += 1
        start_iteration_time = datetime.now()

        if start_with_row_number <= row_number:
            add_row_to_database(raw_object)

        if row_number % 1000 == 0 and row_number != 0:
            print("Обработано записей: {};".format(row_number))
            print(raw_object)

        deadline_time = start_iteration_time + timedelta(milliseconds=250)
        end_iteration_time = datetime.now()

        if deadline_time > end_iteration_time:  # to be sure we don`t exceed lastfm api calling rate
            left_to_wait = float((deadline_time - end_iteration_time).microseconds) / 1000000
            sleep(left_to_wait)

    full_time_end = datetime.now()
    full_time = full_time_end - full_time_start
    print("Времени в общем потрачено: {}".format(full_time))


if __name__ == '__main__':
    main()
```

```
def process_raw_data(track_raw: Track, artist_raw: Artist):

    track_info = get_track_info(track_raw, artist_raw)

    if track_info['name'] is None or track_info['artist']['name'] is None:
        return

    try:
        artist = process_artist(track_info)
        album = process_album(track_info, artist)
        track = process_track(track_info, artist, album)
        process_tags(track_info, track)

    except Exception as e:
        print("Упало в {}; Error: {}".format(process_raw_data.__name__, e))
        print("Track_raw: {}; Artist_raw: {}".format(track_raw, artist_raw))


def add_row_to_database(raw_object: ArtistTrack):
    try:
        track_raw = Track(mbid=raw_object.track_mbid, name_raw=raw_object.track_name)
        artist_raw = Artist(mbid=raw_object.artist_mbid, name_raw=raw_object.artist_name)

        process_raw_data(track_raw, artist_raw)

    except Exception as e:
        print("Упало в {}; Error: {}".format(add_row_to_database.__name__, e))
        print("Объект: {}".format(raw_object))
```

```
def get_track_info(track_raw, artist_raw):

    response = call_track_info_api(track_raw.mbid, track_raw.name_raw, artist_raw.name_raw).json()

    track_info = {
        'mbid': get_value(response, 'track.mbid', None),
        'name': get_value(response, 'track.name', None),
        'name_raw': track_raw.name_raw,
        'duration': get_value(response, 'track.duration', None),
        'album': {
            'mbid': get_value(response, 'track.album.mbid', None),
            'title': get_value(response, 'track.album.title', None)
        },
        'artist': {
            'mbid': get_value(response, 'track.artist.mbid', None),
            'name': get_value(response, 'track.artist.name', None),
            'name_raw': artist_raw.name_raw
        },
        'tags': [get_value(tag, 'name', None) for tag in get_value(response, 'track.toptags.tag', [])]
    }

    return track_info
def call_track_info_api(track_mbid=None, track_name=None, artist_name=None, autocorrect=1):
    method = 'track.getInfo'
    payload = {
        'method': method,
        'mbid': track_mbid,
        'track': track_name,
        'artist': artist_name,
        'autocorrect': autocorrect
    }
    payload.update(COMMON_PAYLOAD)

    response = requests.get(LAST_FM_API_URL, headers=HEADERS, params=payload)
    return response
def process_artist(track_info):
    artist_raw = Artist(
        mbid=get_value_of_type(track_info['artist']['mbid'], str),
        name=get_value_of_type(track_info['artist']['name'], str)
    )

    artist_info = get_artist_if_exists(artist_raw)

    if not artist_info['artist']['exists']:
        artist_raw.save()
        artist = artist_raw
    else:
        artist = artist_info['artist']['artist_obj']

    return artist
def process_album(track_info, artist: Artist):
    album_raw = Album(
        mbid=get_value_of_type(track_info['album']['mbid'], str),
        title=get_value_of_type(track_info['album']['title'], str),
        artist_id=artist.id
    )

    album_info = get_album_if_exists(album_raw)

    if not album_info['album']['exists']:
        album_raw.save()
        album = album_raw
    else:
        album = album_info['album']['album_obj']

    return album
```

```
def process_track(track_info, artist: Artist, album: Album):
    track_raw = Track(
        mbid=get_value_of_type(track_info['mbid'], str),
        name=get_value_of_type(track_info['name'], str),
        duration=get_value_of_type(track_info['duration'], int),
        artist_id=artist.id,
        album_id=album.id
    )

    try:
        track_raw.save()
    except Exception as e:
        if int(e.args[0]) == 1062:  # duplicate entry error number
            pass
        else:
            print("Упало в функции {}".format(process_track.__name__))
            print("Artist: {}; Album: {}; Track: {}; Error: {}".format(artist, album, track_raw, e))

    track = track_raw

    return track
def process_tag(tag_name, track: Track):
    tag_raw = Tag(name=get_value_of_type(tag_name, str))

    tag_info = get_tag_if_exists(tag_raw)

    if not tag_info['tag']['exists']:
        tag_raw.save()
        tag = tag_raw
    else:
        tag = tag_info['tag']['tag_obj']

    try:
        TrackTags(
            track_id=track.id,
            tag_id=tag.id
        ).save()

    except Exception as e:
        if int(e.args[0]) == 1062:  # duplicate entry error number
            pass
        else:
            print("Упало в функции {}".format(process_tag.__name__))
            print("Tag: {}; Track: {}; Error: {}".format(tag, track, e))


def process_tags(track_info, track: Track):
    for tag_name in track_info['tags']:
        process_tag(tag_name, track)
```

На момент написания НИР мне ещё не удалось получить подробную информацию обо всех треках, так как этот процесс является довольно длительным. В дальнейшем будет описано то, каким образом следует спроектировать рекомендательную систему.

Предсказания на основе метода фильтрации на основе содержания

Для реализации CB метода понадобится создать нейронную сеть. На вход ей будут подаваться строки, в кажой из которых будет информация о пользователе и информация о треке. То есть на вход будут подаваться данные о том, треки какого типа будут нравится определенному типу пользователей. После обучения получим модель, через которую можно будет последовательно передавать информацию о каждом пользователе и каждом треке. Таким способом мы получим полную матрицу предпочтений пользователей.

Так как полный набор данных ещё не готов, попытаемся взять некоторую имеющуюся часть и на её основе реализовать предсказание с помощью CB метода при

помощи методов машинного обучения.

Возьмём 10000 строк имеющихся данных из таблицы myapp_rawcommondata2 и поместим данные о пользователях и треках в xlsx файл.

```
def main():

    workbook = Workbook()
    sheet = workbook.active

    fill_header(sheet)

    row_number = 0
    cell_number = 2

    for raw_obj in RawCommonData2.objects.all():
        row_number += 1

        try:
            if row_number < 10000:
                user = get_user(raw_obj)

                fill_row(sheet, raw_obj, user, cell_number)
                cell_number += 1
            elif row_number >= 10000:
                break

        except Exception as e:
            print("EXCEPTION: {}".format(e))

    save_excel(workbook)


if __name__ == '__main__':
    main()
```

```python
def fill_header(sheet: Workbook().active):
    sheet['A1'] = ATTRIBUTES['USER']['MBID']
    sheet['B1'] = ATTRIBUTES['USER']['AGE']
    sheet['C1'] = ATTRIBUTES['USER']['COUNTRY']
    sheet['D1'] = ATTRIBUTES['USER']['GENDER']
    sheet['E1'] = ATTRIBUTES['TRACK']['NAME']
    sheet['F1'] = ATTRIBUTES['ARTIST']['NAME']


def fill_row(sheet: Workbook().active, raw_obj: RawCommonData2, user: LastFmUser, cell_number: int):
    cell_num_srt = str(cell_number)

    sheet['A' + cell_num_srt] = user.mbid
    sheet['B' + cell_num_srt] = user.age
    sheet['C' + cell_num_srt] = user.country
    sheet['D' + cell_num_srt] = user.gender
    sheet['E' + cell_num_srt] = raw_obj.track_name
    sheet['F' + cell_num_srt] = raw_obj.artist_name
```

```python
def get_user(raw_obj: RawCommonData2):
    try:
        user = LastFmUser.objects.get(mbid=raw_obj.user_mbid)
    except ObjectDoesNotExist:
        user = LastFmUser.objects.first()

    return user


def save_excel(workbook: Workbook()):
    workbook.save(filename=FILENAME)
```

В качестве сравниваемых параметров будем использовать все характеристики пользователя (кроме его id) и все характеристики трека (кроме его названия). Все эти характеристики после конкатенации запишем в отдельный, результирующий столбец. В том случае, если значение одной из характеристик в строке будет отсутствовать, будем исключать его из результирующего значения.

Так как объём набора данных сравнительно небольшой, осуществлять СВ метод будем с помощью способа преобразования «объектов предметной области» в точки этого пространства, то есть с помощью векторизации. После векторизации, чтобы определить, подходит ли конкретный трек для конкретного пользователя (то есть определить, можно ли его порекомендовать) будем для каждого трека, который нравится конкретному пользователю, определять расстояние между ним и исследуемым треком.

Расстояние между векторами можно определять на основе нескольких метрик:

- косинусного расстояния
- манхэттэнского расстояния
- евклидового расстояния

В нашем случае будем использовать косинусное расстояние. Чем больше будет значение косинусного расстояния между двумя векторами, тем актуальнее будет рекомендация. Допустим условность: будем считать, что пользователю нравится трек в том случае, если косинусное расстояние хотя бы между одним любимым треком пользователя и исследуемым треком будет больше 0.7.

[16]
```
data.head()
```

	mbid	age	country	gender	track_name	artist_name
0	user_000001	NaN	Japan	m	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007	Deep Dish
1	user_000001	NaN	Japan	m	Composition 0919 (Live_2009_4_15)	坂本龍一
2	user_000001	NaN	Japan	m	Mc2 (Live_2009_4_15)	坂本龍一
3	user_000001	NaN	Japan	m	Hibari (Live_2009_4_15)	坂本龍一
4	user_000001	NaN	Japan	m	Mc1 (Live_2009_4_15)	坂本龍一

[53]
```
data.shape
```

```
(10003, 8)
```

Так как основной набор данных ещё не готов, а в существующей части данных не очень много характеристик у трека, искусственным образом добавим жанры для треков: каждому исполнителю присвоим свой собственный жанр

[17]
```
genres = ['rock', 'rap', 'pop', 'jazz']
artist_genres = dict()

for artist in list(data['artist_name'].unique()):
    if artist not in artist_genres:
        artist_genres[artist] = random.choice(list(genres))
```

[18]
```
def genre_for_track(row):
    return artist_genres[row['artist_name']]
```

[19]
```
data['genre'] = data.apply(genre_for_track, axis=1)

data.head()
```

	mbid	age	country	gender	track_name	artist_name	genre
0	user_000001	NaN	Japan	m	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007	Deep Dish	jazz
1	user_000001	NaN	Japan	m	Composition 0919 (Live_2009_4_15)	坂本龍一	rock
2	user_000001	NaN	Japan	m	Mc2 (Live_2009_4_15)	坂本龍一	rock
3	user_000001	NaN	Japan	m	Hibari (Live_2009_4_15)	坂本龍一	rock
4	user_000001	NaN	Japan	m	Mc1 (Live_2009_4_15)	坂本龍一	rock

[20]
```
features = ['age', 'country', 'gender', 'artist_name', 'genre']

def get_value_of_type(value, type, feature=''):
    if value in [None, 'NaN', 'nan', np.nan] or str(value) == str(np.nan):
        return ''
    else:
      if type == int:
          return feature + str(value) + " "
      elif type == str:
          return feature + value + " "

def combine_features(row):
    age = get_value_of_type(row['age'], int, 'age')
    country = get_value_of_type(row['country'], str, 'country')
    gender = get_value_of_type(row['gender'], str, 'gender')
    artist_name = get_value_of_type(row['artist_name'], str)
    genre = get_value_of_type(row['genre'], str, 'genre')

    return age + country + gender + artist_name + genre
```

[21]
```
data['combined_feature'] = data.apply(combine_features, axis=1)
```

[24]
```
data.head()
```

	mbid	age	country	gender	track_name	artist_name	genre	combined_feature
0	user_000001	NaN	Japan	m	Fuck Me Im Famous (Pacha Ibiza)-09-28-2007	Deep Dish	jazz	countryJapan genderm Deep Dish genrejazz
1	user_000001	NaN	Japan	m	Composition 0919 (Live_2009_4_15)	坂本龍一	rock	countryJapan genderm 坂本龍一 genrerock
2	user_000001	NaN	Japan	m	Mc2 (Live_2009_4_15)	坂本龍一	rock	countryJapan genderm 坂本龍一 genrerock
3	user_000001	NaN	Japan	m	Hibari (Live_2009_4_15)	坂本龍一	rock	countryJapan genderm 坂本龍一 genrerock
4	user_000001	NaN	Japan	m	Mc1 (Live_2009_4_15)	坂本龍一	rock	countryJapan genderm 坂本龍一 genrerock

Найдём косинусное расстояние между каждым объектом. Чем больше оно будет, тем ближе будут находиться вектора объектов, то есть, тем актуальнее будет данная рекомендация.

[25]
```
cv = CountVectorizer()
count_matrix = cv.fit_transform(data['combined_feature'])
cosine_sim = cosine_similarity(count_matrix)
```

[67]
```
def get_index_from_name_(name):
    return data[data.track_name == name].index.values[0]
```

Возьмём конкретного пользователя и несколько треков. Определим, понравятся они ему или нет. Допустим условность, что пользователю нравится трек в том случае, если косинусное расстояние хотя бы между одним любимым треком пользователя и исследуемым треком будет больше 0.7

[79]
```
def can_recommend_track_for_user(user_mbid, track_name):
    user_fav_tracks = data[data.mbid == user_mbid]['track_name'].values

    track_index = get_index_from_name(track_name)

    for fav_track in user_fav_tracks:
        fav_track_index = get_index_from_name(fav_track)
        if cosine_sim[track_index][fav_track_index] > 0.7:
            return True

    return False
```

[82]
```
user = 'user_000017'

track_name1 = 'Oblivion'
track_name2 = 'Mc2 (Live_2009_4_15)'
track_name3 = 'What It Takes'

for track in [track_name1, track_name2, track_name3]:
    print(can_recommend_track_for_user(user, track))
```

```
True
False
True
```

При обучении нейронной сети придется также применять метод векторизации.

Предсказания на основе метода коллаборативной фильтрации

После создания нейронной сети мы получим полную матрицу с псевдорейтингами пользователей. Каждая строка матрицы представляет собой пользовательский вектор. Корреляция двух векторов будет являться показателем их сходства. При создании рекомендаций для конкретного пользователя будут использоваться треки наиболее схожих с ним пользователей. Те треки, которые имеют положительную оценку и которые целевой пользователь ещё не слышал, будут использоваться как рекомендуемые.

Алгоритм можно свести к следующим этапам:

1. Оцениваем всех пользователей с точки зрения сходства с активным пользователем.
- Сходство между пользователями измеряется как корреляция между их векторами оценок.
2. Выбираем n пользователей, которые имеют наибольшее сходство с активным пользователем.

3. Вычисляем прогноз на основе взвешенной комбинации рейтингов выбранных соседей.

На шаге 1 сходство между двумя пользователями вычисляется с использованием коэффициента корреляции Мэтьюса, так как значения рейтингов в нашем случае будут бинарными (0 – не нравится, 1 – нравится):

$$\mathrm{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

		Фактический класс	
		П	**N**
Прогнозируемый класс	**П**	**TP**	FP
	N	FN	**TN**

Где:

- MCC - коэффициент корреляции Мэтьюса;
- P - Положительный;
- N - отрицательный;
- TP - истинно положительный;
- FP - ложноположительный результат;
- TN - истинно отрицательный;
- FN - ложноотрицательный;

Значением фактического класса будет являться значение в векторе целевого пользователя, а значением прогнозируемого класса будет являться значение в векторе пользователя, которого взяли на текущей итерации.

Также, можно было бы вычислять схожесть пользователей как коэффициент корреляции Пирсона:

$$P_{a,u} = \frac{\sum_{i=1}^{m} (r_{a,i} - \overline{r}_a) \times (r_{u,i} - \overline{r}_u)}{\sqrt{\sum_{i=1}^{m} (r_{a,i} - \overline{r}_a)^2 \times \sum_{i=1}^{m} (r_{u,i} - \overline{r}_u)^2}}$$

Где:

- $!r_a$ – средняя оценка целевого пользователя a;
- $!r_u$ – средняя оценка пользователя u;
- $R_{a,i}$ – рейтинг, который дал целевой пользователь a объекту i;
- $r_{u,i}$ – рейтинг, который дал пользователь u объекту i;
- m – количество треков (длина вектора);
- $P_{a,u}$ – коэффициент корреляции между пользователями a и u;

```
def matthews_corr(X, Y):
    A = B = C = D = 0

    if len(X) != len(Y):
        raise Exception("Длины векторов различаются")

    for i in range(len(X)):
        if X[i] == 1 and Y[i] == 1:
            A += 1
        elif X[i] == 0 and Y[i] == 1:
            B += 1
        elif X[i] == 1 and Y[i] == 0:
            C += 1
        elif X[i] == 0 and Y[i] == 0:
            D += 1

    corr = (A*D - B*C) / sqrt((A+B)*(C+D)*(A+C)*(B+D))
    return corr


def pirson_corr(X, Y):
    Xavg = sum(X) / len(X)
    Yavg = sum(Y) / len(Y)

    numerator = sum([ (X[i] - Xavg) * (Y[i] - Yavg) for i in range(len(X)) ])
    denominator1 = sum([ (X[i] - Xavg)**2 for i in range(len(X)) ])
    denominator2 = sum([ (Y[i] - Yavg)**2 for i in range(len(X)) ])

    corr = numerator / sqrt(denominator1 * denominator2)
    return corr
```

```
X = [1, 1, 1, 1, 1, 1, 1, 1, 1, 0]
Y = [0, 0, 0, 0, 1, 0, 0, 0, 0, 1]

print(matthews_corr(X, Y))
print(pirson_corr(X, Y))
```

```
-0.6666666666666666
-0.6666666666666666
```

Как видим, значения коэффициентов корреляции, рассчитанные методами Мэтьюса и Пирсона, одинаковы. Поэтому, для экономии ресурсов, рациональнее будет использовать метод Мэтьюса.

На шаге 3 прогнозы вычисляются как средневзвешенное значение отклонений от среднего значения соседа:

$$p_{a,i} = \overline{r}_a + \frac{\sum_{u=1}^{n} (r_{u,i} - \overline{r}_u) \times P_{a,u}}{\sum_{u=1}^{n} P_{a,u}}$$

Где

- $!r_a$ – средняя оценка целевого пользователя a;
- $!r_u$ – средняя оценка пользователя u;
- $r_{u,i}$ – рейтинг, который дал пользователь u объекту i;
- $p_{a,i}$ – предсказание для целевого пользователя a для объекта i;
- $P_{a,u}$ – коэффициент корреляции между пользователями a и u;
- n – количество ближайших соседей;

Для искомого пользователя характерно наличие сильно коррелированных соседей, у которых есть небольшое количество реальных (а не псевдо) оценок объектов в векторе с одинаковым рейтингом. Рекомендации, которые будут производиться на основе этих

соседей не могут считаться качественными. Чтобы уменьшить корреляцию, основанную на нескольких совместно оцениваемых элементах, мы будем умножать корреляцию на весовой коэффициент значимости (Significance Weighting factor - *sg*).

Возьмём некоторое константное значение *N*. Если у двух пользователей менее *N* элементов с одинаковым рейтингом, мы умножаем их корреляцию на коэффициент $sg_{a,u} = n / N$, где *n* - количество элементов с одинаковым рейтингом. Если количество перекрывающихся элементов больше *N*, то мы оставляем корреляцию неизменной, т.е. $sg_{a,u} = 1$.

Таким образом, мы сможем получить ближайших соседей, и сделать конкретные рекомендации для искомого пользователя.

Вывод

В рамках данной НИР были приведены имеющиеся результаты по проектированию и кодированию рекомендательной системы по подбору музыки для пользователей и те способы, при помощи которых результаты были достигнуты. Также приведен план дальнейшей работы по проектированию и реализации методов фильтрации на основе содержания (CB) и коллаборативной фильтрации (CF) с тем, чтобы на их основе получить конечную программу, которую можно будет интегрировать в имеющиеся АИС различных музыкальных сервисов.

Список использованных источников

1) Курс лекций «Методы машинного обучения» [Электронный ресурс] https://github.com/ugapanyuk/ml_course_2020/wiki/COURSE_MMO
2) Melville P., Mooney R., Nagarajan R. Content-Boosted Collaborative Filtering for Improved Recommendations (англ.) // University of Texas, USA : Материалы конф. / AAAI-02, Austin, TX, USA, 2002. — 2002. — P. 187-192.
3) Документация библиотеки pandas: https://pandas.pydata.org/pandas-docs/stable/index.html