МИНОБРНАУКИ РОССИИ
Федеральное государственное бюджетное образовательное учреждение
высшего образования
**«САРАТОВСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИМЕНИ Н. Г. ЧЕРНЫШЕВСКОГО»**
Кафедра математической кибернетики и компьютерных наук

НЕЙРОННЫЕ СЕТИ

БАКАЛАВРСКАЯ РАБОТА

студента 4 курса 451 группы
направления 09.03.04 — Программная инженерия
факультета КНиИТ
Озерова Данилы Николаевича

Научный руководитель
доцент, к. ф.-м. н. _____ С. В. Миронов

Заведующий кафедрой
к. ф.-м. н. _____ С. В. Миронов

Саратов 2023

СОДЕРЖАНИЕ

ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

ВВЕДЕНИЕ

В современном мире практически на все, что мы покупаем или потребляем в той или иной форме влияют рекомендации, будь то от друзей, рекламы или, как стало с недавнего времени, от источников предложения товаров. Например, платформа Кинопоиск или Озон предлагают список фильмов и товаров, которые по мнению сервиса могут приглянутся пользователю. Большинство ранних рекомендательных моделей пыталась разделить пользователей на стереотипные группы, чтобы рекомендовать данной группе один и тот же товар. Примером такой работы может служить система библиотекаря Гранди []. Однако быстро такой подход получил критику в научном мире, поскольку «люди очень слабы в изучении и описании собственных когнитивных процессов» []. По исследованиям люди часто подчеркивают свои отличительные качества, что затрудняет работу по созданию стереотипов.

Таким образом можно сделать предположение, что предоставление качественных рекомендаций лежит в основе успешности работы современного бизнеса.

Цель итоговой аттестационной работы — изучение и построение различных моделей рекомендательных систем в области сервиса по просмотру анимации в стиле аниме, направленных на предсказание наиболее вероятных аниме, подходящих пользователю. В данной работе под моделью понимается ряд шагов, выполнение которых приведет к построению наиболее эффективной рекомендательной системы в рассматриваемой предметной области.

Поставленная цель определила следующие задачи:

- Выбрать данные для обучения и тестирования модели, провести их анализ.
- Построить различные модели рекомендательной системы.
- Протестировать качество прогнозирования.

1 Теоретическая часть

Основную часть данной работы составляет построение и анализ алгоритмов построения рекомендательных систем. В качестве предметной области была выбрана область стримингового сервиса аниме-фильмов.

Объектом исследования стал процесс прогнозирования интересов пользователя стримингового аниме сервиса. Предметом исследования работы является разработка модели для реализации рекомендательной системы, прогнозирующей интересы пользователя на основе просмотренных аниме.

1.1 Коллаборативная фильтрация

Современные подходы к построению рекомендательных систем используют либо коллаборативную (совместную) фильтрацию, либо фильтрацию на основе контента. Данные методы строят модель на основе прошлых данных о пользователе в системе, а также похожих решений, принятых другими пользователями.

В данной работе ставится цель создать рекомендательную систему для информирования пользователя об аниме, которые ему могут быть интересны. Предметом рекомендации является аниме, источник рекомендации - аудитория. Тип рекомендательной системы - коллаборативная на основе пользователей (user-based). Поскольку появление новых пользователей и аниме вынуждают выполнять перерасчет системы, то user-based тип системы более подходит данной предметной области, поскольку аниме, к примеру, добавляются в систему реже, чем пользователи.

К проблемам совместной фильтрации исследование [] относит проблему холодного старта (недостатка данных при запуске системы), однако в данном случае используется начальный большой датасет с популярного интернет-портала, поэтому данная проблема несущественна.

Совместная фильтрация успешно используется в смежных предметных областях в проектах ???, что дает возможность высказать предположение об эффективности при решении поставленной задачи.

Работа начинается с построения и анализа примитивной модели, когда для всех пар возвращаемое значение равно среднему рейтингу. Затем проводится улучшение модели на такую, которая выводит среднюю оценку между всеми пользователями, которые его оценили. В следующей модели производится сле-

дующая модификация: оценкам пользователей добавляется вес, то есть модель отдает предпочтение тем пользователям, чьи оценки похожи на рассматриваемого пользователя. Определим описанный вес так:

$$\bar{r}_{ua} = \frac{1}{\sum_{u',u' \neq u} s_{uu'}} \sum_{u',u' \neq u} s_{uu'} r_{u'a}$$

, где

$\bar{r}_{ua}$ — это предполагаемая оценка товара a пользователем u, $s_{uu'}$ — коэффициент похожести u на u', $r_{u'a}$ означает оценку пользователем u' товара a.

В данной модели учитывался рейтинг каждого пользователя для прогноза окончательного рейтинга. Выдвинем гипотезу, что группировка пользователей с помощью кластеризации улучшит результат. Самая популярная реализация алгоритма основана на принципе k ближайших соседей. Она позволит выбрать k пользователей с наиболее похожими интересами относительно рассматриваемого человека. Далее данный метод предполагает выбор меры в отношении рассматриваемого пользователя и выбор k наибольших. После подсчета меры для каждого из пользователей производится умножение меры на его оценки. Затем для каждого аниме следует подсчитать сумму калиброванных оценок k наиболее близких пользователей, а затем эту сумму разделить на сумму мер k выбранных пользователей.

$$\bar{r}_{ua} = \frac{1}{\sum_{u' \in N(A)} s_{uu'}} \sum_{u' \in N(A)} s_{ai} r_{ij}$$

, где

$\bar{r}_{ua}$ — это предполагаемая оценка товара a пользователем u, $s_{uu'}$ — коэффициент похожести u на u', $r_{u'a}$ означает оценку пользователем u' товара a, а $N(A)$ определяет множество схожих пользователей определенных алгоритмом (в данной работе - KNN).

В качестве меры s в данной работе, основываясь на результатах статьи [A$_{R}eview_{o}n_{S}imilarity_{M}easurement_{M}ethods_{i}n_{T}rus.pdf$], 1, J.[$Moghaddam$], [$JinChen$] J..

1.2 Коллаборативная фильтрация и нейронные сети

Нейронная коллаборативная фильтрация (НКФ) — это использование нейронной сети определенной архитектуры для моделирования на основе имеющейся информации собственных векторов объектов и пользователей, а также изучения функции, описывающей их взаимодействие, на основе которой и составляются рекомендации.

Первым шагом, в данную модель подаются два различных вектора, характеризующие соответственно аниме и пользователя. Описанные вектора поступаю парой пользователь-аниме на вход нейронной сети. Далее, они по отдельности проходят через независимые нейронные уровни (embedding) для преобразования и уплотнения. На выходе после этого получаем собственные вектора аниме и пользователя. Соответственно, на этом этапе нейронной сети происходит обучение для получения осмысленного векторного представления данных, то есть перевод входных данных в пространство признаков заданной длины. Полученные собственные вектора аниме и пользователя объединяются и поступают в следующий полносвязный нейронный слой. Далее, архитектура сети представляет собой полносвязные уровни, количество которых может варьироваться, а количество нейронов уменьшается с увеличением уровня. Последний уровень, или выход сети, представляет собой однонейронный полносвязный уровень с логистической функцией активации. Соответственно, в качестве ответа сеть выдает вероятность взаимодействия пользователя с аниме в диапазоне от 0 до 1. Цель обучения — минимизации функции ошибки между имеющимися значениями матрицы Y и предсказанными ^{y}. При обучении используются не только положительные примеры, то есть просмотренные аниме, но и отрицательные — аниме, с которым конкретный пользователь никак не взаимодействовал.

Далее рассматривается различные параметры, которые используются при обучении нейронной сети.

1.2.1 Функция активации

Сигмоида — нелинейная функция, хорошо подходящая для задач классификации, имеющая фиксированный диапазон значений, и стремящаяся перевести значения к концам этого диапазона. Формула сигмоидальной функции:

$$F(x) = \frac{1}{1 + e^{-x}}$$

.

1.2.2 Оптимизаторы

Оптимизатор Adam. Его особенность в том, что скорость обучения настраивается для каждого веса связи отдельно с помощью деления коэффициента на скользящие средние значения недавних градиентов и их вторых моментов для этого веса(https://dspace.spbu.ru/bitstream/11701/32252/1/Dissertacia$_m oskovs$)., , , , /.

Оптимизатор RMSprop напоминает градиентный спуск с импульсом, однако уменьшает колебания в вертикальном направлении. Таким образом, алгоритм может учиться быстрее и делать большие шаги в горизонтальном направлении. Главное отличие между RMSprop и градиентным спуском - это метод вычисления градиентов.

Стохастический градиентный спуск (SGD) является методом оптимизации, который имеет особенность, отличающую его от стандартного градиентного спуска - на каждом шаге градиент оптимизирующей функции вычисляется не путем суммирования градиентов от каждого элемента выборки, а именно как градиент от одного случайно выбранного элемента.

1.2.3 Функция потерь

Бинарная кросс-энтропия используется для оценки различий между вероятностными распределениями. Чем больше значение кросс-энтропии, тем больше расхождение между распределениями, а меньшее значение указывает на более схожие распределения.

1.2.4 Метрики

Среднеквадратичная ошибка (Mean Squared Error) – Среднее арифметическое квадратов разностей между предсказанными и реальными значениями модели. Формула:

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

.

Средняя абсолютная ошибка (Mean Absolute Error) отличается только по формуле:

$$MAE = \frac{1}{n}\sum_{1}^{n}|y_i - \bar{y}_i|$$

.

Так как оптимизаторы Adam и RMSprop лучше работают на больших данных, ожидается, что модель с ними будет работать лучше, чем с SGD.

2 Практическая часть

2.1 Формальная постановка задачи

Формальная постановка задачи для рекомендаций выглядит следующим образом. Рассмотрим множество пользователей U и D — множество объектов. Необходимо найти функцию $r, r : U \times D \rightarrow R$, которая формирует вектор рекомендаций R таким образом, что для любого пользователя значение r_i между ним и объектом i является расстоянием (вероятностью взаимодействия) между пользователем u_i и объектом d_i.

2.2 Характеристика ЭВМ

Работа производится на ЭВМ, предоставляемая средой Google Colab. В качестве программного средства используется Jupyter Notebook.

2.3 Описание данных

Датасет под названием Anime Recommendation Database 2020 был получен из социальной сети Kaggle и состоит из данных собранных с популярного зарубежного стримингового аниме сервиса myanimelist.net.

2.3.1 Описание animelist.csv.

Данный dataset содержит оценки, которые ставили пользователи для аниме.

Описание столбцов датасета:

- $\mathrm{user\_id}$ — id пользователя
- $\mathrm{anime\_id}$ — id аниме
- rating — оценка пользователя для аниме
- $\mathrm{watching\_status}$ — статус аниме у пользователя (1 - смотрит в данный момент, 2 - просмотрено, 3 - пока не смотрит, 4 - брошено, 6 - в планах)
- $\mathrm{watching\_episodes}$ — количество просмотренных эпизодов

Таблица содержит 109 миллионов строк. Записи распределены по пользователям, то есть сначала идет информации о просмотренных аниме $\mathrm{user\_id} = 0$ пользователем и так далее. Выглядит это следующим образом:

2.3.2 Описание anime.csv.

В данном датасете 35 столбцов, большинство из них несут различную информацию про каждое аниме, поэтому опишем наиболее значительные данные

для дальнейшей работы:

  - MAL$_I D - id(int) Name - (string)$

— score - средняя оценка аниме от всех пользователей в базе данных MyAnimelist.(float)
— genres - разделенный запятыми список жанров для этого аниме. (string)
— type - фильм, сериал, дополнительная серия и тд.(string)
— episodes - количество частей (int)
— aired - дата трансляции.(date)
— premiered - дата премьеры (date)
— rating - возрастной рейтинг (string)
— Popularity - позиция по количеству пользователей, которые добавили аниме в свой список.(int)
— score-1 - score-10 - количество зрителей, поставивших от 1 до 10 соответственно. (float)

MAL$_I Danime.csvanime_i danimelist.csv, ..anime_i d, id.$

2.3.3 Описание anime$_w ith_s ynopsis.csv.$

Данный датасет представляет собой краткий обзор для каждого аниме.

  - MAL$_I D - id(int); Name - (string);$

— score - средняя оценка аниме от всех пользователей в базе данных MyAnimelist (float);
— genres - разделенный запятыми список жанров для этого аниме (string);
— sypnopsis - описание аниме.

MAL$_I D$.

2.4 Разведочный анализ

Первоначально был изменен размер данных, чтобы учитывать более релевантных пользователей:

```
n_ratings = rating_df['user_id'].value_counts()
rating_df = rating_df[rating_df['user_id'].isin(n_ratings[n_ratings >= 1000].index)]
```

В результате работа будет производится с датасетом размером 11366011 строк.

Таблица распредения данных выглядит следующим образом:

Столбец rating вызывает наибольший интерес для данного исследования. Стандартное отклонение и среднее значение примерно равно, это можно трактовать как то, что данные в этом столбцы равномерно распределены вокруг

среднего значения, т. е. нет большого перевеса в одно из значений. Это подтверждают и значения квантиля этого столбца.

Пропуски и дубликаты в таблие отсутствуют.

Распределение столбца rating было изучено на данном графике:

Отметка rating = 0 выставляется, если пользователь не поставил оценку, этот вариант преобладает над другими. Кроме того, можно заметить, что пользователя чаще ставят оценку “выше среднего”. Можно предположить, что среднее значение более низкое, так как нулевой рейтинг портит эту статистику.

Был рассмотрен параметр watch$_episodes$, :

Можно заметить несколько выбросов на графике. Данный вывод сделан из данных о том, что аниме с самым большим количеством серий насчитывает около 8500 эпизодов.

Было рассмотрено количество ненулевых оценок, которые ставят пользователи. Возьмем десять наибольших (слева - id пользователя, справа - количество ненулевых оценок):

Разрыв в значениях между первым и последним сильно отличаются, то есть некоторые пользователи будут иметь намного больше веса в системе, чем другие.

Были рассмотрены аналогичные сведения об аниме:

Количество ненулевых оценок распределено более равномерно, возможно, в системе некоторые аниме не будут чаще предлагаться, чем другие, только потому что их больше людей оценило.

Это может быть связано с тем, что количество аниме сильно больше, чем пользователей:

2.5 Предобработка данных

Данные изначально не требуют больших изменений (отсутствуют пропуски, дубликаты, все значения - int, rating изменяется в определенном диапазоне). Однако в ходе разведочного анализа были обнаружены выбросы - их необходимо удалить:

```
rating_df = rating_df[rating_df['watched_episodes'] < 9000]
```

Также существуют проблема, важная для рекомендательной системы. В таблице существуют пользователи, которые оценили абсолютно все аниме на 0, и аниме, которые все пользователи оценили на 0. Такие данные не несут никакой

полезной информации, и, более того, могут помешать дальнейшему обучению, поэтому их следует удалить:

```
grouped = rating_df.groupby(['user_id']).agg({'rating': 'mean'})
grouped = grouped.loc[grouped['rating'] == 0]
user_ids = grouped['user_id'].unique()
rating_df = rating_df.loc[~rating_df['user_id'].isin(user_ids)]
grouped = rating_df.groupby(['anime_id']).agg({'rating': 'mean'})
grouped = grouped.loc[grouped['rating'] == 0]
user_ids = grouped['anime_id'].unique()
rating_df = rating_df.loc[~rating_df['anime_id'].isin(user_ids)]
```

2.6 Разработка и анализ моделей

Как описано в разведочном анализе, целевая переменная - рейтинг, который принимает целочисленные значения от 1 до 10. В данной работе проблема формулируется как пример обучения с учителем, где нужно предсказать рейтинг, учитывая пользователя и аниме. Хотя рейтинг принимает дискретные значение в диапазоне [0;10], имеет смысл сформулировать проблему, как задачу регрессии. Так следует поступить, поскольку в случае если модель классификации будет предсказывать оценку аниме с истинным рейтингом 10, то и оценка 9, и оценка 1 будут интерпретированы одинаково - как неверный класс. В отличие от этого регрессионная модель наказывает за второй ответ больше чем за первый и это поведение подходит для данной задачи больше.

Первоначально для анализа необходимо выбрать метрику. В данной работе предлагается использовать RMSE (среднеквадратичную ошибку) для сравнения качества предсказаний моделей, как указано в источнике [https://smltar.com/mlregre

Как было сказано ранее, первая построенная модель - базовая. Подсчитанные результаты будут использоваться как отправная точка для того, чтобы понять смогли ли мы улучшить результаты.

Разработанные в данном исследовании модели будет принимать id пользователя (user$_id$)$id(anime_id), -$.

Определим функцию для базовой модели:

```
def base_model(user_id, anime_id):
        return 5.0
```

Также создадим функцию для подсчета средней квадратической ошибки на данных, где в качестве параметра выступает модель:

```
from sklearn.metrics import mean_squared_error
def score(anime_model):
  id_pairs = zip(X_test['user_id'], X_test['anime_id'])
  y_pred = np.array([anime_model(user, anime) for (user, anime) in id_pairs])
  y_true = np.array(X_test['rating'])
  return mean_squared_error(y_true, y_pred, squared=False)
```

Здесь на строке 3 создаются пары из user$_i$d model$_i$d, .4., 6RMSE mean$_s$quared$_e$rro

Запустив разработанную функцию для базовой модели, получаем показатель меры RMSE равный 4,17.

2.7 Модель на основе среднего значения

Начинаем применять подход коллаборативной фильтрации основанной на пользователях. Для этого первоначально подготовим матрицу, где строка будет представлять id пользователя x, столбец id аниме y, а на пересечении стоять оценка, данная пользователем x аниме y. Для построения такой таблицы воспользуемся функцией pivot$_t$ablepandas.

```
      r_matrix = X_train.pivot_table(values='rating', index = 'user_id',
      ↪  columns='anime_id')
```

```
r_matrix = X_train.pivot_table(values='rating', index = 'user_id',
↪  columns='anime_id')
```

В этой реализации опробуем модель, которая будет выводить среднюю оценку между всеми пользователями, которые его оценят. Все пользователи будут считаться равными. Иными словами, рейтингу каждого пользователя присваивается равный вес.

Учтем случай, что некоторые аниме попали только в тестовый набор: в таком случае как и в базовой модели установим рейтинг 5.

```
def mean_model(user_id, anime_id):
  if anime_id in r_matrix:
    mean_raiting = r_matrix[anime_id].mean()
  else:
    mean_raiting = 5.0
  return mean_raiting

```

Метрика RSME для этой модели составляет 4.03.

Видим, что среднеквадратическое отклонение уменьшилось, следовательно качество предсказаний улучшилось.

2.8 Модель на основе средневзвешенной оценки

В данной модели будем отдавать больше предпочтения тем пользователям, чьи оценки похожи на рассматриваемого пользователя больше.

Поэтому дополним предыдущую модель весовым коэффициентом:

$$\bar{r}_{ua} = \frac{1}{\sum_{u',u' \neq u} s_{uu'}} \sum_{u',u' \neq u} s_{uu'} r_{u'a}$$

, где

В теоретической части было представлено обоснование использование косинусной метрики в качестве s. Для работы с косинусной метрикой подключим функцию cosine$_similaritysklearn.$

```
from sklearn.metrics.pairwise import cosine_similarity
```

Функция cosine$_simNaN.0$:

```
r_matrix_notnull = r_matrix.copy().fillna(0)
```

С помощью функции построим новую матрицу косинусного сходства между пользователями:

```
cosine_sim = pd.DataFrame(cosine_sim, index=r_matrix.index, columns=r_matrix.index)
```

Теперь по аналогии с предыдущей моделью рассчитаем средневзвешенные оценки. Однако теперь еще необходимо учитывать только ненулевые оценки косинусного сходства. То есть. нам нужно избегать пользователей, не оценивших аниме.

```
def model_weightmean(user_id, anime_id):
  if anime_id in r_matrix:
    similars = cosine_sim[user_id]
    # оценки пользователей для аниме
    m_ratings = r_matrix[anime_id]
    # запишем индексы тех элементов, что имеют нулевое косинусное сходство
    idx = m_ratings[m_ratings.isnull()].index
    #удалим оценки, которые содержат Nan
    m_ratings = m_ratings.dropna()
    # удалим косинусное сходство элементов, которые содержат Nan
    similars = similars.drop(idx)
```

```
    # рассчитаем средневзвешенное по формуле
    weightmean = np.dot(similars, m_ratings)/ similars.sum()
  else:
    weightmean = 5.0
  return weightmean
```

Время обработки данной модели значительно больше, чем у предыдущей модели. Тем не менее удалось добиться (небольшого) улучшения показателя RSME равного 4.019.

2.8.1 Модель кластеризации

Попробуем построить модель на основе кластеризации.

Для кластеризации используем алгоритм k-средних, а затем будем использовать для рекомендаций только пользователей из одного кластера.

Нам необходимо решить следующие задачи:

— Найти k-ближайших соседей, у которых есть рейтинг аниме a.

— Вывести средний рейтинг k пользователей для аниме a.

А также подберем гиперпараметры KNN, которыми выступают:

— Число соседей (n$_neighbors$); $-$;

— Степень для метрики Минковского (p) — 1 или 2 часто лучшие. Когда p = 1, это эквивалентно использованию manhattan$_distance(l1)euclidean_distance(l2)p = 2$.

```
def model_weightmean(user_id, anime_id):
from sklearn.neighbors import KNeighborsRegressor
from sklearn.model_selection import GridSearchCV


knn = KNeighborsRegressor()


param_grid = [
{'n_neighbors': range(1, 10, 1), 'metric' : ['cosine']},
{'n_neighbors': range(1, 10, 1), 'metric' : ['minkowski'],
'p': [1, 2, 3]},
]
grid_search = GridSearchCV(knn, param_grid, cv=5, n_jobs = -1, verbose = 1)
best_model = grid_search.fit(X_train, y_train)

```

В данном коде импортируется регрессионный алгоритм KNN и оценщик GridSearchCV для подбора параметров из библиотеки sklearn (строки 1-2). В строке 4 создается регрессор. На 6-10 строке приведен словарь гиперпараметров на которых будет работать оценщик. Опробуем регрессор с косинусной метрикой и числом соседей от 1 до 10 с шагом 1, а также регрессор с метрикой Минковского, степенью этой метрики от 1 до 3, а также с числом соседей от 1 до 10 с шагом 1. Далее объявляется оценщик с заданным словарем гиперпараметров и параметром cv = 5. Это означает, что здесь используется пятикратная кросс-валидация. Следовательно датасет делится на пять частей, где 4 используются для обучения, а пятый для проверки. Таким образом оценщик проходит датасет пять раз, чтобы каждая часть была проверочной ровно один раз.

После обучения наилучшими параметрами оценщик выбрал: косинусную метрику, степень метрики 2, число соседей равное одному:

```
print('Best metric:', best_model.best_estimator_.get_params()['metric'])
print('Best p:', best_model.best_estimator_.get_params()['p'])
print('Best n_neighbors:', best_model.best_estimator_.get_params()['n_neighbors'])
```

Сделаем предсказание на тестовой выборке и оценим мерой RMSE:

```
y_true = np.array(X_test['rating'])
np.array(X_test['rating'])
y_pred = best_model.predict(X_test)
mean_squared_error(y_true, y_pred, squared=False)
```

Наблюдаем, что RSME, полученный этой моделью, равен 0.94. Это лучший результат, который был достигнут.

2.8.2 Нейронная сеть

Построение модели нейронной сети, как и описывалось в теории, начинается с уровня embedding. Здесь векторы пользователей и аниме уплотняются и преобразуются:

```
embedding_size = 128
 user = Input(name = 'user', shape = [1])
 user_embedding = Embedding(name = 'user_embedding',
                    input_dim = n_users,
                    output_dim = embedding_size)(user)
```

```
anime = Input(name = 'anime', shape = [1])
anime_embedding = Embedding(name = 'anime_embedding',
                    input_dim = n_animes,
                    output_dim = embedding_size)(anime)

```

Далее они объединяются и выравниваются:

```
x = Dot(name = 'dot_product', normalize = True, axes = 2)([user_embedding,
    anime_embedding])
x = Flatten()(x)
```

```
x = Dense(1, kernel_initializer='he_normal')(x)
x = BatchNormalization()(x)
```

Наконец, последним слоем является функция активации. Другие функции не рассматриваются (например, elu, relu), так как такая вариация занимает слишком много времени для обучения (порядка 6 часов), а все рассмотренные функции, кроме сигмоиды, вызывают переобучение. Поэтому было решено использовать только один вариант.

```
x = Activation(hp.Choice('activation', values=['sigmoid']))(x)
```

В качестве функции потерь была выбрана бинарная кросс-энтропия. Бинарная кросс-энтропия показывает, насколько отличается изначальный рейтинг от рейтинга, предсказанного нейросетью.

В качестве оптимизатора предложено три варианта. Таким образом будет построено три модели нейронной сети:

```
model.compile(loss='binary_crossentropy', metrics=["mae", "mse"],
    optimizer=hp.Choice('optimizer', values=['adam','rmsprop','SGD']))
```

Для обучения использован BayesianOptimization, в качестве метрики для оптимизации выбрана mse, т.к. именно она позже будет использована для сравнения с результатами другого способа построения рекомендательной системы.

```
tuner = BayesianOptimization(
    RecommenderNet,
    objective='val_mse',
    max_trials=10,
    directory='test_directory'
    )
```

Обучение производится с использование следующих параметров:

```
tuner.search(X_train_array,
             y_train,
             batch_size=10000,
             epochs=10,
             validation_split=0.2,
             )

```

Изначально сеть обучалась на 20 эпохах, но первые попытки показали, что после 10-ой эпохи во всех моделях улучшение метрик прекращается, поэтому было решено уменьшить это значение.

Результаты обучения следующие:

Лучше всего себя показала модель с оптимизатором Adam, хотя и результаты с rmsprop не сильно хуже. Score вышел действительно маленьким (меньше 0.1) это говорит о правильно составленной нейронной сети и удачном подборе параметров. Осталось проверить эти результаты на тестовой выборке:

Проверка на тестовых данных подтвердила результаты обучения.

2.9 Результаты

В результате проделанной работы были изучены и реализованы различные методы построения рекомендательной системы.

Сравнение методов построения системы производится с помощью меры RMSE. В нейронной сети подсчитывался параметр MSE, однако из него легко получить требуемый для сравнения:

Таблица сравнения построенных методов в работе:

Таблица 1 – Результат сокращения словарей неисправностей при помощи масок

Название метода	Мера RMSE
Базовый	4.176227963
Коллаборативный на основе среднего значения	4.019703832
Коллаборативный на основе средневзвешенного значения	4.019703832
Нейронная сеть (Adam)	0.247386338
Нейронная сеть (rmsprop)	0.252922913
Нейронная сеть (SGD)	0.374432905

2.10 Практическая деятельность

Внедрение результата в практическую деятельность в рекомендательной системе можно представить наглядно. Для этого строится функция предсказания. Построенные модели позволяют показать два варианта систем: Item-based и user-based.

2.10.1 Item-based на основе нейронной сети

Для начала получаются веса аниме из модели:

```
def extract_weights(name, model):
    weight_layer = model.get_layer(name)
    weights = weight_layer.get_weights()[0]
    weights = weights / np.linalg.norm(weights, axis = 1).reshape((-1, 1))
    return weights


anime_weights = extract_weights('anime_embedding', model)
```

Именно на основе этих значений будет строится вектор расстояний до различных аниме, которые отражают схожесть.

В функции вектор со всеми весами умножается на вес текущего аниме для нахождения вектора расстояний. Далее берется n самых больших и наиболее подходящих для заявленного аниме:

```
index = getAnimeFrame(name).anime_id.values[0]
weights = anime_weights
dists = np.dot(weights, weights[index])
sorted_dists = np.argsort(dists)
if neg:
  closest = sorted_dists[:(n + 1)]
else:
  closest = sorted_dists[(-n - 1):]
```

Результатом вызова функции

```
find_similar_animes('xxxHOLiC', n=10,  neg=False)
```

будет следующая таблица:

2.10.2 User-based подход в коллаборативном KNN

Для использования модели создан метод predict. На вход он получает id пользователя и по нему рекомендует аниме.

```
1  knn = KNeighborsRegressor(metric='cosine', p = 2, n_neighbors=1)
2  n_knn = knn.fit(X_train, y_train)
3  def predict(user_id):
4    test_set = X[X['user_id'] == user_id]
5    distances, indeces =  n_knn.kneighbors(test_set)
6    final_table = pd.DataFrame(distances, columns = ['distance'])
7    final_table['index'] = indeces
8    final_table = final_table.set_index('index')
9    result = final_table.join(X_train,on='index')
10   result = result.join(synopsis_df, on='anime_id')
11   return result[['distance','Name', 'Score', 'Genres']].head(5)
```

На строках 1-2 производится создание обучение модели с подобранными гиперпараметрами. В строке 4 выбираются те аниме из датасета, что были оценены выбранным пользователем. Функция kneighbors возвращает из обученного KNN расстояния и индексы (в обучающем наборе) сходных записей. На строках 6-8 создается первоначальная таблица, куда записываются полученные из алгоритма расстояния и индексы. Затем на строках 9-10 по индексу данная таблица объединяется с другой, содержащей подробную информацию о рекомендуемом аниме. В итоге метод возвращает таблицу, где представлен индекс аниме, его метрика, название, оценка и метки жанров. К примеру функция для пользователя с id 573 вернет:

ЗАКЛЮЧЕНИЕ

В настоящей работы приведен пример оформления студенческой работы средствами системы LaTeX.

Показано, как можно оформить документ в соответствии:

— с правилами оформления курсовых и выпускных квалификационных работ, принятых в Саратовском государственном университете в 2012 году;

— с правилами оформления титульного листа отчета о прохождении практики в соответствии со стандартом.

ПРИЛОЖЕНИЕ А
Нумеруемые объекты в приложении

Таблица 2 – Results of pass-fail dictionary reduction with the help of masks

Circuit	Number of modelled faults	Number of test vectors in the test set	The volume of pass-fail dictionary, bit	The volume of found mask	The volume of masked dictionary, bit	% of pass-fail dictionary	CPU running time, min
S298	177	322	56994	30	5310	9,32%	0,07
S344	240	127	30480	29	6960	22,83%	0,04
S349	243	134	32562	35	8505	26,12%	0,05
S382	190	2074	394060	28	5320	1,35%	0,43
S386	274	286	78364	65	17810	22,73%	0,26
S400	194	2214	429516	32	6208	1,45%	0,99
S444	191	2240	427840	30	5730	1,34%	0,98
S526	138	2258	311604	28	3864	1,24%	0,61
S641	345	209	72105	58	20010	27,75%	0,24
S713	343	173	59339	58	19894	33,53%	0,19
S820	712	1115	793880	147	104664	13,18%	9,09
S832	719	1137	817503	151	108569	13,28%	9,20
S953	326	14	4564	13	4238	92,86%	0,01
S1423	293	150	43950	58	16994	38,67%	0,15
S1488	1359	1170	1590030	158	214722	13,50%	26,69

$$F(x) = \int_a^b f(x)\,dx. \tag{1}$$

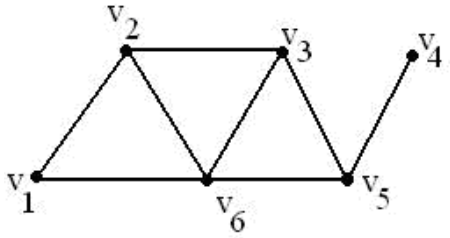

Рисунок 1 – Подпись к рисунку

Таблица 3

0	1
1	0

ПРИЛОЖЕНИЕ Б

Листинг программы

Код приложения task.pl.

```
use locale;
use encoding "cp866";
{
    print "Имя папки: "; my $folder_name=<>;
    chomp($folder_name);
    my @files = `chcp 866 & attrib $folder_name\\*.pl`;
    if (substr($files[1],0,15) eq 'Не найден путь:') {
        print "Путь не найден. Попробуйте еще.\n";
        redo;
    }
    elsif (substr($files[1],0,15) eq 'Не найден файл:') {
        print "Папка не содержит файлов .txt .\n";
        last;
    }
    else {
        foreach my $file (@files[1 .. $#files]){
            my $file_name = substr($file, 11);
            chomp($file_name);
            open(FH,"<$file_name") or die $!;
            my %hash = ();
            foreach $chunk (<FH>){
                my @words = $chunk =~ /([\@\%\$][a-zA-Z_0-9]+[\[\{]?)/g;
                foreach my $word (@words) {
                    $word = "\$".substr($word, 1)
                      if (substr($word, 0, 1) eq '@' &&
                          substr($word, -1) eq '[');
                    $word = "\$".substr($word, 1)."["
                      if (substr($word, 0, 1) eq '@');
                    $word = "\$".substr($word, 1)."{"
                      if (substr($word, 0, 1) eq '%');
                    $hash{$word}++;
                };
            };
            my @xs = keys %hash;
            print @xs;
            close(FH);
            my $ans = scalar(@xs);
```

```
38                print "$file_name : $ans\n";
39            }
40        }
41    }
```

ПРИЛОЖЕНИЕ В
Многостраничная таблица

Таблица 4 – ГОСТ DIN ISO — Таблица соответствия стандартов

Стандарт ГОСТ	Наименование	Стандарт DIN	Стандарт ISO
1	2	3	4
ГОСТ 397-79	Шплинты	DIN 94	ISO 1234
ГОСТ 1144-80	Шурупы с полукруглой головкой	DIN 96DIN 7981	ISO 7049
ГОСТ 1145-80	Шурупы с потайной головкой	DIN 97DIN 7982	ISO 7050
ГОСТ 1146-80	Шурупы с полупотайной головкой	DIN 95DIN 7983	ISO 7051
ГОСТ 1476-93	Винты установочные с коническим концом и прямым шлицем классов точности А и В	DIN 553	ISO 7434
ГОСТ 1477-93	Винты установочные с плоским концом и прямым шлицем классов точности А и В	DIN 438DIN 551	ISO 4766ISO 7436
ГОСТ 1478-93	Винты установочные с цилиндрическим концом и прямым шлицем классов точности А и В	DIN 417	ISO 7435
ГОСТ 1481-84	Винты установочные с шестигранной головкой и цилиндрическим концом классов точности А и В	DIN 561	
ГОСТ 1482-84	Винты установочные с квадратной головкой и цилиндрическим концом классов точности А и В	DIN 479	
ГОСТ 1485-84	Винты установочные с квадратной головкой и засверленным концом классов точности А и В	DIN 479	

Продолжение таблицы **??**

1	2	3	4
ГОСТ 1486-84	Винты установочные с квадратной головкой и ступенчатым концом со сферой классов точности А и В	DIN 480	
ГОСТ 1488-84	Винты установочные с квадратной головкой и буртиком классов точности А и В	DIN 478	
ГОСТ 1491-80	Винты с цилиндрической головкой классов точности А и В	DIN 84	ISO 1207
ГОСТ 3032-76	Гайки-барашки	DIN 315	
ГОСТ 3033-79	Болты откидные	DIN 444	
ГОСТ 3057-90	Пружины тарельчатые	DIN 2093	
ГОСТ 3070-88	Канат стальной двойной свивки типа ТК конструкции 6х19 (1+6+12)+1 о.с.	DIN 3060	
ГОСТ 3128-70	Штифты цилиндрические незакаленные	DIN 7DIN 6325	ISO 2338ISO 8734
ГОСТ 3129-70	Штифты конические незакаленные	DIN 1	ISO 2339
ГОСТ 4751-73	Рым-болты	DIN 580	ISO 3266
ГОСТ 5915-70	Гайки шестигранные стальные класса точности В	DIN 555DIN 934	ISO 4032ISO 4033ISO 8673ISO 8674
ГОСТ 5916-70	Гайки шестигранные низкие класса точности В	DIN 439DIN 936	ISO 4035ISO 4036ISO 8675

Продолжение таблицы **??**

1	2	3	4
ГОСТ 5918-73	Гайки шестигранные прорезные и корончатые класса точности В	DIN 935	EN ISO 7035EN ISO 7036EN ISO 7037
ГОСТ 5919-73	Гайки шестигранные прорезные и корончатые низкие класса точности В	DIN 937	EN ISO 7038
ГОСТ 5927-70	Гайки шестигранные класса точности А	DIN 555DIN 934	ISO 4032ISO 4034ISO 8673
ГОСТ 5932-73	Гайки шестигранные прорезные и корончатые класса точности А	DIN 935DIN 937	EN ISO 7035EN ISO 7036EN ISO 7037
ГОСТ 6393-73	Гайки круглые с отверстиями на торце "под ключ"класса точности А	DIN 1816	
ГОСТ 6402-70	Шайбы пружинные	DIN 127	
ГОСТ 6958-78	Шайбы увеличенные. Классы точности А и С	DIN 440DIN 9021	ISO 7094ISO 7093-1ISO 7093-2
ГОСТ 7786-81	Болты с потайной головкой и квадратным подголовком класса точности С	DIN 608	
ГОСТ 7795-70	Болты с шестигранной уменьшенной головкой и направляющим подголовком, класс точности В		