Тихоновс Василийс,
Выпускник Санкт-Петербургского политехнического университета
Петра Великого

Сечинский Егор Валерьевич,
Выпускник Санкт-Петербургского политехнического университета
Петра Великого

ПРИМЕНЕНИЕ АЛГОРИТМА SVD ДЛЯ ПОСТРОЕНИЯ РЕКОМЕНДАТЕЛЬНОЙ СИСТЕМЫ

Аннотация: в данной статье отображены результаты исследования рекомендательного алгоритма на основе SVD. В исследовании оценивается оптимальность данной системы.

Ключевые слова: рекомендационная система, оценка, фильтрация, прогноз, машинное обучение, SVD, основаная на содержимом, гибридная фильтрация, PYTHON.

Введение

В современном информационном обществе объем доступной информации постоянно растет, что может затруднять пользователям находить релевантный контент и товары, отвечающие их индивидуальным интересам и предпочтениям. В этом контексте рекомендательные системы становятся все более востребованными инструментами, обеспечивая персонализированные рекомендации и содействуя удовлетворению потребностей пользователей [3, 5].

В данной статье рассмотрена реализация рекомендательного алгоритма на основе сингулярного разложения матриц(SVD).

Работа имеет практическое значение для компаний, которые заинтересованы в создании персонализированных рекомендаций для своих пользователей. Результаты исследования могут быть использованы при выборе оптимального подхода к персонализации рекомендаций, а также при разработке и реализации алгоритмов машинного обучения для этой цели [4].

Сингулярное разложение матриц

Сингулярное разложение матрицы (Singular Value Decomposition, SVD) - это метод разложения матрицы. Теорема о сингулярном разложении матрицы [1] гласит, что у любой матрицы A размера $m \times n$ существует разложение в призведение трех матриц:

$$\underset{n \times m}{A} = \underset{n \times m}{U} \times \underset{n \times m}{\sum} \times \underset{n \times m}{V^T}, \quad (1)$$

Матрицы U и V ортогональные, а $\sum$ – диагональная.

$$UU^T = I_n, VV^T = I_M,$$

$$\sum = diag(\lambda_1, \dots, \lambda_{\min(n,m)}), \lambda_1, \dots, \lambda_{\min(n,m)} \geq 0.$$

Помимо обычного разложения существует усеченное разложение, когда из всех значений λ остаются только первые d значений, а остальные приравниваются к нулю:

$$\lambda_{d+1}, \dots, \lambda_{\min(n,m)} := 0.$$

Таким образом, у матриц U и V остаются первые d столбцов, а матрица $\sum$ становится квадратной.

$$\underset{n \times m}{A'} = \underset{n \times d}{U'} \times \underset{d \times d}{\Sigma'} \times \underset{d \times m}{V'^T}. \quad (2)$$

Матрица A' является ее наилучшим низкоранговым приближением матрицы A с точки зрения средне-квадратичного отклонения.

Посредством усеченного разложения получаем приблизительное разложение матрицы A. Можно упростить разложение, обозначив произведение $\underset{n \times d}{U'} \times \underset{d \times d}{\Sigma'}$ как одну матрицу (рисунок 1).

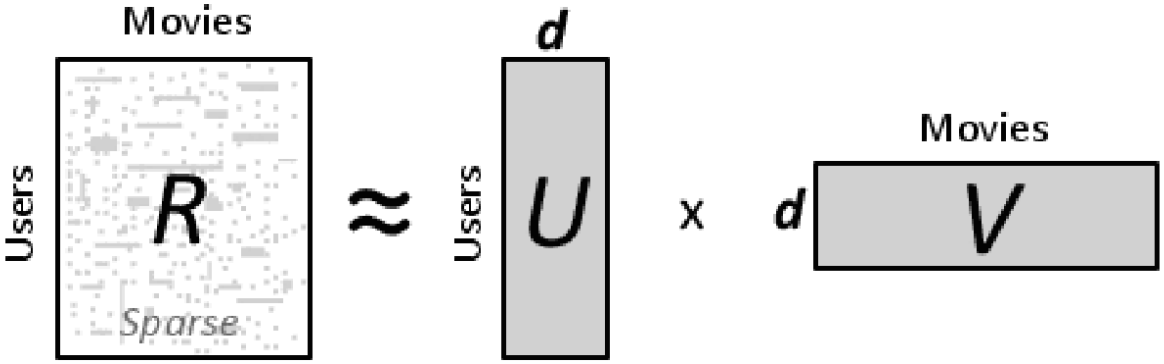

Рисунок 1 – Обозначение U'*E' за одну матрицу

Чтобы предсказать оценку пользователя U для объекта I, обозначим набор параметров для пользователя как вектор $\bar{p}_u$ и вектор $\bar{q}_u$. Скалярное произведение этих веткоров является предсказанием:

$$\hat{r}_{ui} = \langle \bar{p}_u, \bar{q}_u \rangle. \quad (3)$$

Использование данного алгоритма позволяет не только предсказывать оценки, но и выявлять скрытые признаки объектов и пользователей. На каждой координате вектора пользователя получаем оценку, отражающую свойство самого пользователя. У объекта на соответствующей позиции будет оценка, указывающая на релевантность объекта пользователю по данному свойству.

Машинное обучение для работы с неполными данными

Решить проблему с тем, что матрица оценок R полностью не известна, можно при помощи машинного обучения, для чего нужно составить модель предсказания, которая будет работать схожим с SVD образом. Такая модель зависит от множества параметров: векторов пользователей $\bar{p}_u$ и векторов объектов $\bar{q}_u$. При заданных параметрах необходимо получить скалярное произведение $\bar{p}_u$ и $\bar{q}_u$:

$$\hat{r}_{ui}(\Theta) = p_u^T q_i,$$
$$\Theta = \{p_u, q_i \mid u \in U, i \in I\}. \quad (4)$$

Однако полного набора векторов не существует, его нужно получить. При помощи оценок пользователей можно найти такие оптимальные параметры, при которых модель будет предсказывать оценки максимально точно:

$$E_{(u,i)}(\hat{r}_{ui}(\Theta) - r_{ui})^2 \to \min_{\Theta}. \quad (5)$$

Нужно подобрать параметры так, чтобы на тех полученных оценках ошибка была как можно меньше. Также следует добавить регуляризатор для борьбы с переобучением. Регуляризация заключается том, что следует оптимизировать не только ошибку, но и сумму ошибки и некоторой функции от параметров (например, норму вектора параметров). Это позволяет ограничить размер параметров в решении, уменьшает степень свободы модели:

$$\sum_{(u,i)\in D} (\hat{r}_{ui}(\Theta) - r_{ui})^2 + \lambda \sum_{\theta\in\Theta} \theta^2 \to \min_{\Theta}. \quad (6)$$

Чтобы найти оптимальные параметры, нужно оптимизировать функционал:

$$J(\Theta) = \sum_{(u,i)\in D} (p_u^T q_i - r_{ui})^2 + \lambda(\sum_u ||p_u||^2 + \sum_i ||q_i||^2). \quad (7)$$

Для каждого пользователя и объекта существует свой вектор, который необходимо оптимизировать. Функцию, зависящую от большого количества переменных, можно найти при помощи метода градиентного спуска [2], для этого требуется построить градиент — вектор из частных производных по каждому параметру:

$$gradJ(\Theta) = (\frac{\partial J}{\partial \theta_1}, \frac{\partial J}{\partial \theta_2}, \dots, \frac{\partial J}{\partial \theta_n})^T. \quad (8)$$

Метод градиентного спуска (рисунок 2) - это численный метод, который применяется для нахождения локального экстремума при помощи движения вдоль градиента. Таким образом получаем:

$$\Theta_{t+1} = \Theta_t - \eta \, grad \, J(\Theta). \quad (9)$$

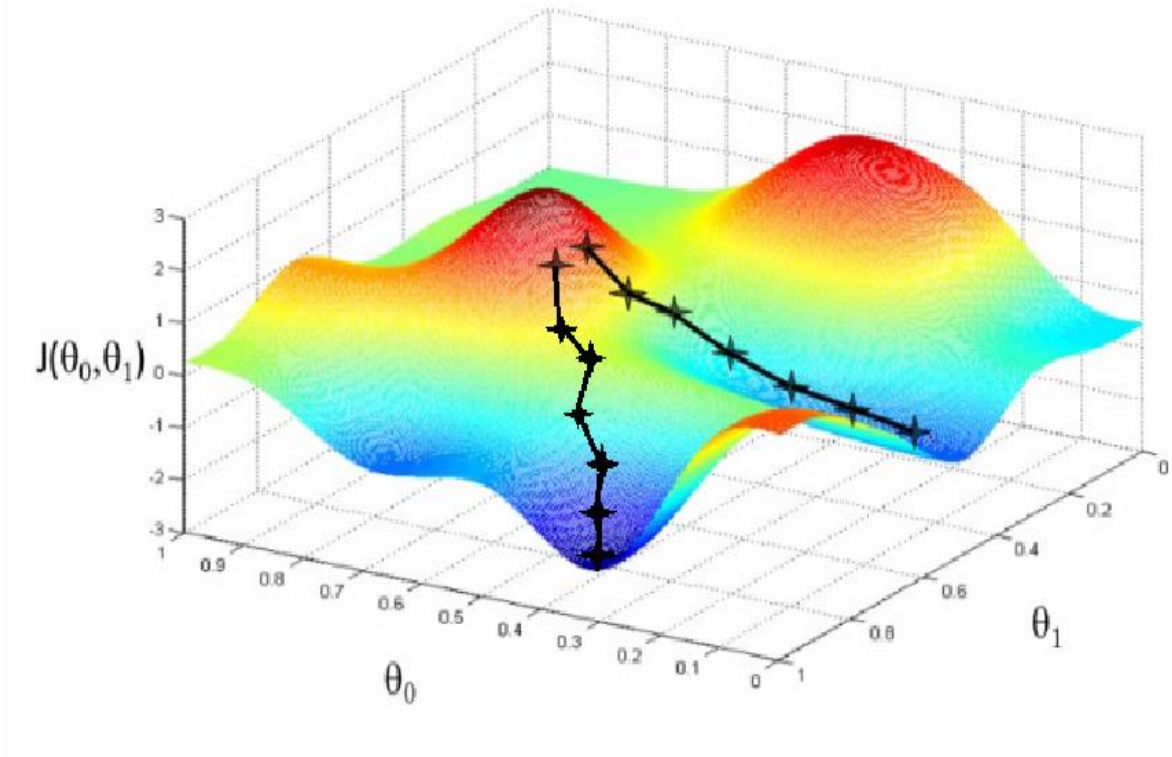

Рисунок 2 – Визуализация метода градиентного спуска

Подготовка данных

Для реализации данной системы был использован набор данных MovieLens 100K. MovieLens - это набор данных с рейтингами фильмов с сайта MovieLens, который был собран за определенный период времени. Этот набор данных основан на базе данных сервиса imdb.

Набор данных состоит из 100 000 оценок от 1 до 5 от 943 пользователей для 1682 фильмов, каждый пользователь оценил хотя бы 20 фильмов. Датасет содержит информацию о пользователях и фильмах с указанием жанра, к которому

принадлежит фильм. Информация о распределении оценок пользователями представлена на рисунке 4. Можно сделать вывод, что в данном случае пользователи предпочитают ставить просмотренным фильмам оценку 4.

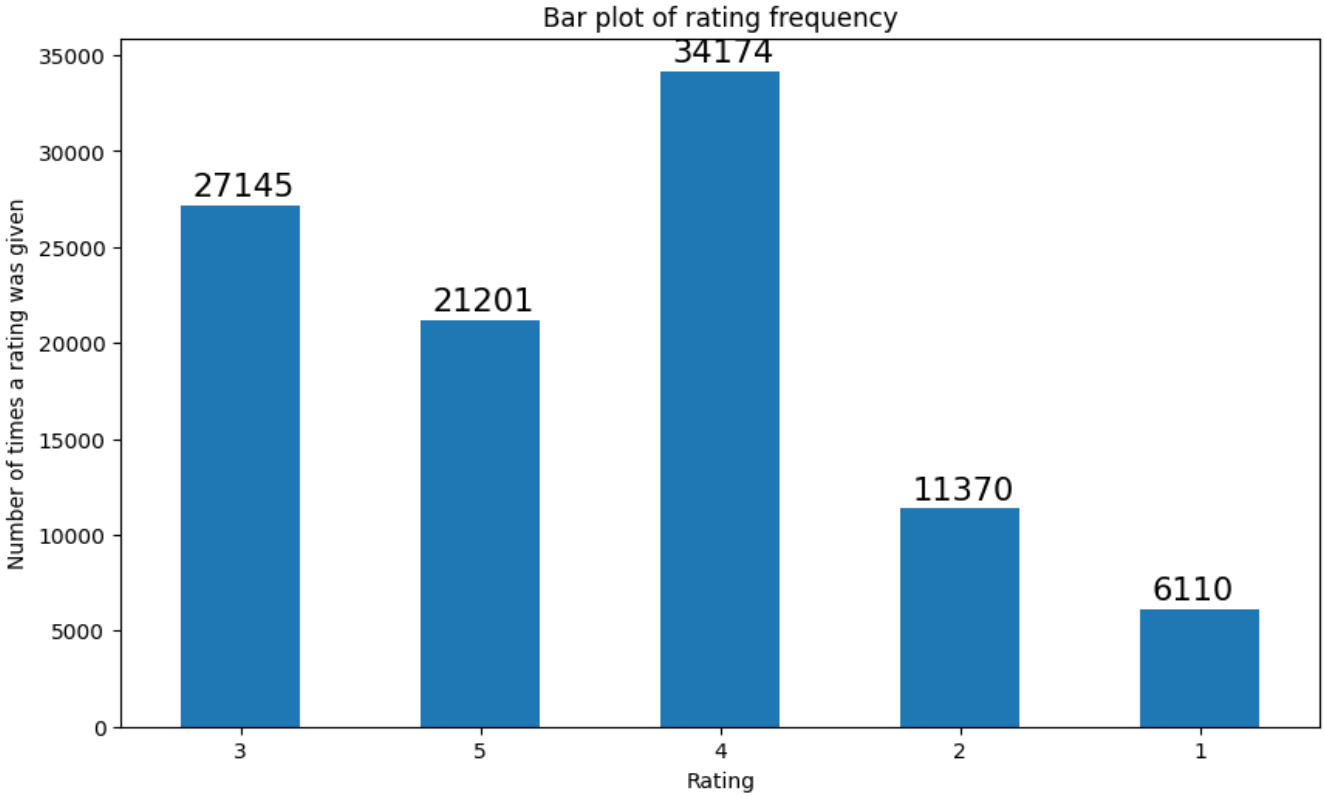

Рисунок 4 – Распределение оценок

Реализация на Python

Матричная факторизация - это математическая операция для матриц. Она обычно более эффективна в коллаборативной фильтрации, потому что позволяет обнаруживать латентные (скрытые) особенности, лежащие в основе взаимодействий между пользователями и элементами (фильмами).

Матрица полезности может быть сформирована из существующего объединенного датасета и нормализована по сущности (фильму или пользователю), с которой необходимо найти сходство (рисунок 5).

```
utility_matrix = np.asarray([[np.nan for j in range(len(unique_users))] for i in range(len(unique_movies))])
print("Shape of Utility matrix: ",utility_matrix.shape)

for i in range(len(ratings_list)):

  ## ith entry in users list and subtract 1 to get the index, we do the same for movies but we already defined a dictionary to get the index.
  utility_matrix[movies_dict[movie_list[i]]][users_list[i]-1] = ratings_list[i]

utility_matrix
```

```
Shape of Utility matrix:  (1664, 943)
array([[ 2., nan, nan, ..., nan, nan, nan],
       [ 5., nan, nan, ..., nan, nan, nan],
       [ 3., nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

```
mask = np.isnan(utility_matrix)
masked_arr = np.ma.masked_array(utility_matrix, mask)
temp_mask = masked_arr.T
rating_means = np.mean(temp_mask, axis=0)

filled_matrix = temp_mask.filled(rating_means)
filled_matrix = filled_matrix.T
filled_matrix = filled_matrix - rating_means.data[:,np.newaxis]
```

```
filled_matrix = filled_matrix.T / np.sqrt(len(movies_dict)-1)
filled_matrix
```

```
array([[-0.02227217,  0.01608636, -0.01226094, ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       ...,
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ],
       [ 0.        ,  0.        ,  0.        , ...,  0.        ,
         0.        ,  0.        ]])
```

Рисунок 5 – Создание нормализованной матрицы полезности

Сингулярное разложение выполняется на матрице полезности и латентных признаках строк и столбцов (в данном случае, фильмов и пользователей) при помощи библиотеки numpy (рисунок 6).

```
## Computing the SVD of the input matrix

U, S, V = np.linalg.svd(filled_matrix)
```

Рисунок 6 – Вычисление SVD

В декомпозиции SVD, где A - это m x n матрица полезности, U - это m x r ортогональная левая сингулярная матрица, которая представляет собой отношение между пользователями и латентными факторами, S - это r x r диагональная

матрица, которая описывает силу каждого латентного фактора, а V - это r x n диагональная правая сингулярная матрица, которая указывает на сходство между элементами и латентными факторами. Латентные факторы здесь - это характеристики элементов, например, жанр. SVD уменьшает размерность матрицы полезности A, извлекая ее латентные факторы.

SVD снижает размерность и отображает важные латентные особенности, которые могут аппроксимировать все значения в матрице полезности (включая нулевые значения). Уже имеющиеся значения в матрице полезности могут быть использованы для оценки прогнозов и корректировки параметров в латентных особенностях (если используется матричная факторизация), или помочь выбрать количество латентных особенностей из разложения SVD. Необходимо решить, сколько латентных особенностей следует выбрать, чтобы заполнить нулевые значения и построить рекомендации.

Определение функции для расчета косинусного сходства на данном датафрейме и извлечение запрошенного количества тесно связанных индексов фильмов выполнена с помощью библиотеки numpy einsum, которая оценивает конвенцию суммирования Эйнштейна на операндах (рисунок 7).

```
def top_cosine_similarity(data, movie_id, top_n=10):
  index = movie_id
  movie_row = data[index, :]
  magnitude = np.sqrt(np.einsum('ij, ij -> i', data, data))
  similarity = np.dot(movie_row, data.T) / (magnitude[index] * magnitude)
  sort_indexes = np.argsort(-similarity)
  return sort_indexes[:top_n]
```

Рисунок 7 – Функция расчета косиусного сходства

Результат работы данного алгоритма на представленном датасете отображен на рисунке 8.

```
Enter the Movie name: 101 Dalmatians (1996)
Enter Number of movie recommendations needed: 10
Top 10 movies which are very much similar to the Movie- 101 Dalmatians (1996) are:

Black Beauty (1994)
Free Willy 2: The Adventure Home (1995)
Evening Star, The (1996)
Robin Hood: Men in Tights (1993)
Cool Runnings (1993)
Turbo: A Power Rangers Movie (1997)
Remains of the Day, The (1993)
City Hall (1996)
Children of the Corn: The Gathering (1996)
```

Рисунок 8 – Подобранные рекомендации

Для определения качества предсказанных оценок воспользуемся мерой RMSE. Для этого нужно определить функцию оценки (рисунок 9).

```
def rmse(prediction, ground_truth):
    # Оставим оценки, предсказанные алгоритмом, только для соотвествующего набора данных
    prediction = np.nan_to_num(prediction)[ground_truth.nonzero()].flatten()
    # Оставим оценки, которые реально поставил пользователь, только для соотвествующего набора данных
    ground_truth = np.nan_to_num(ground_truth)[ground_truth.nonzero()].flatten()

    mse = mean_squared_error(prediction, ground_truth)
    return sqrt(mse)
```

Рисунок 9 – Функция для оценки

В данном случае использована встроенная в библиотеку sklearn функция mean_squared_error. Оценка точности проведена на тестовой выборке, которая была построена из части исходного датасета. Для проверки работы реализованных методов проведена оценка релевантности рекомендаций. Данный алгоритм показал ошибку 1.42.

ЗАКЛЮЧЕНИЕ

В данной статье рассмотрен один из методов реализации рекомендательных систем – алгоритм SVD. Описана разработка алгоритма и его реализация. Оценка по метрике RMSE указывает на неплохую точность данного подхода.

Список литературы:

1. J. Hopcroft, R. Kannan Foundations of Data Science: учебное пособие // J. Hopcroft, R. Kannan - Cambridge University Press, 2020.

2. Козлов В. Н. Системный анализ, оптимизация и принятие решений: учебное пособие // В. Н. Козлов. – Санкт- Петербург: Издательство Политехнического университета, 2011.

3. Liu et al., 2011 - "Personalized news recommendation based on click behavior": A survey

4. G. Linden. IEEE Internet Computing: Amazon.com recommendations: item-to-item collaborative filtering // G. Linden, B. Smith, J. York. – New York, 2003.

5. J. Bobadilla. Recommender systems survey // J. Bobadilla, F. Ortega, A. Hernando, A. GutieRrez. – Canada, 2013.