Наука и Образование
МГТУ им. Н.Э. Баумана
Сетевое научное издание
ISSN 1994-0408

Наука и Образование. МГТУ им. Н.Э. Баумана. Электрон. журн. 2016. № 07. С. 251–259.

Представлена в редакцию: 06.08.2016
Исправлена: 14.09.2016

УДК 378; 004; 51-78

Пример проектного подхода к обучению в области обработки больших данных на основе построения рекомендательной системы с применением методов коллаборативной фильтрации с использованием Apache Spark и Python

Музалевский Д. С.[1,*], Гапанюк Ю. Е.[1] *racounter18@gmail.com

[1]МГТУ им. Н.Э. Баумана, Москва, Россия

В рамках международного научного конгресса "Наука и инженерное образование. SEE-2016", II международная научно-методическая конференция «Управление качеством инженерного образования. Возможности вузов и потребности промышленности» (23-25 июня 2016 г., МГТУ им. Н.Э. Баумана, Москва, Россия).

Рассмотрен подход построения рекомендательных систем с использованием принципов коллаборативной фильтрации. Обсуждается принцип работы метода чередующихся наименьших квадратов для построения персональных рекомендаций. Описывается реализация метода чередующихся наименьших квадратов с использованием возможностей библиотеки машинного обучения Pyspark.Mllib. Рассматривается использование системы Apache Spark для обработки и анализа данных. Описывается реализация подхода с использованием языка Python.

Ключевые слова: рекомендации, Python, коллаборативная фильтрация, метод Alternative Least Squares, Spark

Введение

Большинство интернет сайтов располагает функцией показа своим пользователям персональных рекомендаций. Рекомендательные системы предсказывают предпочтительность объекта для конкретного пользователя, основывая свой прогноз на данных, указанных пользователем явно, или собранных из истории его взаимодействия с данным сайтом либо с различными другими ресурсами.

Для составления рекомендаций обычно используются методы коллаборативной фильтрации. Когда требуется предсказать, насколько понравится пользователю новый объект, для этого используются оценки похожих клиентов. Эти методы хорошо известны и применяются во многих проектах: Netflix, Amazon, Last.fm. За последние годы качество алгоритмов этого типа значительно увеличилось. Толчком для этого послужило соревнование на лучший алгоритм предсказания, проведенное Netflix. Тем не менее, до сих пор акту-

ально решение проблемы «холодного старта» системы для новых пользователей и объектов, качество и скорость составления рекомендаций.

В этой работе реализован алгоритм Чередующихся Наименьших Квадратов (Alternating Least Square). Оценки пользователей различных кинофильмов сайта MovieLens.org послужили данными для работы.

Рассмотрена работа системы Apache Spark для обработки и анализа данных рекомендаций, а также библиотеки Pyspark.MLlib, в состав которой входит вышеупомянутый метод ALS.

Описание предметной области

Рекомендательные системы предназначены для поиска объектов, которые понравятся пользователю или будут полезны. В типичных системах есть список пользователей $U = (u_1, u_2, \ldots, u_m)$ и предметов $I = (i_1, i_2, \ldots, i_n)$. В ходе взаимодействия с системой пользователи контактируют с объектами, формируя матрицу рейтингов R, где $r_{u_a,i}$ – рейтинг предмета $i \in I$ у пользователя $u_a \in U$. Матрица рейтингов сильно разрежена, т.к. количество различных предметов в системе велико и уже известные u_a предметы $I_{u_a} \subseteq I$ составляют малую долю от общего количества. Задача рекомендательной системы формулируется как вычисление предсказания и рекомендации для заданного пользователя (user based recommendations) или товара (item based recommendations). Существует много разновидностей подходов к построению рекомендаций. Из основных и наиболее часто используемых в настоящее время стоит выделить принцип коллаборативной фильтрации (Alternating Least Squares, Cosine Similarity, Tanimoto Similarity), принцип ассоциативного анализа (Appriori algorithm), а также метод, основанный на графах сходства (Random Walk with Restarts algorithm). В настоящей работе, мы будем рассматривать рекомендации, основанные на коллаборативной фильтрации, поскольку в настоящее время этот подход является основным в построении рекомендаций, достаточно простым и в то же время весьма эффективным. Реализация будет осуществляться на основе алгоритма Наименьших Чередующихся Квадратов (Alternating Least Squares, ALS), который доступен в библиотеке для машинного обучения системы Spark – pyspark.MLlib.

Описание принципов коллаборативной фильтрации

Идея коллаборативной фильтрации заключается в формировании списка рекомендованных объектов на основе мнений пользователей, ведущих себя похожим образом. Анализируя профили, рекомендательная система находит таких пользователей, после чего оценка предпочтимости нового объекта рассчитывается с использованием их оценок. Существует два подхода к коллаборативной фильтрации: основанные на пользователях (user-based) и на предметах (item-based). Первые оперируют схожестью пользователей, вторые – схожестью предметов.

Схожесть рассчитывается как коэффициент корреляции между многомерными векторами, векторами рейтингами треков для user-based и векторами рейтингов трека у различ-

ных пользователей для item-based методов. Для этого может использоваться коэффициент корреляции Пирсона или косинус угла между векторами. Для них дополнительным плюсом является нормированность, так как значения укладываются в [0,1]. Для item-based методов хорошо зарекомендовал себя уточненный косинус угла, в котором из рейтингов $r_{u,i}$ вычитаются средние значения рейтинга r^- для данного пользователя, как показано в выражении.

Это помогает учесть различные подходы к составлению рейтинга у пользователей, одни из которых могут оперировать лишь высокими оценками, а другие выставлять их предметам.

$$sim(i,j) = \frac{\sum_{u \in U_{i,j}} (r_{u,i} - \bar{r_u})(r_{u,j} - \bar{r_u})}{\sqrt{\sum_{u \in U} (r_{u,i} - \bar{r_u})^2} \sqrt{\sum_{u \in U_{i,j}} (r_{u,j} - \bar{r_u})^2}}$$

Здесь $U_{i,j} \subseteq U$ – множество пользователей, оценивших как предмет i, так и j. Схема принципа коллаборативной фильтрации, отражена на рис. 1.

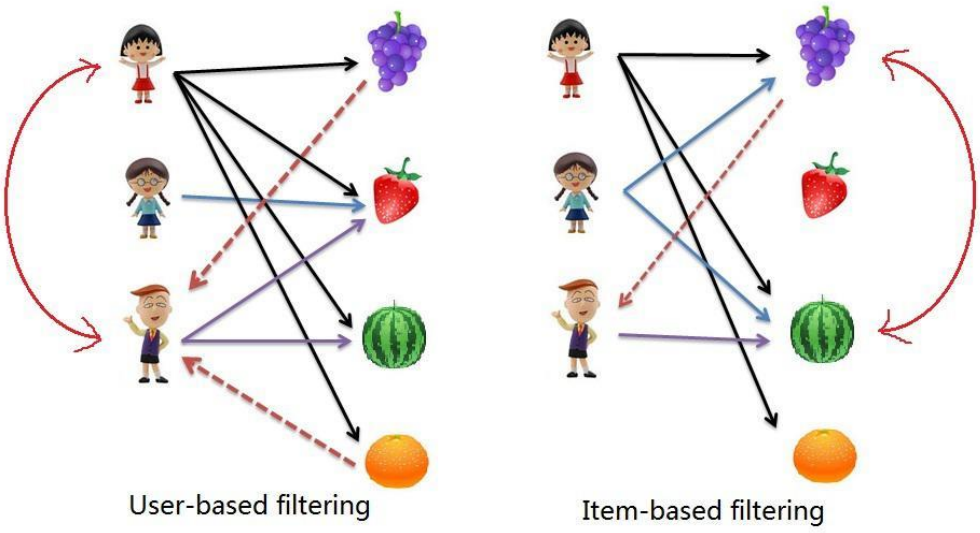

Рис. 1. User-Based и Item-Based принципы фильтрации

Главными проблемами коллаборативной фильтрации являются размер и разреженность матрицы рейтингов, и проблема холодного старта – расчет рекомендации для новых пользователей и новых предметов. Методы фильтрации содержимого этих недостатков лишены. Тем не менее, тот факт, что учитывается история предыдущих оценок, делает коллаборативную фильтрацию крайне эффективной и популярной.

Факторизация Матриц и Метода Чередующихся Наименьших Квадратов

В теореме о сингулярном разложении утверждается, что у любой матрицы **A** размера $n \times m$ существует разложение в произведение трех матриц: U, Σ и V^T

$$\underset{n \times m}{\boldsymbol{A}} = \underset{n \times n}{\boldsymbol{U}} \times \underset{n \times m}{\boldsymbol{\Sigma}} \times \underset{m \times m}{\boldsymbol{V}^T}$$

Матрицы U и V ортогональные, а Σ — диагональная (хотя и не квадратная).

$$\boldsymbol{U}\boldsymbol{U}^T = \boldsymbol{I}_n, \quad \boldsymbol{V}\boldsymbol{V}^T = \boldsymbol{I}_m,$$

$$\boldsymbol{\Sigma} = \mathsf{diag}\big(\lambda_1, \ldots, \lambda_{\min(n,m)}\big), \quad \lambda_1 \geqslant \ldots \geqslant \lambda_{\min(n,m)} \geqslant 0$$

Причем лямбды в матрице Σ будут упорядочены по невозрастанию. Сейчас мы эту теорему не будем доказывать, просто воспользуемся самим разложением.

Помимо обычного разложения, бывает еще усеченное, когда из лямбд, остаются только первые d чисел, а остальные мы полагаем равными нулю

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

Это равносильно тому, что у матриц U и V мы оставляем только первые d столбцов, а матрицу Σ обрезаем до квадратной $d \times d$.

$$\underset{n\times m}{\boldsymbol{A}'} = \underset{n\times d}{\boldsymbol{U}'} \times \underset{d\times d}{\boldsymbol{\Sigma}'} \times \underset{d\times m}{\boldsymbol{V}'^T}$$

Оказывается, что полученная матрица $\mathbf{A}'$ хорошо приближает исходную матрицу $\mathbf{A}$ и, более того, является наилучшим низкоранговым приближением с точки зрения среднеквадратичного отклонения.

Чтобы предсказать оценку пользователя U для фильма I, мы берем некоторый вектор p_u (набор параметров) для данного пользователя и вектор для данного фильма q_i. Их скалярное произведение и будет нужным нам предсказанием: $\hat{r}_{ui} = \langle p_u, q_i \rangle$.

Попытаемся придумать модель предсказания, которая будет работать сходным с SVD образом. Модель будет зависеть от многих параметров — векторов пользователей и фильмов. Для заданных параметров возьмем вектор пользователя, вектор фильма, получим их скалярное произведение:

$$\hat{r}_{ui}(\Theta) = \boldsymbol{p}_u^T \boldsymbol{q}_i,$$

$$\Theta = \{\boldsymbol{p}_u, \boldsymbol{q}_i \mid u \in U, i \in I\}$$

Но так как векторов мы не знаем, их еще нужно получить. Идея заключается в том, что у нас есть оценки пользователей, при помощи которых мы можем найти такие оптимальные параметры, при которых наша модель предсказывала бы эти оценки как можно лучше:

$$\mathbf{E}_{(u,i)} \big(\hat{r}_{ui}(\Theta) - r_{ui}\big)^2 \to \min_{\Theta} .$$

Итак, мы хотим найти такие параметры θ, чтобы квадрат ошибки был как можно меньше. Но тут есть парадокс: мы хотим меньше ошибаться в будущем, но мы не знаем, какие оценки у нас будут спрашивать. Соответственно и оптимизировать это мы не можем. Но нам известны уже проставленные пользователями оценки. Попробуем подобрать параметры так, чтобы на тех оценках, которые у нас уже есть, ошибка была как можно меньше. Кроме того, добавим еще одно слагаемое — регуляризатор.

$$\underbrace{\sum_{(u,i)\in\mathcal{D}} \left(\hat{r}_{ui}(\Theta) - r_{ui}\right)^2}_{\text{качество на обучающей выборке}} + \underbrace{\lambda \sum_{\theta\in\Theta} \theta^2}_{\text{регуляризация}} \to \min_{\Theta}$$

Чтобы оптимизировать параметры используем следующий функционал:

$$J(\Theta) = \sum_{(u,i)\in\mathcal{D}} \left(\boldsymbol{p}_u^T \boldsymbol{q}_i - r_{ui}\right)^2 + \lambda\left(\sum_u \|\boldsymbol{p}_u\|^2 + \sum_i \|\boldsymbol{q}_i\|^2\right)$$

Параметров много: для каждого пользователя, для каждого объекта у нас есть свой вектор, который мы хотим оптимизировать. У нас есть функция, зависящая от большого количества переменных. Чтобы найти ее минимум, мы можем воспользоваться методом Чередующихся Наименьших Квадратов (ALS), отраженном на рис. 2.

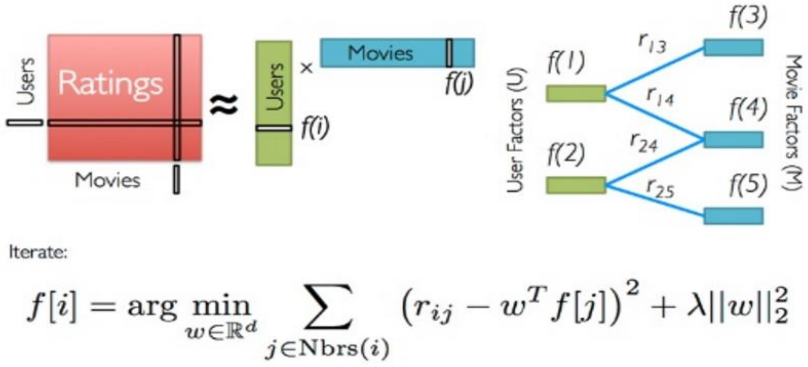

Рис. 2. Метод Чередующихся Наименьших Квадратов

Например, если нам нужно посчитать минимум для параболы, мы точно знаем, где минимум. Оказывается, что функционал, который мы пытаемся оптимизировать, — сумма квадратов ошибок плюс сумма квадратов всех параметров — это тоже квадратичный функционал, он очень похож на параболу. Для каждого конкретного параметра, если мы зафиксируем все остальные, это будет как раз параболой. Т.е. минимум по одной координате мы можем точно определить. На этом соображении и основан метод ALS. В нем мы попеременно точно находим минимумы то по одним координатам, то по другим:

$$\boldsymbol{p}_u^*(\Theta) = \arg\min_{\boldsymbol{p}_u} J(\Theta) = (\boldsymbol{Q}_u^T \boldsymbol{Q}_u + \lambda \boldsymbol{I})^{-1} \boldsymbol{Q}_u^T \boldsymbol{r}_u,$$

$$\boldsymbol{q}_i^*(\Theta) = \arg\min_{\boldsymbol{q}_i} J(\Theta) = (\boldsymbol{P}_i^T \boldsymbol{P}_i + \lambda \boldsymbol{I})^{-1} \boldsymbol{P}_i^T \boldsymbol{r}_i.$$

Мы фиксируем все параметры объектов, оптимизируем точно параметры пользователей, дальше фиксируем параметры пользователей и оптимизируем параметры объектов и действуем итеративно:

$$\forall u \in U \quad \boldsymbol{p}_u^{2t+1} = \boldsymbol{p}_u^*(\Theta_{2t}),$$

$$\forall i \in I \quad \boldsymbol{q}_i^{2t+2} = \boldsymbol{q}_i^*(\Theta_{2t+1}).$$

Описание архитектуры системы Apache Spark.

Apache Spark – это высокопроизводительное средство обработки данных, призванное заменить или дополнить технологию MapReduce путем обобщения и модификации использованных для ее функционирования технологий. Spark дает повышение скорости обработки данных (до 100 раз по сравнению с MapReduce при условии работы в оперативной памяти и до 10 раз при условии взаимодействия системы с жестким диском) посредством уменьшения количества операций чтения и записи на жесткий диск – теперь существенная часть операций производится в реальном времени. Такая технология обработки информации стала возможной благодаря хранению информации о каждом операторе в оперативной памяти. Все процессы, связанные с данными, происходят на одном и том же кластере данных, в одном и том же приложении.

Фреймворк Apache Spark состоит из четырех функциональных модулей, работающих в пределах одного кластера данных и одновременно распределенных горизонтально по всему кластеру компьютеров, работающих на Hadoop.

Первый модуль носит название SparkSQL – и, сообразно ему, дает возможность интеграции SQL-запросов со всеми без исключения элементами фреймворка. Благодаря представлению информации в виде так называемых упругих распределенных наборов данных (RDD), осуществляется быстрый и простой доступ к данным через прикладные интерфейсы на Python, Scala и Java. Кроме того, такая технология позволяет запрашивать данные и одновременно запускать сложные алгоритмы их анализа. Модель архитектуры Spark отражена на рис. 3.

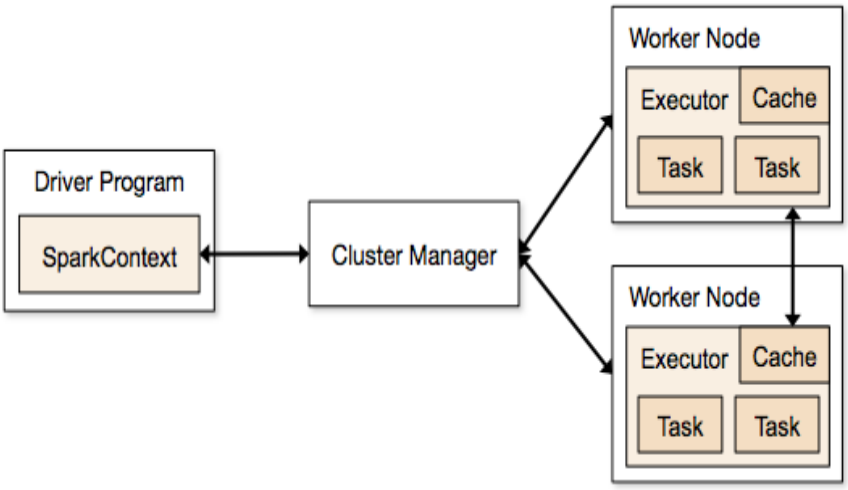

Рис. 3. Схема работы Spark

Второй элемент ApacheSpark – SparkStreaming, компонент, позволяющий пользователю писать и запускать приложения на Scala и Java в потоковом режиме. При этом, приложение сможет одновременно работать как с потоками данных, так и осуществлять пакетную обработку без существенных изменений в коде. Плюс ко всему, фреймворк способен автоматически восстанавливать данные после ошибочных действий со стороны системы – от пользователя в этом случае не потребуется написать ни строчки кода.

Следующие два компонента призваны решать вполне четкие задачи. Первый из них, библиотека машинного обучения MLib, обладает отличной способностью интеграции: ее можно подключать к прикладным интерфейсам на все той же троице языков – Java, Python и Scala. Еще два ключевых момента заключаются в скорости тестирования и обучения (все те же 100x по сравнению с MapReduce) и простотой развертки: библиотека работает на уже существующих компьютерах кластера Hadoop и с уже существующими данными. Некоторые из этих алгоритмов работают и с потоковыми данными, например, линейная регрессия с использованием обычного метода наименьших квадратов или кластеризация по методу k-средних (список вскоре расширится).

Последний, пятый, компонент фреймворка – прикладной интерфейс GraphX для работы с графами и графо-параллельных вычислений. Модуль обладает такими важными характеристиками как гибкость, то есть способность «бесшовной» работы как с графами, так и с коллекциями данных и скорость работы, приложение показывает наиболее высокие результаты в своем классе.

Постановка задачи исследования

Целью исследования является определение величины параметра Rank для получения рекомендаций оптимального качества. За меру качества рекомендаций возьмем величину RMSE (Root Mean Squared Error, Корень из средней квадратичной ошибки). Сравнив RMSE для разных значений Rank и найдя наименьший, мы получим самые качественно настроенные рекомендации. Данный показатель вычисляется следующим образом:

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n}}$$

Описание работы.

Файлы с данными загружают в RDD системы Apache Spark. Каждый из двух загружаемых файлов представляет собой набор колонок, разделенных запятыми. Набор данных состоит из 10 329 записей в файле movies.csv (movieId, title, genres) и 105 339 записей в файле ratings.csv (userId, movieId, rating)

Так как структура файлов достаточно понятна и проста, может использоваться метод split(), являющийся родным методом для языка Python с целью парсинга этих файлов.

Для дальнейшей работы DataSet разбивается на три отдельные части - на тренировочную, валидационную и тестовую. Деление производится в формате 60%-20%-20%. Большая доля разбиения, как правило, приходится на тренировочную часть для того, чтобы увеличить точность алгоритма.

После выделения отдельных частей (трех новых RDD) модель должна быть тренирована и выбран предпочтительный параметр Rank для построения рекомендаций. Это будет определяться по значению ошибки - Rank с минимальным значением этого показателя будет считаться предпочтительнее. После проведенных вычислений, результаты отразим в табл.1:

Таблица 1. Результаты показателя RMSE для различных Rank

Rank	RMSE
1	0,8984
2	0,9006
3	0,9159
4	0,9144
5	0,9150
6	0,9211
7	0,9255
8	0,9254
9	0,9276
10	0,9260

Выводы по результатам моделирования

Исследование показывает, что показатель ошибки RMSE растет при увеличении количества факторов в модели (показатель Rank), причем наибольшие пики роста наблюдаются при рассмотрении 3 и 6 факторов соответственно.

Заключение

В данной работе была изучена предметная область, связанная с вопросом персональных рекомендаций, были изучены вопросы, связанные с проектированием необходимого решения с использованием Apache Spark, произведено исследование по определению оптимального показателя Rank для построения модели.

Список литературы

[1]. Sarwar B., Karypis G., Konstan J., Reidl J. Item-based collaborative filtering recommendation algorithms // Proceedings of the 10th international conference on World Wide Web (WWW10). (May 1-5. 2001. Hong Kong). ACM. 2001. P. 285–295.

[2]. Takács G., Pilászy I., Németh B., Tikk D. Matrix factorization and neighbor based algorithms for the Netflix prize problem. // RecSys '08. Proceedings of the 2008 ACM Conference on Recommender Systems. (Lausanne, Switzerland, October 23-25). / New York, NY, USA: ACM. 2008. P. 267-274. DOI: 10.1145/1454008.1454049

[3]. Konstas I., Stathopoulos V., Jose J.M. On social networks and collaborative recommendation // SIGIR '09: Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval. (19-23 July 2009, New York, USA) / New York, NY, USA: ACM. 2009. P. 195–202. DOI: 10.1145/1571941.1571977

[4]. Cantador I., Konstas I., Jose J. Categorising social tags to improve folksonomy-based recommendations // Web Semantics: Science, Services and Agents on the World Wide Web. 2011. Vol. 9. Is. 1. P. 1–15.

[5]. Groh G., Ehmig C. Recommendations in taste related domains: collaborative filtering vs. social filtering // GROUP'07: Proceedings of the 2007 international ACM conference on Supporting group work. (November 4th-7th, 2007. Sanibel Island, Florida, United States). / New York, NY, USA: ACM. 2007. P. 127–136. DOI: 10.1145/1316624.1316643

[6]. Slaney M., Casey M. Locality-sensitive hashing for finding nearest neighbors [lecture notes] // IEEE. Signal Processing Magazine. 2008. Vol. 25. Is. 2. P. 128–131. DOI: 10.1109/MSP.2007.914237

[7]. Dean J., Ghemawat S. Mapreduce: simplified data processing on large clusters // Communications of the ACM - 50th anniversary issue: 1958 - 2008. New York, NY, USA: ACM. 2008. Vol. 51. Is. 1. P. 107– 113. DOI: 10.1145/1327452.1327492

[8]. Castelluccio M. The music genome project. // Strategic Finance. 2006. Vol. 88. №. 6. P. 57–58.

[9]. Lemire D., Maclachlan A. Slope one predictors for online rating-based collaborative filtering. / SIAM Data Mining (SDM'05). (Newport Beach, California, April 21-23, 2005). // Society for Industrial Mathematics. 2005. Vol. 5. P. 471–480.

[10]. Herlocker J.L., Konstan J.A., Terveen L.G., Riedl J.T. Evaluating collaborative filtering recommender systems // ACM Transactions on Information Systems (TOIS). 2004. Vol. 22. № 1. P. 5–53.