Лабораторная работа №5

Разработать приложение под Apache Spark, предсказывающее рейтинги фильмов. В качестве набора данных использовать данные MovieLens (файл ml-latest-small.zip, 100 000 оценок, 9 000 фильмов, 700 пользователей).

На экран вывести:

- **Количество оценок и фильмов в наборе данных**
- **Примеры оценок и примеры фильмов из набора данных**
- **Топ 50 фильмов с наивысшим средним рейтингом (среди фильмов для которых есть более 100 оценок от пользователей)**
- **Параметры обученных моделей и значения ошибок RMSE (выборки для валидации и для тестирования)**
- **Оценки для собственного списка просмотренных фильмов**
- **Рекомендации (Топ20) по фильмам (фильмы с наивысшим рейтингом) для самого себя**

Введение

Предсказание того, что хочет пользователь является одним из направлений в Big Data и Machine Learning. Данное направление довольно новое и получило название «Рекомендательные системы». Алгоритмы, работающие в этой области, помогают компании Google показывать релевантную рекламу, Amazon`y – показывать нужные пользователям товары, Netflix`y – рекомендовать пользователям фильмы, которые им должны понравиться.

В этой лабораторной работе Вашей задачей является используя базовые техники и возможности Apache Spark и MLlib разработать систему, которая смогла бы рекомендовать пользователям фильмы.

Комментарии и рекомендации по выполнению работы:

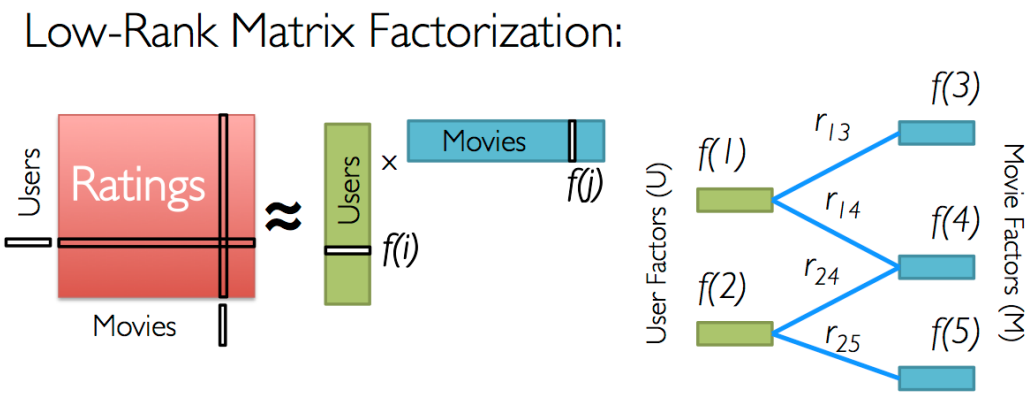
1. **Часть 1. Базовые рекомендации**
    a. Формат входных данных можно посмотреть в заголовках csv файлов из предложенного набора данных.
    b. Для представления данных удобнее всего использовать кортежи.
    c. Простейшая модель рекомендаций может базироваться на средних оценках пользователей, с учетом общего числа оценок (рекомендация должна быть взвешенной, т.е. необходима некоторая граница по количеству оценок, когда мы считаем рекомендацию возможной).
    d. Средние оценки элементарно вычисляются по набору данных после агрегации по идентификатору фильма.
    e. Использование ограничений на количество оценок, необходимых для участия в рейтинге – это лишь один из способов улучшить качество рекомендаций.
2. **Часть 2. Совместная фильтрация**
    a. Совместная фильтрация (Collaborative filtering) - это метод генерации автоматических предсказаний интересов пользователей за счет сбора информации о предпочтениях от большого количества пользователей. В основе совместной фильтрации лежит предположение, что если пользователь A имеет схожие

предпочтения с пользователем B по определенному вопросу, то наиболее вероятно, что A согласится с мнением B по другому вопросу, нежели с пользователем, выбранным случайно. Более подробно можно прочитать здесь.

Вначале пользователи ставят оценки некоторым объектам (например, фильмам, играм и т.д.). После этого система делает предсказание оценок пользователя по тем объектам, которые он еще не оценивал, эти предсказания основываются на имеющихся оценках других пользователей, которые ставили похожие оценки, как рассматриваемый пользователь.

Для рекомендации фильмов стоит начать с матрицы оценок фильмов различными пользователями. Каждый столбец будет представлять пользователя, а каждая строка - фильм.

Поскольку не все пользователи оценили все фильмы, некоторые элементы в матрице (в реальных задачах - большая их часть) останутся неизвестными, вот почему нам требуется совместная фильтрация (collaborative filtering). Идея совместной фильтрации состоит в том, чтобы разложить, рассмотренную выше матрицу, на две матрицы (одна будет описывать свойства каждого пользователя, вторая свойства каждого фильма), произведение которых дадут искомую матрицу.



Iterate:

$$f[i] = \arg\min_{w \in \mathbb{R}^d} \sum_{j \in \mathrm{Nbrs}(i)} \left(r_{ij} - w^T f[j]\right)^2 + \lambda ||w||_2^2$$

Нам необходимо выбрать эти две матрицы таким образом, чтобы ошибка для пар пользователь/фильм с известной оценкой была минимизирована. Алгоритм Чередования наименьших квадратов (Alternating Least Squares - ALS) достигает этого заполняя случайными числами матрицу пользователей и затем оптимизируя значения фильмов таким образом, чтобы ошибка была наименьшей. Затем он фиксирует матрицу фильмов (она больше не меняется) и оптимизирует значения матрицы пользователей. Название алгоритма произошло именно из этого чередования матриц.

Описанный процесс оптимизации показан в правой части картинки, приведенной выше. Задавая фиксированное количество пользовательских коэффициентов (чисел в матрице пользователей), алгоритм использует известные оценки, чтобы

найти наилучшие значения для коэффициентов фильмов (проводимая оптимизация приведена в нижней части картинки). Далее "чередуя" матрицы мы подбираем лучшие пользовательские коэффициенты для заданных коэффициентов из матрицы фильмов.

b. Для обучения модели рекомендуется разбить имеющиеся данные на 3 набора: для обучения (training), для валидации (validation) и для тестирования (testing). Рекомендуется использовать соотношение 6:2:2 для размера этих наборов данных. Разбиение удобнее всего сделать с помощью функции **randomSplit**.
c. В процессе выполнения работы Вам будет необходимо сгенерировать несколько моделей с различными параметрами. Чтобы определить какая модель лучше, удобнее всего использовать среднеквадратичную ошибку (Root Mean Square Error - RMSE) для оценки ошибки каждой модели. RMSE часто используется, как мера отличия между наблюдаемыми и предсказанными значениями, это очень хорошая мера точности модели для сравнения ошибок различных моделей для конкретной переменной/параметра, но не для сравнения между переменными, т.к. RMSE зависит от шкалы.
d. Для расчета RMSE удобнее всего использовать **RegressionMetrics** из Spark MLlib. На вход функция оценки ошибки должна принимать набор предсказанных значений и набор наблюдаемых значений.
e. Библиотека машинного обучения Spark MLlib содержит реализацию алгоритма Alternating Least Squares (ALS). Для обучения модели, кроме обучающей выборки требуется передать алгоритму ряд параметров. **Rank** – это количество строк в матрице пользователей или количество столбцов в матрице фильмов. В общем случае меньшие значения ранга будут давать большую ошибку на обучающей выборке, но большие значения ранга могут привести к переобучению. **Iterations** – количество итераций в обучении алгоритма. **Lambda** – параметр регуляризации. Для данной лабораторной работы рекомендуется использовать следующие значения: rank – 4, 8, и 12 (три разных модели, из которых следует выбрать лучшую, т.е. с меньшей ошибкой RMSE); iterations – 5; lambda – 0.1.
f. Для валидации моделей рекомендуется создать специальный набор данных (из validationRDD), не включающий значения оценок. Далее можно будет сравнить значения, предсказанные моделью, с validationRDD для оценки RMSE модели.
g. Далее следует оценить RMSE модели на тестовой выборке.

3. **Часть 3. Рекомендации для себя**
    a. Чтобы алгоритм выдал рекомендации для Вас, необходимо сформировать собственный список оценок и добавить его в обучающую выборку. Для этого удобнее всего использовать Топ 50 фильмов с наивысшим рейтингом из Части 1 и проставить оценки тем фильмам, которые вы смотрели.
    b. Свои оценки удобнее всего заносить для пользователя с ID = 0, т.к. этот идентификатор не используется в оригинальном наборе данных.
    c. Обучив модель (с наилучшими параметрами из Части 2) на выборке с вашими данными и проверив RMSE, можно сделать предсказания какие фильмы из не просмотренных могли бы Вам понравиться. Для этого необходимо подготовить выборку с идентификаторами не оцененных вами фильмов и идентификатором вашего пользователя (0).
    d. Отсортировав предсказания по рейтингу, вы получите Топ-фильмов, которые могли бы Вам понравиться.