ВМК

Задание 7. Рекомендательная система фильмов на данных MovieLens

Практикум 317 группы, весна 2016

Начало выполнения задания: 14 апреля 2016 года.
Срок сдачи: **5 мая 2016 года, 23:59.**
Среда для выполнения задания: Python 3.4 / 2.7 (при использовании mrjob).
Текст задания последний раз обновлялся 14 апреля 2016 г.

Содержание

1 Рекомендательные системы

Сегодня рекомендательные системы встречаются повсеместно. В интернет-магазине вы можете увидеть блоки с «похожими товарами», на новостном сайте «похожие новости» или «новости, которые могут вас заинтересовать», на сайте с арендой фильмов это могут быть блоки с «похожими фильмами» или «рекомендуем вам посмотреть».

Задача рекомендательной системы заключается в нахождении небольшого числа фильмов (Item), которые скорее всего заинтересуют конкретного пользователя (User), используя информацию о предыдущей его активности и характеристиках фильмов.

Широко известен конкурс компании Netflix, которая в 2006 году предложила предсказать оценки пользователя для фильмов в шкале от 1 до 5 по известной части оценок. Победителем признавалась команда, которая улучшит RMSE на тестовой выборке на 10% по сравнению с их внутренним решением. За время проведения конкурса появилось много новых методов решения поставленной задачи.

Мы рассмотрим два подхода к построению рекомендаций [1]: content-based и collaborative filtering. В задаче коллаборативной фильтрации мы рассмотрим два наиболее популярных подхода: neighborhood и latent factor.

Обычно в таких задачах выборка представляет собой тройки $(u, i, r_{u,i})$, где u – пользователь, i – фильм, $r_{u,i}$ – рейтинг. Далее будем считать, что рейтинги нормализованы на отрезок $[0, 1]$.

[1] Francesco Ricci et al, Recommender Systems Handbook, 2011

1.1 Content-based подход

В таком подходе рекомендательная система пытается найти фильмы на основе: характеристик фильмов (например, жанр, режиссер, год выхода), профиля каждого пользователя в терминах характеристик фильмов, характеристик пользователей (например, пол, профессия).

Для каждой пары u, i необходимо придумать признаки $f^n_{u,i}$, основанные на профиле пользователя, собранном на обучении, и характеристиках пользователей и фильмов, известных даже для новых пользователей и фильмов.

Следующий набор признаков можно использовать для рекомендательной системы:

- $f^1_{u,i}$ – категориальный признак, возраст пользователя
- $f^2_{u,i}$ – категориальный признак, профессия пользователя
- $f^3_{u,i}$ – набор булевых признаков, по одному на каждый жанр, к которому отнесен фильм
- $f^4_{u,i}$ – категориальный признак, пол пользователя
- $f^5_{u,i}$ – $(u_g \cdot m_g)/n_g$, где u_g – вектор средних оценок пользователя в пространстве жанров, m_g – булевый вектор для фильма в пространстве жанров, n_g – количество жанров, указанных для фильма
- $f^6_{u,i}$ – средний рейтинг пользователя
- $f^7_{u,i}$ – средний рейтинг фильма
- $f^8_{u,i}$ – константный признак

Категориальные признаки необходимо закодировать набором булевых векторов, по одному на каждое значение признака. Полученные признаки обозначим как $\{g^n_{u,i}\}_{n=1..N}$.

Далее предлагается искать рейтинг как линейную комбинацию числовых признаков:

$$\hat{r}_{u,i} = \sum_{n=1}^{N} g^n_{u,i}\theta_n \tag{1}$$

Для настройки весов можно, например, воспользоваться Ridge-регрессией. В качестве структурного параметра на этапе тестирования алгоритма рекомендуется выбрать $\lambda = 0.2$. Предложенное значение структурного параметра не является оптимальным, находить оптимальное значение необходимо кросс-валидацией.

1.2 Ridge-регрессия

В этом методе настройки линейной регрессии минимизируется следующий функционал:

$$||Xw - y||^2 + \lambda||w||^2.$$

Решением является:

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y.$$

Обратим внимание, что решение можно найти без непосредственного обращения матрицы. Нужно воспользоваться методом решения СЛАУ.

1.3 Neighborhood подход в коллаборативной фильтрации

Имея матрицу user-item из оценок пользователей можно определить меру adjusted cosine similarity похожести товаров i и j как векторов в пространстве пользователей:

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

где U – множество пользователей, которые оценили фильмы i и j, $\overline{r_u}$ – средний рейтинг пользователя u.

Рейтинги для неизвестных фильмов считаются по следующей формуле:

$$\hat{r}_{u,i} = \sum_{j:\ r_{u,j}\neq 0} sim(i,j) r_{u,j} \Big/ \sum_{j:\ r_{u,j}\neq 0} sim(i,j) \tag{3}$$

Такой подход называется item-oriented. Обратим внимание на то, что $sim(i,j) \in [-1,1]$. Это может привести к делению на ноль или значениям $\hat{r}_{u,i}$ вне диапазона $[0,1]$. Избавиться от этой проблемы можно, например, положив равными нулю отрицательные значения $sim(i,j)$.

Имеет право на существование и симметричный user-oriented подход,

Выбор между user- и item-oriented подходом зависит от размерности матрицы и от среднего кол-ва оценок на пользователя/фильм.

1.4 Latent factor подход в коллаборативной фильтрации

В этом подходе оценка r_{ui} пользователя u, поставленная фильму i, ищется как скалярное произведение векторов p_u и q_i в некотором пространстве $\mathbb{R}^K$ латентных признаков:

$$\hat{r}_{ui} = p_u^T q_i \tag{4}$$

Иными словами, модель находит пространство признаков, в котором мы описываем и фильмы и пользователей и в котором рейтинг является мерой близости между фильмами и пользователями.

Для настройки модели будем минимизировать следующий функционал:

$$\sum_{(u,i,r_{ui})} (r_{ui} - p_u^T q_i)^2 + \lambda_p p_u^T p_u + \lambda_q q_i^T q_i, \tag{5}$$

где суммирование ведется по всем тройкам (u, i, r_{ui}) выборки, слагаемые с λ_p и λ_q добавлены для регуляризации.

В статье [2] описан метод оптимизации ALS (Alternating Least Squares) для функционала (5).

В методе проводятся N итераций, в рамках каждой итерации сначала оптимизируется p при фиксированном q, затем q при фиксированном p.

Составим матрицу P из векторов p_u и матрицу Q из векторов q_i. Матрицей $Q[u] \in \mathbb{R}^{n_u \times K}$ будем обозначать подматрицу матрицы Q только для товаров, оцененных пользователем u, где n_u – количество оценок пользователя u.

Шаг перенастройки p_u при фиксированной матрице Q сводится к настройке ridge-регрессии и выглядит так:

$$A_u = Q[u]^T Q[u] \tag{6}$$

$$d_u = Q[u]^T r_u \tag{7}$$

$$p_u = (\lambda_p n_u I + A_u)^{-1} d_u \tag{8}$$

Формулы для перенастройки q_i при фиксированной матрице P выглядят аналогично.

Для тестирования реализации предлагается использовать $\lambda_p = 0.2$, $\lambda_q = 0.001$, $N = 20$, $K = 10$, $Q = 0.1$ * np.random.random(...), $P = 0.1$ * np.random.random(...). Предлагаемые значения структурных параметров не являются оптимальными, их необходимо находить кросс-валидацией.

1.5 Сравнение методов

Neighborhood и latent factor подходы на практике показывают лучшие результаты по сравнению с content-based подходом, так как не используют специфичные для задачи данные (например, описание фильмов жанрами), а пытаются найти более тонкие закономерности в пользовательских предпочтениях. С другой стороны, neighborhood и latent factor подходы страдают от проблемы холодного старта: они не могут выдать рекомендации для новых пользователей или фильмов.

2 Задание

Первая задача состоит в том, чтобы реализовать и сравнить три варианта рекомендательной системы фильмов на данных MovieLens.

Необходимо сравнить скорость работы и качество получаемых рекомендаций в метрике MSE. Вторая задача состоит в получении опыта написания распределенных приложений с помощью пакета mrjob и кластера Amazon.

2.1 Content-based

В этой задаче вам необходимо реализовать описанные в разделе 1.1 признаки $\{f^n_{u,i}\}_{n=1..8}$. Заметим, что описанные признаки не используют все доступные характеристики пользователей и фильмов. Вам предлагается придумать и добавить в модель 2 дополнительных признака. Как можно использовать названия фильмов?

2.2 Neighbourhood based коллаборативная фильтрация

Предлагается реализовать описанный в 1.3 item-based подход. Для user-based необходимо только привести формулы. В neighborhood подходе необходимо исследовать качество и время работы в зависимости от длины списка похожих товаров: для каждого товара можно хранить только первые N самых похожих на него по мере $sim(i, j)$, что уменьшает требования к памяти и ускоряет работу алгоритма. Необходимо предоставить таблицу, в которой для разумных значений N отражено качество на обучении и на контроле, а также время работы

[2] Istvan Pilaszy, Fast ALS-based Matrix Factorization for Explicit and Implicit Feedback Datasets

алгоритма. Необходимо сделать выводы по таблице. Программу необходимо писать в парадигме map-reduce с использованием пакета mrjob.

Перед запуском на кластере программу необходимо протестировать в нераспределенном режиме и убедиться в ее правильности и работоспособности.

2.3 Latent factor based коллаборативная фильтрация

В latent factor подходе необходимо исследовать качество и время работы в зависимости от размерности K пространства латентных признаков. Ведет ли увеличение K к переобучению? Необходимо предоставить таблицу, где для каждого разумного значения K отражено качество на обучении и на контроле, а также время работы. Необходимо сделать выводы по таблице.

Также необходимо выписать формулы для перенастройки q_i при фиксированной матрице P.

2.4 Знакомство с Amazon AWS. Первый запуск map-reduce приложения

Нам понадобится свой кластер, на котором мы могли бы ставить эксперименты, учиться пользоваться технологией hadoop и в дальнейшем, если нам понравится, использовать в своей работе. Можно, конечно, затребовать компьютерный класс и поднять на нем распределенную файловую систему. Неудобство этой реализации в том, что пространственно-временная доступность кластера будет весьма ограничена. Можно, конечно, настроить доступ к кластеру с помощью web-интерфейса. Это позволит нам не только работать в лаборатории, но и раздавать или продавать наш распределенный ресурс:). Однако действия в этом направлении могут натолкнуться на противодействие со стороны университета. Кроме того, фокус тема нашего курса "Восстановление зависимостей в больших массивах данных а вовсе не распределенные файловые системы и hadoop является лишь инструментом, а не целью. Но раз идея поднятия кластера пришла нам голову, то она, скорее всего, уже воплотилась в виде готовых сервисов и мы воспользуемся одним из них: **Amazon Web Service** компании Amazon.

Amazon Web Service — вебсервис, который позволят обрабатывать огромные количества разнообразных данных. Сервис основан на совместном использовании EC2 и S3, а также фреймворка Hadoop.

По заверениям Amazon, используя Elastic MapReduce вы с легкостью сможете:

- Разрабатывать приложения для обработки большого массива данных на любом удобном для вас языке: Java, Ruby, Perl, Python, PHP, R, or C++.
- Загружать данные и приложения по их обработке в Amazon S3. Надежность, масштабируемость, легкость в использовании — это все он, Amazon S3.
- Стартовать через AWS Management Console так называемый MapReduce «job flow». Вы просто напросто выбираете нужный инстанс Amazon EC2, далее выбираете путь к данным и приложению по их обработке, которые находятся на Amazon S3, нажимаете кнопку «Создать Job Flow» и MapReduce начнет свою работу.
- Мониторить статус job flow посредством AWS Management Console, коммандной строки или же специального API. После окончания работы результат помещается в Amazon S3.

Для того, чтобы воспользоваться преимуществами Amazon AWS, надо выполнить следующий алгоритм действий.

- **Шаг 1.** Регистрируемся на сайте http://aws.amazon.com/ru/.
- **Шаг 2.** Запускаем консоль управления AWS (рис. 1).

  Из этого большого мешка нам понадобятся 2 инструмента: **S3 (Scalable Storage in Cloud)** из раздела Storage and Content Delivery и **Elastic Map Reduce (EMR)** из раздела **Analytics**.
- **Шаг 3.** Создаем место для хранения программ и данных. Хранилище S3 нам нужно для хранения данных и результатов и на этом складе нам надо отгородить себе уголок. Выбираем пункт S3 и жмем на кнопочку “Create Bucket”. Важным моментом здесь является то, что имя Вашего уголка должно быть уникальным в пределах всего склада, а поэтому простые названия вроде "bucket"уже, скорее всего, заняты и придется быть более креативным.
- **Шаг 4.** Запустим тестовый пример, который выполняет распределенный подсчет количества различных слов в тексте. Для тестового приложения используется готовый текст, расположенный где-то в репозитории S3. Загружать его не надо. Выбираем пункт **Elastic Map Reduce (EMR)** из раздела **Analytics** и далее последовательно выбираем **Create Cluster**, **Go to advanced options** и **Configure sample application** (рис.2). В появившемся окне выбираем приложение "Word count определяем место в нашем отгороженном уголке мегапространства S3, где будет храниться результат разбора текста. Нажимаем кнопочку "Create cluster". Далее всю работу берет на себя EMR Amazon, и где-то через полчаса можно будет увидеть результат.

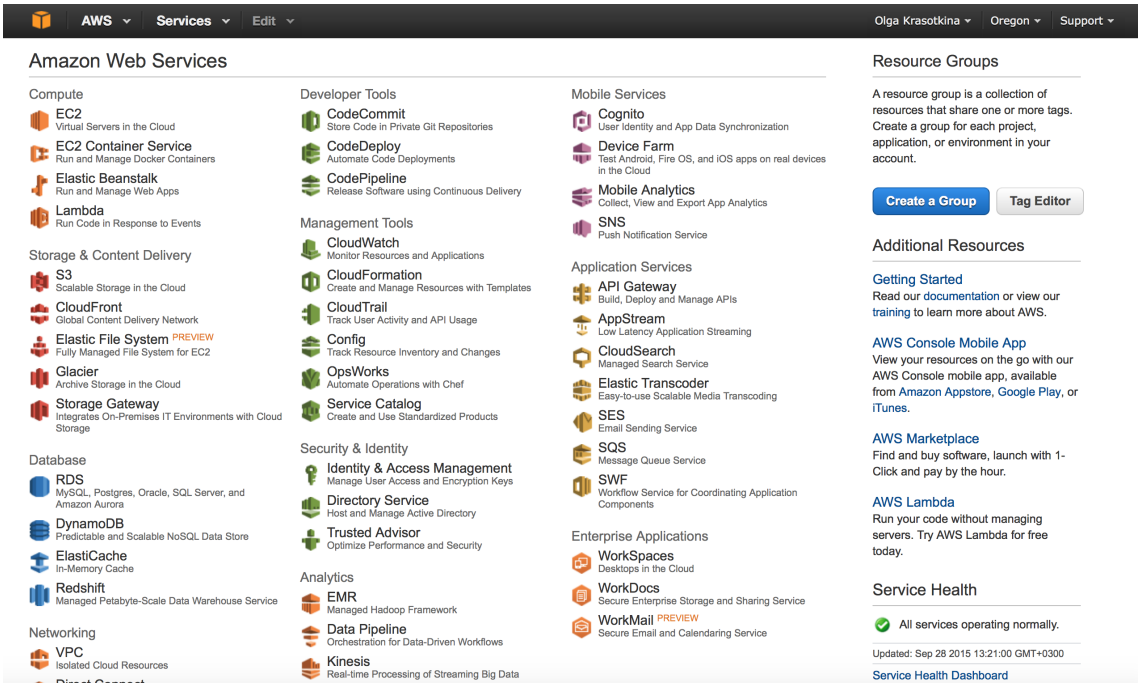

Рис. 1: Консоль управления AWS

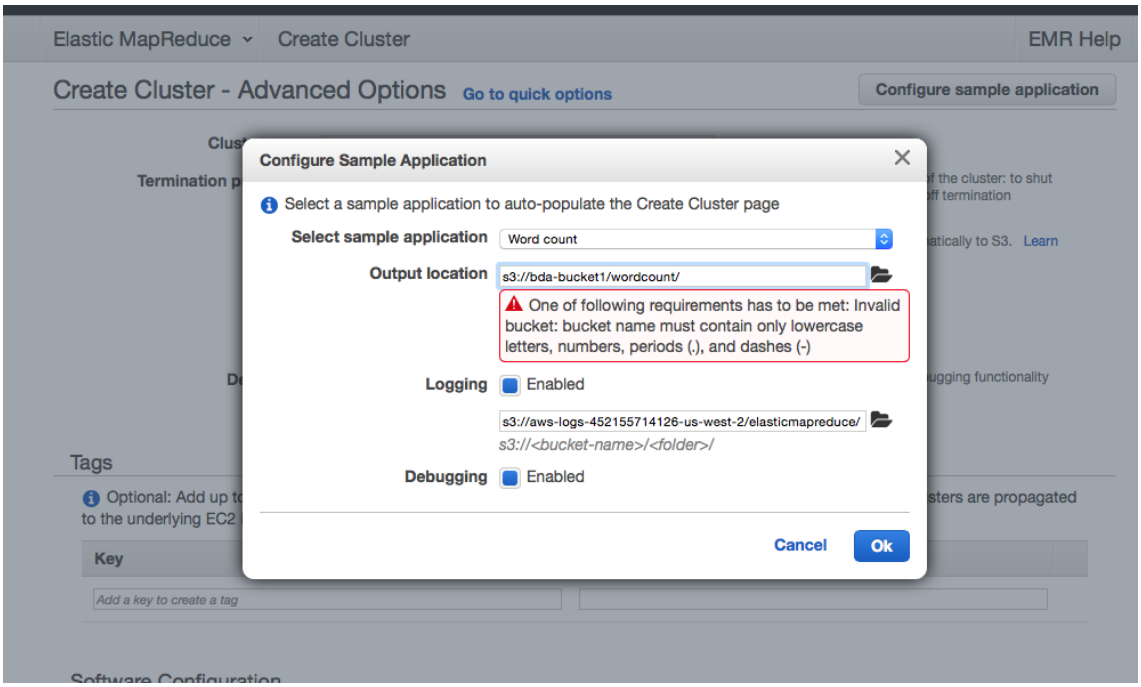

Рис. 2: Создание тестового приложения MapReduce

2.5 Настройка и использование пакета mrjob

Нам необходимо простое в использовании средство, которое позволит тестировать и отлаживать наши алгоритмы восстановления зависимостей локально, загружать их и данные в кластер Amazon и запускать распределенные вычисления.

С учетом того, что большинство исследовательских проектов пишется на Python, для обозначенных выше целей нам больше всего подойдет библиотека mrjob http://mrjob.readthedocs.org/en/latest/index.html. Библиотека mrjob представляет собой обертку на Python для стандартных функций Hadoop Streaming.

Mrjob это простейший способ писать программы на Python, которые запускаются на распределенной системе Hadoop. Используя mrjob, вы получаете возможность тестировать ваши программы локально без инсталяции hadoop и создания кластера. Кроме того, mrjob глубоко интегрирована с Amazon Elastic MapReduce. Это позволяет вам запускать ваши программы в облаке так же легко, как вы запускаете их на локальной машине.

Достоинства mrjob:

- Библиотека mrjob документирована лучше, чем любой другой фрейморк, выполняющий те же функции.
- Библиотека mrjob, отличие от других фреймворков, позволяет тестировать ваши приложения без разворачивания hadoop
- интерфейс запуска вашего приложения остается одним и тем же вне зависимости от того, выполняется приложение локально или в облаке
- не требует написания специальных скриптов для загрузки данных и программ в кластер
- Лучше других приложение интегрирована с Amazon EMR
- позволяет отлаживать ваши программы локально с использованием вашей любимой среды разработки, а не разбирать лог файлы запуска программы в кластере
- нет необходимости в сериализации данных, библиотека mrjob все сделает за вас

Недостатки mrjob:

- Библиотека mrjob не даст вам того уровня доступа к API Hadoop как предоставляют библиотеки Dumbo или Pydoop. Но для пользовательских задач такого уровня доступа и не нужно.
- Согласно результатам тестов приложения, написанные с помощью mrjob работают несколько медленнее, чем написанные с помощью dumbo, hadoop streaming и pydoop, но это время с лихвой окупается временем, затраченным на подготовку задачи для решения на кластере

Будем исходить из предположения, что Python версии 2.5, 2.6 или 2.7 уже стоит. Для установки mrjob выполните команду

pip install mrjob

или клонируйте git репозиторий mrjob и запустите команду

python setup.py install.

Теперь попробуем скомпилировать тестовый пример и убедимся, что все установилось корректно и работает.

В качестве тестового примера рассмотрим программу подсчета статистик в некотором тексте:

```
from mrjob.job import MRJob

class MRWordFrequencyCount(MRJob):

    def mapper(self, _, line):
        yield "chars", len(line)
        yield "words", len(line.split())
        yield "lines", 1

    def reducer(self, key, values):
        yield key, sum(values)

if __name__ == '__main__':
    MRWordFrequencyCount.run()
```

Программа представляет собой пользовательский класс, отнаследованный от класса MRJob, с переопределением функций этого класса в соответствии с реализуемым алгоритмом. В рассмотренном примере класс называется *MRWordFrequencyCount* и осуществляется переопределение функций *mapper* и *reducer*.

Функция *mapper* применяется построчно к анализируемому тексту, производя для каждой строки три пары вида (key, value): (”chars”, numchars), (”words”, numwords), (”lines”, 1). Функция *reducer* просто выполняет сложение значений для каждого ключа.

Для начала протестируем наше приложение локально. В ходе теста нам понадобится один или несколько текстовых файлов. Пусть один из них называется my_file.txt.

Чтобы запустить распределенное приложение , надо создать объект класса *MRWordFrequencyCount* и вызвать его функцию *run*(), передав в параметрах список текстовых файлов.

Реализовать запуск приложения можно двумя способами: в командной строке и с помощью скрипта на Python.

При запуске распределенного приложения в командной строке в виде

```
python word_count.py my_file.txt
```

вывод результата будет осуществляться непосредственно в консоль. Когда вывод результата в консоль не удобен, можно перенаправить выходной поток в файл вот так

```
python word_count.py <my_file.txt> out.txt
```

Встраивание процедуры вызова распределенной обработки в код можно осуществить следующим образом:

```
rom word_count import MRWordFrequencyCount
mr_job = MRWordFrequencyCount(args=["my_file.txt"])
with mr_job.make_runner() as runner:
    runner.run()
    for line in runner.stream_output():
        key, value = mr_job.parse_output_line(line)
        print key, value
```

Осуществим запуск нашей программы на кластере Amazon. Для этого нам необходимо указать свои параметры доступа к кластеру в конфигурационном файле mrjob.conf, который необходимо положить в домашнюю директорию пользователя. Минимальное содержимое конфигурационного файла для запуска задачи на кластере следующее

```
runners:
  emr:
    aws_access_key_id: XXXXXXXXXXXXXXX
    aws_secret_access_key: XXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Если исходные данные находятся на локальном компьютере, то обработка их на кластере будет выглядеть так

```
python word_count.py -r emr <my_file.txt> out.txt
```

Можно загрузить данные в выделенный нами уголок пользователя в пространстве Amazon S3 с помощью консоли управления Amazon Management S3.

В этом случае обработка загруженных в хранилище S3 данных будет осуществляться следующим образом

```
python word_count.py -r emr  <s3://bda-bucket1/lab1/my_file.txt>  out.txt
```

2.6 Реализация задания

На вход программе подается файл из записей вида (user, item, score).

На выходе ожидается файл с готовыми рекомендациями из записей (user, [(item, score), ...]).

Для оценки масштабируемости полученного метода используете выборки разного размера

3 Данные

Данные MovieLens 1M (http://files.grouplens.org/datasets/movielens/ml-1m.zip) представляют собой 1 миллион оценок от 6000 пользователей для 4000 фильмов, а также дополнительную информацию о характеристиках фильмов и пользователей.

В архиве 4 файла:

- `README` (Описание набора данных),
- `ratings.dat` (1000209 рейтингов вида UserID::MovieID::Rating::Timestamp),
- `movies.dat` (характеристики 3900 фильмов вида MovieID::Title::Genres),
- `users.dat` (характеристики 6040 пользователей вида UserID::Gender::Age::Occupation::Zip-code).

3.1 Обучение и контроль

Пусть для каждого пользователя его оценки отсортированы по дате выставления. Предлагается взять в обучающую выборку первые 80% оценок, а оставшиеся 20% использовать в качестве контрольной выборки.

Можно использовать следующий фрагмент кода для разделения выборки:

```
import math
train_frac = 0.8
train = []
test = []
for u, itemList in ratings.items():
    # itemList = [(i, r, t), ...]
    all = sorted(itemList, key=lambda x: x[2])
    thr = int(math.floor(len(all) * train_frac))
    train.extend(map(lambda x: (u, x[0], x[1] / 5.0), all[:thr]))
    test.extend(map(lambda x: (u, x[0], x[1] / 5.0), all[thr:]))
print("ratings in train:", len(train))
print("ratings in test:", len(test))
```

4 Требования к реализации

При работе разрешается использовать сторонние пакеты `numpy`, `scipy`, `matplotlib`. Постарайтесь уделить особое внимание оптимизации кода, используйте векторизацию и матричные вычисления, где это возможно. Все входные данные необходимо считывать из одного zip-архива, не распаковывая его в файловой системе.

Для сдачи задания необходимо предоставить отчет в IPython notebook с кодом для воспроизведения всех результатов.

5 История изменений