ВМК

Задание 7. Рекомендательная система фильмов на данных MovieLens

Практикум 317 группы, весна 2015

Начало выполнения задания: 3 апреля 2015 года.
Срок сдачи: **16 апреля 2015 года, 23:59.**
Среда для выполнения задания: Python 3.4 (желательно) / 2.7 (при выполнении бонусной части).
Текст задания последний раз обновлялся 10 апреля 2015 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 предлагается воспользоваться Ridge-регрессией. Для проверки реализации предлагается воспользоваться $\lambda = 0.2$. Предложенное значение гипер-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, формулы для него предлагается вывести самостоятельно.

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 гипер-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

Ваша 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.

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 выписать 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 алгоритма. Необходимо сделать выводы по таблице.

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

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 Бонус

В бонусной части предлагается реализовать алгоритм коллаборативной фильтрации (item-based neighborhood) на платформе кластерных вычислений Apache Spark. Также предлагается настроить и проверить на тестовых данных готовый алгоритм ALS рекомендаций из Apache Spark MLlib.

2.5 Установка Apache Spark

Предварительно необходимо установить Oracle JDK 7 (http://www.oracle.com/technetwork/java/javase/downloads/jdk7-downloads-1880260.html), python 2.7 (https://www.python.org/downloads, после установки под Windows добавляем `C:\Python27;C:\Python27\Scripts` в Path). Далее скачиваем собранный Apache Spark 1.3.0 (http://d3kbcqa49mib13.cloudfront.net/spark-1.3.0-bin-hadoop2.4.tgz). Так как Spark работает на JVM (Java Virtual Machine), то скачанные бинарные файлы должны работать на всех системах. Распаковываем архив и запускаем «./bin/pyspark» (в Linux, Mac OS X) или «cmd.exe /c bin/pyspark.cmd» (в Windows, запускать от имени Администратора). После запуска мы попадаем в интерпретатор Python, где уже создан объект sc (контекст Spark, через который осуществляется работа). Можем выполнить код для тестирования работы:

```
sc.parallelize(range(1000000)).flatMap(lambda x: [x] * 1000).count()
```

В ответ мы должны увидеть «1000000000» в консоли.

Веб-интерфейс с прогрессом выполнения задач на Spark доступен по адресу: http://localhost:4040.

Если Вам не повезло и команда не работает, то читаем что-нибудь из следующего: http://genomegeek.blogspot.ru/2014/11/how-to-install-apache-spark-on-mac-os-x.html, http://blog.prabeeshk.com/blog/2014/10/31/install-apache-spark-on-ubuntu-14-dot-04/, https://docs.sigmoidanalytics.com/index.php/How_to_build_SPARK_on_Windows.

2.6 Запуск Apache Spark в IPython notebook

Устанавливаем ipython notebook:

```
Unix (в Bash):
pip2.7 install ipython[notebook]
Windows (в cmd.exe):
pip2.7.exe install ipython[notebook] pyreadline
```

Для запуска используем команду (запускаем мастер процесс в ipython notebook, с использованием 4 ядер):

```
Unix (в Bash):
IPYTHON=1 IPYTHON_OPTS="notebook" ./bin/pyspark --master local[4]
Windows (в cmd.exe):
set IPYTHON=1
set IPYTHON_OPTS="notebook"
cmd.exe /c bin/pyspark.cmd --master local[4]
```

2.7 Apache Spark Demo

Справка по python API с примерами использования: http://spark.apache.org/docs/latest/api/python/pyspark.html. Демо с семинара по Apache Spark доступно по адресу: http://nbviewer.ipython.org/urls/dl.dropbox.com/s/ridlf40shgbuvvt/Apache%20Spark%20Demo.ipynb.json?dl=0.

2.8 Реализация задания

На вход ожидается RDD train/test из записей вида (user, item, score). RDD можно получить из списков питона при помощи sc.parallelize.

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

Необходимо так же посчитать качество на контроле (test) при помощи Apache Spark.

Полезно для решения задачи получить следующие RDD из записей:

- (item, [(user, score), ...])
- (user, [(item, score), ...])
- ((item1, item2), [(score1 * score2), ...])

Не забывайте про наличие операций join, groupByKey. Для оценки масштабируемости полученного метода предлагается воспользоваться датасетом в 10 раз больше: http://files.grouplens.org/datasets/movielens/ml-10m.zip. Оцените скорость работы от количества ядер/размера данных. Сравните со скоростью однопоточной реализации в Python.

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. Сдача задания осуществляется через SVN.

5 История изменений

10 апреля

1. Уточнены требования к Neighbourhood based подходу.
2. Добавлено описание бонусной части.