ТЕХНОСФЕРА

Методы распределенной обработки больших объемов данных в Hadoop

Лекция 10: Apache Mahout

Москва 2014

Apache Mahout in Action

- Цель проекта - получить масштабируемую библиотеку машинного обучения:
  - Масштабируемость больших объемов данных
  - Масштабируемость бизнес задач
  - Масштабируемость сообщества
- Режимы работы:
  - В кластере Hadoop
  - Автономно, на одном компьютере
- Открытый исходный код на Java, со свободной для коммерческого использования лицензией Apache Software
- Страница проекта: http://mahout.apache.org

Реализованные алгоритмы

- Классификация:
  - Наивный Байесовский классификатор (Naive Bayes)
  - Скрытая Марковская модель (CMM/HMM)
  - Стохастический градиентный спуск (SGD)
  - Random Forest
- Кластеризация:
  - Метод k-средних (k-Means)
  - «Слабая» или пред кластеризация - Canopy
  - Нечеткая кластеризация - Fuzzy k-Means (Fuzzy C-Means / FCM)
  - StreamingKMeans
  - Спектральная кластеризация (Spectral Clustering)
- Рекомендации:
  - Коллаборативная фильтрация (CF)

Классификация: Naive Bayes

Два алгоритма с поддержкой Map-Reduce:

- Multinomial Naive Bayes
- Transformed Weight-normalized Complement Naive Bayes (CBayes)

Multinomial Naive Bayes - это просто наивный байесовский классификатор

Преимущество: классификаторы быстры и достаточно точны

Недостаток: обычно отказывают, когда размер обучающих примеров для каждого класса не сбалансирован или когда данные не являются достаточно независимыми.

CBayes- комплементарный наивный байесовский классификатор. Пытается решить некоторые недостатки, характерные для байесовского классификатора, при сохранении его простоты и быстродействия.

Классификация: Naive Bayes

Пример исходного корпусе текстов:
http://people.csail.mit.edu/jrennie/20Newsgroups/20news-bydate.tar.gz

1. Запуск предпроцессинга текста. Указываем входной корпус текстов, выходной каталог, где мы получим результаты TF-IDF преобразования (опция -wt tfidf) и параметры самого преобразования, в частности L2 нормы (опция -n 2).

```
mahout seq2sparse
  -i ${PATH_TO_SEQUENCE_FILES}
  -o ${PATH_TO_TFIDF_VECTORS}
  -nv
  -n 2
  -wt tfidf
```

2. Запуск построения модели. По умолчанию используется простой байесовский классификатор. Но можем выбрать модель CBayes (опция -c)

```
mahout trainnb
  -i ${PATH_TO_TFIDF_VECTORS}
  -el
  -o ${PATH_TO_MODEL}/model
  -li ${PATH_TO_MODEL}/labelindex
  -ow
  -c
```

3. Запуск классификатора на тестовой выборке.

```
mahout testnb
  -i ${PATH_TO_TFIDF_TEST_VECTORS}
  -m ${PATH_TO_MODEL}/model
  -l ${PATH_TO_MODEL}/labelindex
  -ow
  -o ${PATH_TO_OUTPUT}
  -c
  -seq
```

Пример запуска обработки исходного корпуса текстов:

https://github.com/apache/mahout/blob/master/examples/bin/classify-20newsgroups.sh

Классификация: Naive Bayes

```
=======================================================
Confusion Matrix
-------------------------------------------------------
 a   b   c   d   e   f   g   h   i   j   k   l   m   n   o   p   q   r   s   t <--Classified as
381  0   0   0   0   9   1   0   0   0   1   0   0   2   0   1   0   0   3   0 |398 a=rec.motorcycles
  1 284  0   0   0   0   1   0   6   3  11   0  66   3   0   6   0   4   9   0 |395 b=comp.windows.x
  2   0 339  2   0   3   5   1   0   0   0   0   1   1  12   1   7   0   2   0 |376 c=talk.politics.mideast
  4   0   1 327  0   2   2   0   0   2   1   1   0   5   1   4  12   0   2   0 |364 d=talk.politics.guns
  7   0   4  32  27  7   7   2   0  12   0   0   6   0 100   9   7  31   0   0 |251 e=talk.religion.misc
 10   0   0   0   0 359   2   2   0   0   3   0   1   6   0   1   0   0  11   0 |396 f=rec.autos
  0   0   0   0   0   1 383   9   1   0   0   0   0   0   0   0   0   3   0   0 |397 g=rec.sport.baseball
  1   0   0   0   0   0   9 382   0   0   0   0   1   1   1   0   2   0   2   0 |399 h=rec.sport.hockey
  2   0   0   0   0   4   3   0 330   4   4   0   5  12   0   0   2   0  12   7 |385 i=comp.sys.mac.hardware
  0   3   0   0   0   0   1   0   0 368   0   0  10   4   1   3   2   0   2   0 |394 j=sci.space
  0   0   0   0   0   3   1   0  27   2 291   0  11  25   0   0   1   0  13  18 |392 k=comp.sys.ibm.pc.hardware
  8   0   1 109   0   6  11   4   1  18   0  98   1   3  11  10  27   1   1   0 |310 l=talk.politics.misc
  0  11   0   0   0   3   6   0  10   6  11   0 299  13   0   2  13   0   7   8 |389 m=comp.graphics
  6   0   1   0   0   4   2   0   5   2  12   0   8 321   0   4  14   0   8   6 |393 n=sci.electronics
  2   0   0   0   0   0   4   1   0   3   1   0   3   1 372   6   0   2   1   2 |398 o=soc.religion.christian
  4   0   0   1   0   2   3   3   0   4   2   0   7  12   6 342   1   0   9   0 |396 p=sci.med
  0   1   0   1   0   1   4   0   3   0   1   0   8   4   0   2 369   0   1   1 |396 q=sci.crypt
 10   0   4  10   1   5   6   2   2   6   2   0   2   1  86  15  14 152   0   1 |319 r=alt.atheism
  4   0   0   0   0   9   1   1   8   1  12   0   3   0   2   0   0   0 341   2 |390 s=misc.forsale
  8   5   0   0   0   1   6   0   8   5  50   0  40   2   1   0   9   0   3 256|394 t=comp.os.ms-windows.misc
=======================================================
Statistics
-------------------------------------------------------
Kappa                                       0.8808
Accuracy                                   90.8596%
Reliability                                86.3632%
Reliability (standard deviation)            0.2131
```

Кластеризация: k-Means

k-Means – простой метод кластеризации с заранее предопределенным количеством кластеров.

Преимущество: простота реализации.

Недостаток: очень чувствителен к первоначальному набору центров кластеров.

Fuzzy k-Means (c-Means) – реализация, позволяющая определить объект к разным кластерам с некой вероятностью.

Что делать если вы не знаете, как расставить начальные центры кластеров?

Используйте Canopy))

Кластеризация: к-Means

Пример исходного корпусе текстов, новости агенства Reuters за 1987 год:
http://kdd.ics.uci.edu/databases/reuters21578/reuters21578.tar.gz

1. Запуск предпроцессинга текста. Удаляем слишком часто используемые термины (опция –maxDFPercent или –x). Все слова преобразуем в вектор в числовом формате. Предварительно необходимо корпус текстов обработать утилитой seqdirectory.

```
mahout seq2sparse
  -i ${PATH_TO_SEQUENCE_FILES}
  -o ${PATH_TO_VECTORS}
  --maxDFPercent 85
  --namedVector
```

2. Запуск кластеризацию. Для старта необходимо иметь стартовые кластера (опция –c). Указываем меру определения расстояния из стандартного класса, например org.apache.mahout.common.distance.CosineDistanceMeasure.

```
mahout kmeans
  -i ${PATH_TO_VECTORS}/tfidf-vectors
  -c ${PATH_TO_CLUSTERS}
  -o ${PATH_TO_OUTPUT}
  -dm …CosineDistanceMeasure
  -c
```

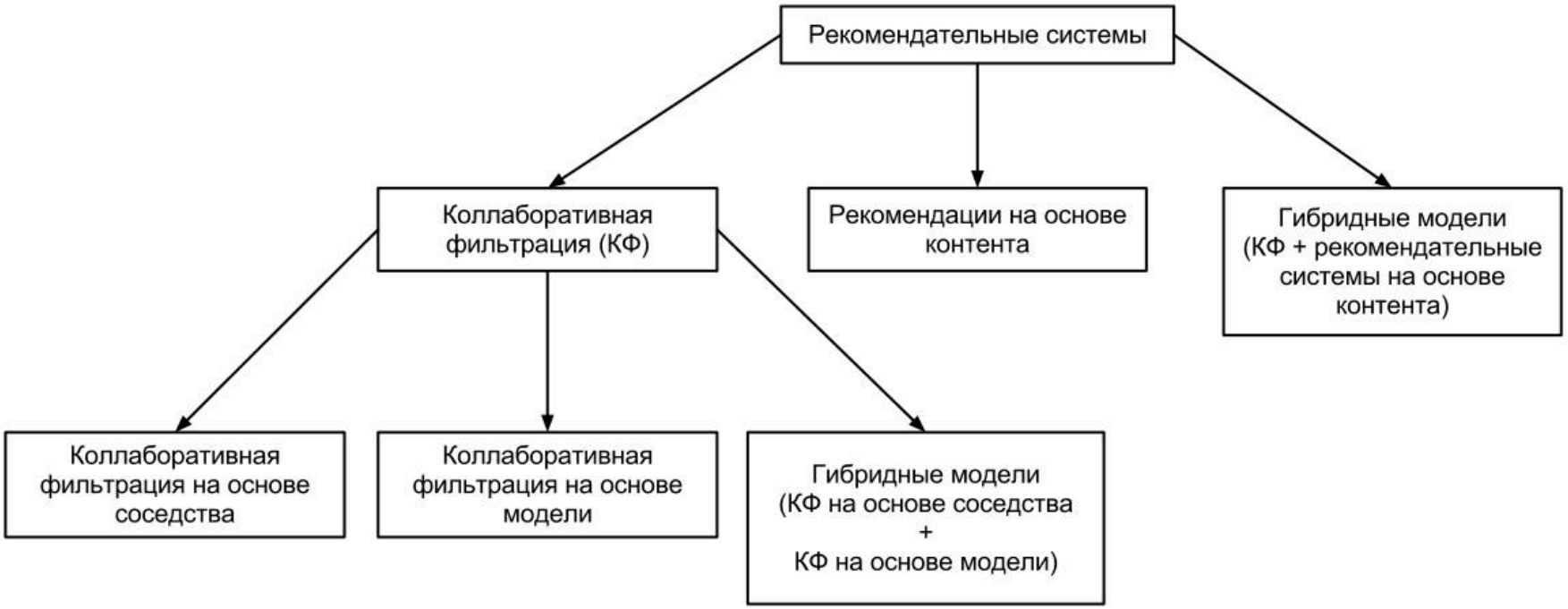
3. Запуск утилиты, преобразующий результат в «читабельный» формат для дальнейшего анализа.

```
mahout clusterdump
  -i ${PATH_TO_OUTPUT}/clusters*final
  -o ${PATH_TO_DUMP}
  -d ${PATH_TO_VECTORS}/dictionary.file-0
  -dt sequencefile … $
{PATH_TO_CLASTERED_POINTS}
```

Пример запуска обработки исходного корпуса текстов:

https://github.com/apache/mahout/blob/master/examples/bin/cluster-reuters.sh

Рекомендации



Рекомендации: Коллаборативная фильтрация




Термин «Коллаборативная фильтрация» впервые употребил Дэвид Голдберг (David Goldberg) из компании Xerox PARC в 1992 году в статье «Using collaborative filtering to weave an information tapestry».

Системы коллаборативной фильтрации обычно применяют двухступенчатую схему:

1. Находят тех, кто разделяет оценочные суждения «активного» (прогнозируемого) пользователя;
2. Используют оценки сходно мыслящих людей, найденных на первом шаге, для вычисления прогноза.

Рекомендации: Коллаборативная фильтрация

Типичный кейс реализации - рекомендация продуктов в eCommerce, такие как Amazon, Netflix, Overstock.

- Строим матрицу, определяющую отношения между парами предметов, для нахождения похожих предметов.
- Используя построенную матрицу и информацию о пользователе, строим прогнозы его оценок.

Рекомендации: Коллаборативная фильтрация

Существуют, как минимум, два подхода в Коллаборативнай фильтрации:

1. User-based filtration - фильтрация по схожести пользователей, базирующаяся на измерении подобия пользователей;

Собрав данные о том, что людям нравится, нужно как-то определить, насколько их вкусы схожи. Для этого каждый человек сравнивается со всеми другими и вычисляется коэффициент подобия (или оценка подобия). Наиболее распространены два способа оценки:

- Эвклидово расстояние

$$d(x,y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2}$$

- Коэффициент корреляции Пирсона (упрощенный аналог, полученная с помощью преобразования)

$$r_{xy} = \frac{n\sum(x_i \cdot y_i) - \sum x_i \cdot \sum y_i}{\sqrt{\left(n\sum x_i^2 - \left(\sum x_i\right)^2\right) \cdot \left(n\sum y_i^2 - \left(\sum y_i\right)^2\right)}}$$

2. Item-based filtration - фильтрация по схожести предметов, сравнивающая оценки, данные различными пользователями.

Наиболее распространенный метод фильтрации. Самый часто используемый подход – выявление людей, которым эти предметы понравились. И посмотреть, какие предметы им понравились еще

Рекомендации: Item-based

Коллаборативная фильтрация по предметам на основании статистики покупок.

Рассмотрим пример с использованием подобия с использованием двоичных данных, построенного на основании расчета косинусного расстояния между векторами:

$$\text{similarity}(\vec{A}, \vec{B}) = \cos(\vec{A}, \vec{B}) = \frac{\vec{A} \cdot \vec{B}}{\|\vec{A}\| * \|\vec{B}\|}$$

Покупатель	**Предмет 1**	**Предмет 2**	**Предмет 3**
Джон	Купил	Не покупал	Купил
Марк	Не покупал	Купил	Купил
Люси	Не покупала	Купила	Не покупала

Рассчитаем подобие/косинус между «Предмета 1» и «Предмет 2»:

$$\frac{(1,0,0) \cdot (0,1,1)}{\|(1,0,0)\| * \|(0,1,1)\|} = 0$$

«Предмета 1» и «Предмет 3»:

$$\frac{(1,0,0) \cdot (1,1,0)}{\|(1,0,0)\| * \|(1,1,0)\|} = \frac{1}{\sqrt{2}} \approx 0.71$$

«Предмета 2» и «Предмет 3»:

$$\frac{(0,1,1) \cdot (1,1,0)}{\|(0,1,1)\| * \|(1,1,0)\|} = \frac{1}{2} = 0.5$$

Таким образом, пользователь, находящийся на странице «Предмета 1», получит «Предмет 3» в качестве рекомендации; на странице «Предмета 2» — «Предмет 3» и на странице «Предмета 3» — «Предмет 1» (и затем «Предмет 2»).

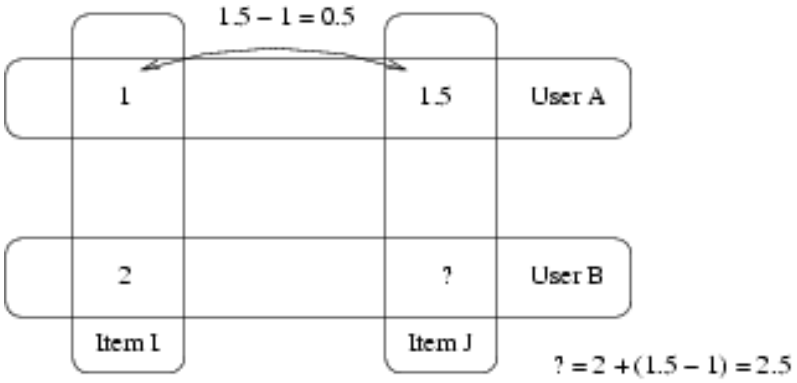
Рекомендации: Slop One алгоритм

Коллаборативная фильтрация Slope One для предметов с оценками.

Более простой алгоритм с существенным уменьшением требований к ресурсам, основанный на отношении рейтинга двух предметов.

Отличие от регрессионного анализа состоит в использовании упрощенной формулы регрессии с всего одним предикатом:

$$f(x) = x + b$$

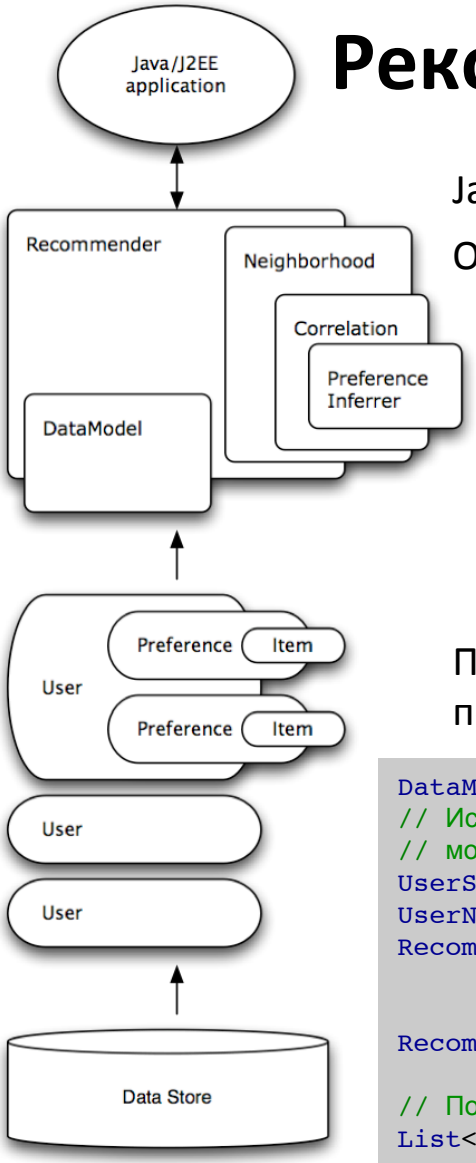


- Джо выставил оценку 1 для Селин Дион и 1.5 для Линдсей Лохан.
- Джил оценила Селин Дион на 2 балла.
- Какую оценку выставит Джил для Линдсей Лохан?
- Ответ алгоритма Slope One: 2.5 (1.5-1+2=2.5).

Чтобы применить алгоритм Slope One для заданных n предметов, надо рассчитать и сохранить среднюю разницу и количество голосов для каждой из n^2 пар предметов.

Семейство алгоритмов описаны в статье «Slope One Predictors for Online Rating-Based Collaborative Filtering» Daniel Lemire и Anna Maclachlan. http://lemire.me/fr/abstracts/SDM2005.html

Рекомендации: Apache.teste



Java конструктор рекомендаций org.apache.mahout.cf.taste.impl

Основные реализованные классы:

- DataModel – данные вида <userID>,<itemID>,<prefValue>
- UserSimilarity – определение близости для модели User-Based
- ItemSimilarity – определение близости для модели Item-Based
- UserNeighborhood – группировка пользователей по близости
- Recommender
  - GenericUserBasedRecommender – метод схожести пользователей
  - GenericItemBasedRecommender – метод схожести объектов
  - CachingRecommender

Пример кода для реализации фильтрации методом схожести пользователей:

```
DataModel model = new FileDataModel(new File("data.txt"));
// Используем встроенный метод, на основании коэффициента корреляции Пирсона,
// можно использовать, в качестве альтернативы, более простую метрику EuclideanDistanceSimilarity
UserSimilarity userSimilarity = new PearsonCorrelationSimilarity(model);
UserNeighborhood neighborhood = new NearestNUserNeighborhood(3, userSimilarity, model);
Recommender recommender = new GenericUserBasedRecommender(model,
                                                           neighborhood,
                                                           userSimilarity);
Recommender cachingRecommender = new CachingRecommender(recommender);

// Получаем рекомендацию в виде RecommendedItem [item:XXX, value:Y]
List<RecommendedItem> recommendations = cachingRecommender.recommend(1234, 10);
```

Рекомендации: Apache.teste

Проблема реализации на основании близости пользователей – высокая вычислительная сложность и то, что в Mahout **нет** распределенной реализации фильтрации методом схожести пользователей.

Хороший пример для тестирования «Million Song Dataset»: http://labrosa.ee.columbia.edu/millionsong/tasteprofile

P.S: Используется в качестве примера в облачном сервисе Microsoft Azure, использующем Hadoop и Mahout: http://azure.microsoft.com/ru-ru/documentation/articles/hdinsight-mahout/

Пример кода для реализации фильтрации методом схожести объектов Item-Based в teste:

```
DataModel model = new FileDataModel(new File("data.txt"));
Collection<GenericItemSimilarity.ItemItemSimilarity> correlations =
...;
ItemSimilarity itemSimilarity = new GenericItemSimilarity(correlations);
Recommender recommender = new GenericItemBasedRecommender(model, itemSimilarity);
Recommender cachingRecommender = new CachingRecommender(recommender);

// Получаем рекомендацию
List<RecommendedItem> recommendations = cachingRecommender.recommend(1234, 10);
```

Рекомендации: Item-Based with Hadoop

- Матрицу близости объектов можно посчитать заранее
- Вектор объектов, обычно, менее разрежен, чем вектор пользователей
- Есть реализация под Hadoop

Пример запуска простой задачии. На вход подается файл все в том же формате userID, itemID and preference, на выходе userID и масив itemIDs и scores:

```
$ mahout recommenditembased -s SIMILARITY_LOGLIKELIHOOD -i /path/to/input/file -o /path/to/
desired/output --numRecommendations 25
```

Более «продвинутый» алгоритм, рекомендованный для использования в eCommerce. Построенный с использованием метода Alternating Least Squares (ALS, или Метод покоординатного спуска) . В отличии от классических подходов item и user based, способен дать прогноз на основании скрытых факторов. Первый шаг, для получения ALS матрицы факторов:

```
$ mahout parallelALS --input $als_input --output $als_output --lambda 0.1 --implicitFeedback
true --alpha 0.8 --numFeatures 2 --numIterations 5  --numThreadsPerSolver 1 --tempDir tmp
```

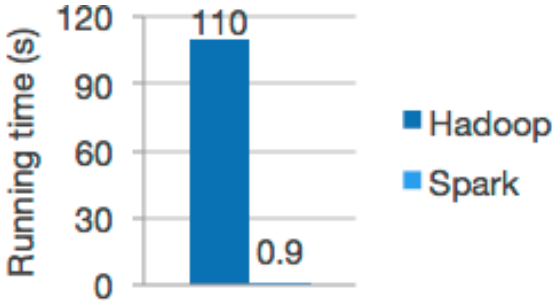
Второй шаг, на вход которому подается полученная на первом шаге матрица, формерует рекомендации для пользователей:

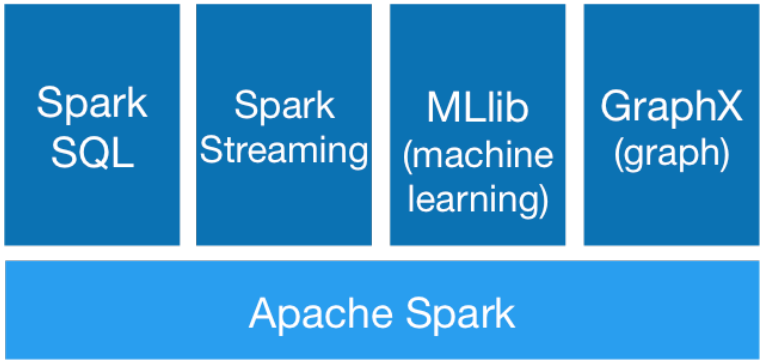
```
$ mahout recommendfactorized --input $als_input --userFeatures $als_output/U/ --itemFeatures
$als_output/M/ --numRecommendations 1 --output recommendations --maxRating 1
```

Рекомендации: Mahout with Spark

Spark работает как на Hadoop, Mesos, так и на локально, или в облаке. Может получить доступ к разнообразным данным, в том числе HDFS, Cassandra, HBase, S3.

Преимущество: запуск задач в 100 раз быстрее Hadoop MapReduce и в 10 раз быстрее, чел локально «с диска». Поддерживает Scala и Python.





Spark SQL - унифицирует доступ к структурированным данным.

Spark Streaming - позволяет легко строить масштабируемые отказоустойчивые потоковых приложений.

MLlib - масштабируемая machine learning библиотека.

GraphX - API для графов и диаграммы-параллельные вычисления.

Рекомендации: Cooccurrence Recommenders

spark-itemsimilarity – аналог Item-Based реализации Mahuot mapreduce версии.

Если все значения оставить по умолчанию, то запустить очень просто:

```
userID1,itemID1
userID2,itemID2
...
```

С командной строки:

```
bash$ mahout spark-itemsimilarity --input in-file --output out-dir
```

Это позволит локально запустит Spark, который выведет текст:

```
itemID1<tab>itemID2:value2<space>itemID10:value10...
```

spark-rowsimilarity – является дополнением к spark-itemsimilarity. На вход можно подать данные выхода предыдущей модели и получить на выходе группы, как пользователей так и объектов, схожие по результатам рекомендаций:

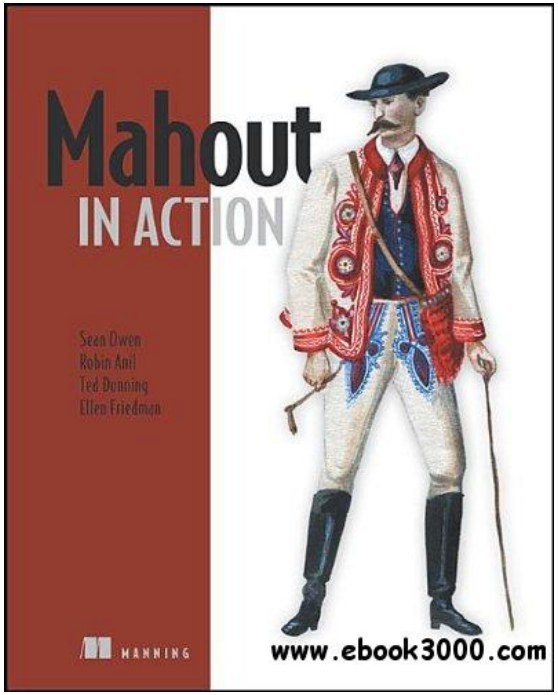
```
rowID<tab>columnID1:strength1<space>columnID2:strength2...
```

На выходе, с сортировкой по убыванию:

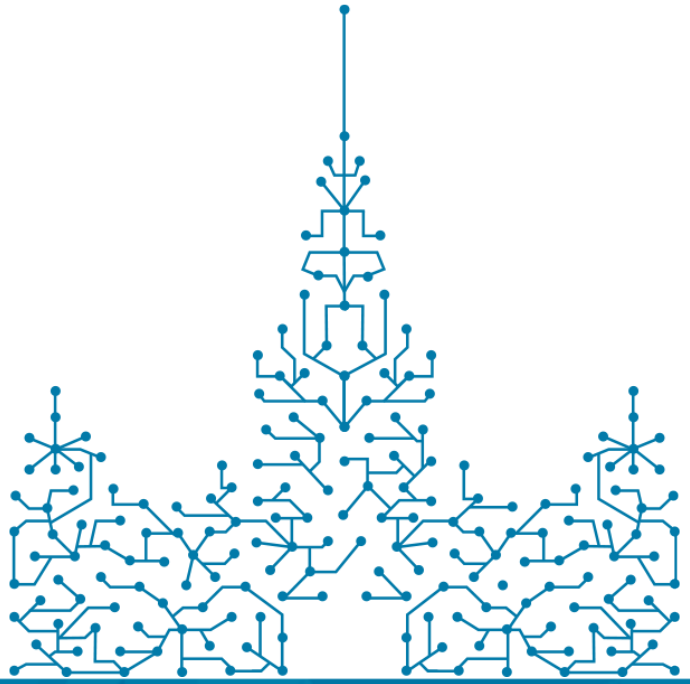
```
rowID<tab>rowID1:strength1<space>rowID2:strength2...
```

Также может использоваться для поиска схожего текстового контента, например сравнивая тэги или токены различных постов.

Ресурсы



Apache Mahout in Action
Sean Owen, Robin Anil



Вопросы?