# Benchmarking Spark Machine Learning Using BigBench

Sweta Singh(✉)

IBM, Dallas, USA
singhswe@us.ibm.com

**Abstract.** Databases such as dashDB are adding High Speed Connectors for Spark to efficiently extract large volumes of data. This allows them to be combined with other unstructured data sources and perform Machine Learning (ML) on top of it. Machine Learning is a key ingredient for such use cases. In order to assess performance of the data connectors and machine language frameworks, we sought benchmarks that have the ability to scale the size of datasets to very large volumes and apply Machine Learning algorithms. After exploring several options, we found BigBench to be a good fit. In this paper, we talk about our experiences of using BigBench with special focus on its 5 Machine Learning queries and their default implementation in Spark. We discuss on how we could improve effectiveness of BigBench for benchmarking Machine Learning by avoiding bias and inclusion of real time analytics. We also think that there is scope for improving the coverage of Machine Learning by adding more use cases like Collaborative Filtering. Lastly, we share some interesting visualization of 4 ML queries using SPSS Modeler and our experiments on different Clustering and Classification algorithms.

**Keywords:** Collaborative filtering using machine learning · Predicting accuracy of data sets · Visualization of bigbench machine learning queries using SPSS

## 1 Introduction

Machine Learning is being increasingly applied on large volumes of data to be able to predict outcomes with high efficiency and accuracy. The performance of such use cases are determined by two key aspects

(a) Optimized data exchange between analytics engines like Spark [1, 18, 19] and the data store. This is important due to the iterative nature of machine learning algorithms, especially for large data sets that do not fit in memory
(b) Scalability and accuracy of the Machine Learning frameworks

To address the first aspect above, several databases like IBM dashDB [2], Cassandra, Couchbase etc. are employing techniques for optimized data connectors that make bi-directional communication between Spark and databases more efficient by

(a) Generating parallel SQL statements under the covers, to read from multiple database partitions concurrently

(b) Localizing the data exchange between Spark and database node if they are hosted on the same cluster

IBM dashDB Local [3] has the capability to run Spark applications that analyze data in a dashDB database and write results back to the database. It achieves highly optimized and parallel data transfer between dashDB and Spark by collocating DB2 data nodes and Spark executors. An I/O layer, implemented as a DB2 Fenced Mode Process (FMP), acts as an interface between dashDB and Spark. dashDB unloads the data of the local partition into the FMP, which then passes the data to the Spark layer. A single transfer unit contains multiple blocks and potentially millions of rows, hence providing significant speed up for exchange of data.

We sought benchmarks to evaluate the performance benefits of connectors. These were the key pre-requisites:

(a) The benchmark should be representative of a good use case for Spark and database integration. Machine Learning has to be a key component of the use case
(b) It should be able to scale data volumes, allowing for large data volume transfers and be bidirectional - read/write to database
(c) It should invoke Machine Learning algorithms via SQL interface (Stored Procedure) or via Spark jobs (using customized RDD to connect to data source)
(d) It should support multiple streams to test both scalability and resource management in an integrated solution where Spark and database co-exist on the same cluster

One option that we explored were open data sets available as part of UCI machine learning repository [4]. They have 336 datasets based on "real" world data, which is precisely the reason why they are an attractive option. However, after examining some of the new and popular data sets, we found that their key drawback was the small dataset size, ranging from few KB to less than 500 MB.

Given our key requirements of large data transfers, ability to scale and run multi-stream, we found BigBench [7–9, 17] to be an apt choice. There are five queries in BigBench (Q05, Q20, Q25, Q26 and Q28) that involve interaction between database and Spark. These cover three Machine Learning algorithms in Clustering (K-Means) and Classification (Logistic Regression and Naive Bayes). Since data exchange volumes will be large and data cannot be entirely cached in memory for reuse, the iterative nature of Machine Learning algorithms involves multiple trips to the data source and hence the performance of data connectors is stressed well enough. BigBench also supports bidirectional transfer between data layer and Spark by supporting writing back the scoring results to the database.

Our experiments were conducted with the following configuration:

BigBench Scale Factor = 1 TB
dashDB Local cluster, CentOS7.0-64 and Spark 1.6.2
4 nodes with the following configuration:

- 24 cores (2.6 GHz Intel Xeon-Haswell)
- 512 GB memory
- 6 internal SSDs (960 GB SanDisk CloudSpeed 1000 SSD)
- 10000 Mbps full duplex N/W card

We also studied the effectiveness of BigBench in assessing Machine Learning frameworks as we intend to extend our study to assess Machine Learning algorithm implementation in Spark MLlib versus IBM ML algorithms.

In the following sections, we discuss our key observations and recommendations.

## 2 Collaborative Filtering Using Machine Learning

We propose to add a "Recommender System" use case in BigBench. It is a very relevant use case in e-commerce and can be implemented using Collaborative filtering, one of the most widely used and researched methods. Collaborative Filtering is based on the principle that if two users rate x items similarly, they will likely rate other items similarly. It is known for two unique challenges:

(a) Data Sparsity: For a large customer base and a large product set, the subset of items rated by users is very small. This leads to a sparse user-item association matrix and hence poses a challenge to the predictions of the algorithm.
(b) Scalability: With a large customer base and range of items, the computational complexity of Collaborative Filtering grows very quickly. There are several optimizations to reduce the cost of calculating similarity but there is often a trade-off between performance and accuracy.

Matrix Factorization is one of the key methods to implement Collaborative Filtering. It is often classified as a latent factor model. The ratings are explained by characterizing both items and users on a number of factors automatically inferred from the ratings pattern [13–15]. The sparse rating matrix is modeled as the product of low-rank user and item factors. Latent factors are learnt by minimizing the reconstruction error of the observed ratings. The unknown ratings can then be computed by multiplying these factors.

Given that BigBench already provides the [user, product, review] triplet in web_clickstreams, Matrix Factorization will be a good add-on to the BigBench Machine Learning arsenal to study the performance of the Machine Learning framework and the cluster.

Spark MLlib [11] implements Matrix Factorization using Alternating Least Squares (ALS) method [16, 20–22]. ALS alternates between fixing the user feature matrix and the item feature matrix. When one is fixed, the other is solved by minimizing the Root Mean Squared Error. This is repeated until convergence.

Below, we share an initial prototype for BigBench Recommendation System using Explicit Feedback.

The input Vector can be extracted from product_reviews table in 3 possible ways

(a) Select all items, users, ratings from product_reviews table
    SELECT PR_USER_SK, PR_ITEM_SK, PR_REVIEW_RATING FROM product_reviews;
(b) Predict recommendations in a specific item category
    SELECT PR_USER_SK, PR_ITEM_SK, PR_REVIEW_RATING FROM product_reviews pr, item

WHERE I_CATEGORY = ‘Books’ AND i_item_sk = pr_item_sk

(c) Predict recommendations to specific item category and class
SELECT PR_USER_SK, PR_ITEM_SK, PR_REVIEW_RATING FROM product_reviews pr, item WHERE I_CATEGORY = ‘Books’ AND I_CLASS = ‘fiction’

The Spark job for Recommender does the following:

(a) Accepts arguments for iterations and rank that are parameters of the ML algorithm
(b) Transforms Rows of data frame to Rating object
(c) Trains ALS model on a partial data set (90%) and predicts the outcome on the held back data
(d) Measures the accuracy using Root Mean Squared Error (RMSE)

Below is a snippet from Recommender output on 1 TB scale factor of BigBench. At this scale factor, the product_reviews table has 4.45 million ratings. 1.7 million customers have rated 556,000 items.

```
Actual Rating: 5.0 Predicted Rating: 4.46
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 2.0 Predicted Rating: 1.99
Actual Rating: 5.0 Predicted Rating: 4.98
Actual Rating: 4.0 Predicted Rating: 3.95
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 5.42
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 5.0 Predicted Rating: 4.99
Actual Rating: 4.0 Predicted Rating: 4.03
Actual Rating: 1.0 Predicted Rating: 0.94
Actual Rating: 5.0 Predicted Rating: 4.84
```

Spark uses a blocked implementation of ALS. A User column block stores a subset of user factor matrix and an Item column block stores a subset of item factor matrix. The rating matrix is stored as two separate RDDs. One RDD is partitioned by user and the other RDD is partitioned by item. While updating the user factor matrix, ratings for each user in its partition is locally available. However, item factor vector corresponding to items rated by the user must be shuffled across the nodes.

Below is a snapshot of ALS training stage execution with twenty iterations. Figure 1 shows that Job 3 is the most expensive job. Figure 2 shows a snippet of Job 3 breakdown. Job 3 comprises forty stages, two for each iteration. The user or the item factor matrix is updated alternately in each stage. At the end of each stage, the updated factor matrix is shuffled across the nodes. A major chunk of the time is spent in the shuffle.

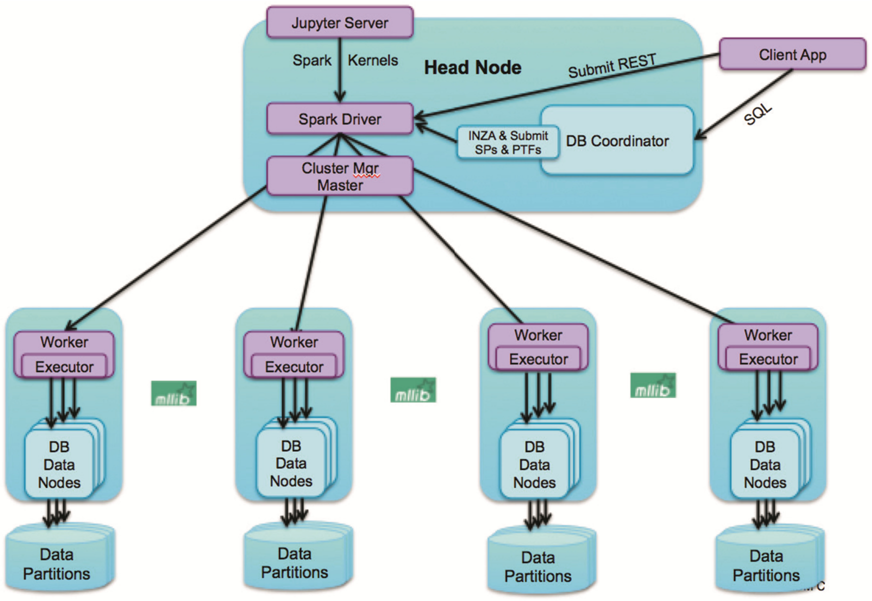
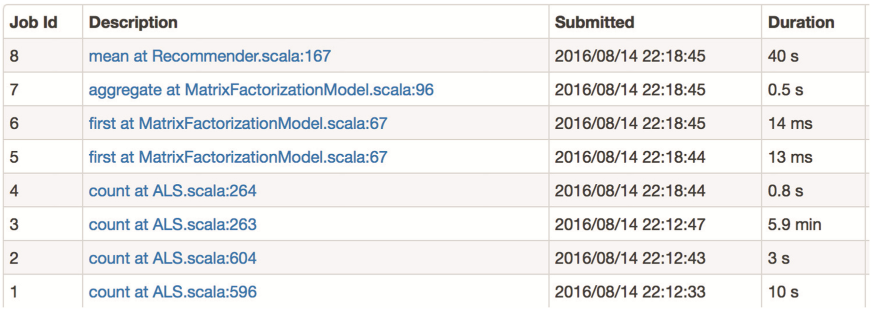


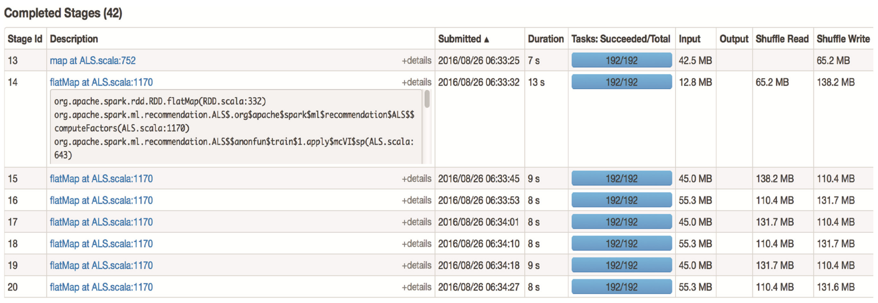
**Fig. 1.** dashDB Spark integration layout



| Job Id | Description | Submitted | Duration |
|---|---|---|---|
| 8 | mean at Recommender.scala:167 | 2016/08/14 22:18:45 | 40 s |
| 7 | aggregate at MatrixFactorizationModel.scala:96 | 2016/08/14 22:18:45 | 0.5 s |
| 6 | first at MatrixFactorizationModel.scala:67 | 2016/08/14 22:18:45 | 14 ms |
| 5 | first at MatrixFactorizationModel.scala:67 | 2016/08/14 22:18:44 | 13 ms |
| 4 | count at ALS.scala:264 | 2016/08/14 22:18:44 | 0.8 s |
| 3 | count at ALS.scala:263 | 2016/08/14 22:12:47 | 5.9 min |
| 2 | count at ALS.scala:604 | 2016/08/14 22:12:43 | 3 s |
| 1 | count at ALS.scala:596 | 2016/08/14 22:12:33 | 10 s |

**Fig. 2.** Spark monitoring UI showing the jobs

We did some experiments to study the impact of the number of factors or rank on the performance and accuracy of Collaborative Filtering. The intermediate RDDs are 100% cached. As we increase the rank, we observe that the accuracy of the algorithm changes, with consistent drop in performance. This performance drop is attributed to increase in shuffle time due to larger size of the factor matrix (Fig. 3).



Completed Stages (42)

| Stage Id | Description | | Submitted ▲ | Duration | Tasks: Succeeded/Total | Input | Output | Shuffle Read | Shuffle Write |
|---|---|---|---|---|---|---|---|---|---|
| 13 | map at ALS.scala:752 | +details | 2016/08/26 06:33:25 | 7 s | 192/192 | 42.5 MB | | | 65.2 MB |
| 14 | flatMap at ALS.scala:1170<br>org.apache.spark.rdd.RDD.flatMap(RDD.scala:332)<br>org.apache.spark.ml.recommendation.ALS$.org$apache$spark$ml$recommendation$ALS$$<br>computeFactors(ALS.scala:1170)<br>org.apache.spark.ml.recommendation.ALS$$anonfun$train$1.apply$mcVI$sp(ALS.scala:<br>643) | +details | 2016/08/26 06:33:32 | 13 s | 192/192 | 12.8 MB | | 65.2 MB | 138.2 MB |
| 15 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:33:45 | 9 s | 192/192 | 45.0 MB | | 138.2 MB | 110.4 MB |
| 16 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:33:53 | 8 s | 192/192 | 55.3 MB | | 110.4 MB | 131.7 MB |
| 17 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:34:01 | 8 s | 192/192 | 45.0 MB | | 131.7 MB | 110.4 MB |
| 18 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:34:10 | 8 s | 192/192 | 55.3 MB | | 110.4 MB | 131.7 MB |
| 19 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:34:18 | 9 s | 192/192 | 45.0 MB | | 131.7 MB | 110.4 MB |
| 20 | flatMap at ALS.scala:1170 | +details | 2016/08/26 06:34:27 | 8 s | 192/192 | 55.3 MB | | 110.4 MB | 131.6 MB |

**Fig. 3.** Spark monitoring UI showing a breakdown of the expensive job

We also propose to extend the usage of Collaborative Filtering to construct a streaming scenario in BigBench. We can stream a portion of web click stream data during the workload execution and use saved MatrixFactorization model to predict the use rating or generate top item recommendations for users on the fly (Fig. 4).

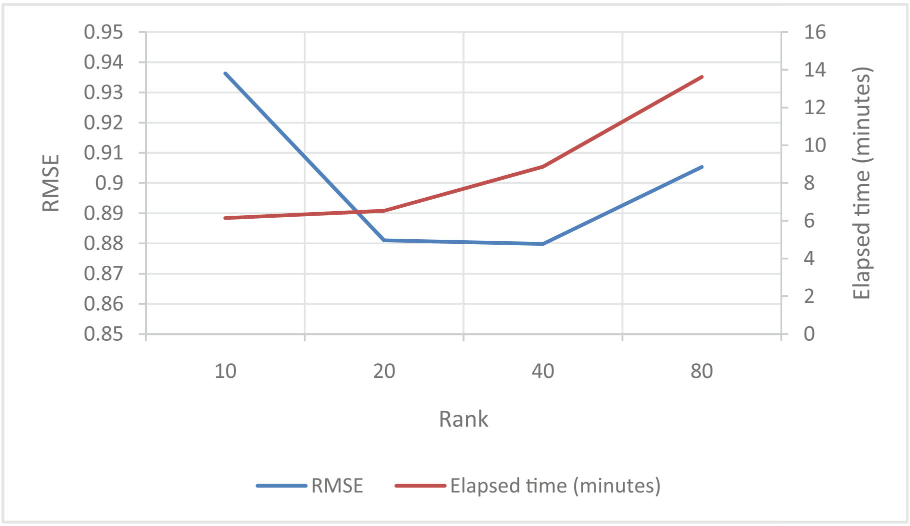


**Fig. 4.** Impact of increasing rank on RMSE & Performance of ALS (Lower is better)

## 3 Measuring Accuracy and Tuning Machine Learning Algorithms

A best practice in Machine Learning is to test the accuracy of the model on a data set that it is not trained on. In other words, we should have disjoint data sets for training and testing the models. The accuracy is reported on the test data set. We see that the data set is split for Naïve Bayes algorithm using SQL queries to segregate data but not so for Logistic Regression. We'd recommend making the measurement of accuracy consistent across all supervised ML algorithms in BigBench by either employing a method similar to Naïve Bayes to split the training/test data or using mechanisms like Cross Validation to achieve the same. This will help avoid bias and obtain a better estimate of the model's generalized error.

In a modified Logistic Regression scenario similar to Scenario #2 in Sect. 4.1 below, we split the data set into a 70% training data and 30% test data. Running prediction on the training data set vs running prediction on a 30% test data set suggests that Area Under Curve (AUC) on test data reduces by 4.4% (Fig. 5).

| | Training data | Test data |
|---|---|---|
| Precision | 0.7587515979164543 | 0.7281337029540884 |
| AUC | 0.5253761346454113 | 0.5024392467169441 |
| Confusion Matrix | 2284497.0 276466.0 | 662283.0 180646.0 |
| | 478607.0 90287.0 | 74435.0 20895.0 |

**Fig. 5.** Comparison of accuracy on training data vs test data

On a related note, one common use case in Machine Learning is model selection and tuning the parameters of Machine Learning algorithms. We could modify Q05 or any new ML algorithm in BigBench to represent such a ML tuning use case. For example, test the Logistic Regression or Collaborative Filtering with multiple values of regularization parameters. The optimal regularization parameter is one that reports the best accuracy on the test data set.

## 4 Visualization of Machine Learning Use Case in SPSS Modeler

IBM SPSS Modeler [3] is a powerful data mining workbench that helps build accurate predictive models quickly and intuitively, without the need for any programming. It provides a rich set of machine learning algorithms and facilitates comparison of alternate ML models.

We analyzed the existing 4 Machine Language use cases in BigBench using SPSS Modeler. Our intent was to:

(a) Gain insights about the data and relative importance of features in predicting outcomes
(b) Assess the cluster quality and size for the three K-Means clustering scenarios
(c) Assess alternative models that could be incorporated into the BigBench test suite for increased coverage

We discuss our findings in the sections below.

### 4.1 Q05 – Logistic Regression

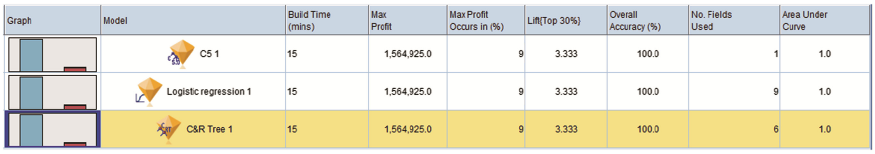
This use case predicts if a visitor will be interested in a given item category, based on demographics and existing users online activities (interest in items of different categories).

The target variable for the use case is a binary variable indicating whether a visitor is interested in a specific item category. It is 1 if the user's clicks in the category is greater than the average clicks in that category for the entire population.

**Model Assessment.** The model selected and its accuracy depend on the clicks in the specified item category. If the target category is included in the input feature vector, the model is able to predict with 100% accuracy, as it is able to learn that the clicks of the category are determining the outcome. Therefore, we suggest adding a use case in which the clicks in the item category are deliberately excluded from the input feature vector. This will enable BigBench to exercise a wider range of ML algorithms like Neural Networks and also, help increase the computational complexity of these algorithms e.g. increased depth of the Tree. The experiments below provide more details.

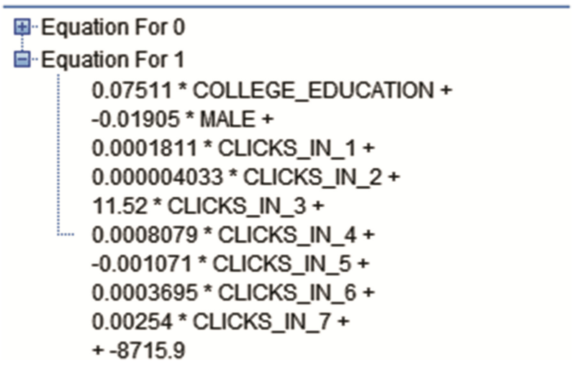
Scenario #1: Q05 is executed with target item category "Books". The target variable is 1 if the number of clicks by a customer is greater than average number of clicks in "Books" category and 0 otherwise. CLICKS_IN_3 column is the number of clicks in "Books" category and is a part of input feature vector.

SPSS Modeler evaluated several classification algorithms and arrived at three models: C51, Logistic Regression and C&R Tree. C51 and C&R Tree belong to the tree family. As shown below, all algorithms predict with 100% accuracy, as they are able to determine CLICKS_IN_3 as the key predictor (Fig. 6).



| Graph | Model | Build Time (mins) | Max Profit | Max Profit Occurs in (%) | Lift{Top 30%} | Overall Accuracy (%) | No. Fields Used | Area Under Curve |
|---|---|---|---|---|---|---|---|---|
| | C5 1 | 15 | 1,564,925.0 | 9 | 3.333 | 100.0 | 1 | 1.0 |
| | Logistic regression 1 | 15 | 1,564,925.0 | 9 | 3.333 | 100.0 | 9 | 1.0 |
| | C&R Tree 1 | 15 | 1,564,925.0 | 9 | 3.333 | 100.0 | 6 | 1.0 |

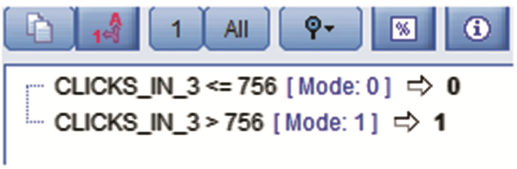
**Fig. 6.** Top 3 Classification models filtered by SPSS, sorted by Area Under Curve



```
Equation For 0
Equation For 1
    0.07511 * COLLEGE_EDUCATION +
    -0.01905 * MALE +
    0.0001811 * CLICKS_IN_1 +
    0.000004033 * CLICKS_IN_2 +
    11.52 * CLICKS_IN_3 +
    0.0008079 * CLICKS_IN_4 +
    -0.001071 * CLICKS_IN_5 +
    0.0003695 * CLICKS_IN_6 +
    0.00254 * CLICKS_IN_7 +
    + -8715.9
```

**Fig. 7.** Parameters of Logistic Regression

Logistic Regression shows very high correlation for CLICKS_IN_3. Similarly, C51 is able to infer the condition that decides the target variable in a one-level tree structure (Fig. 7).

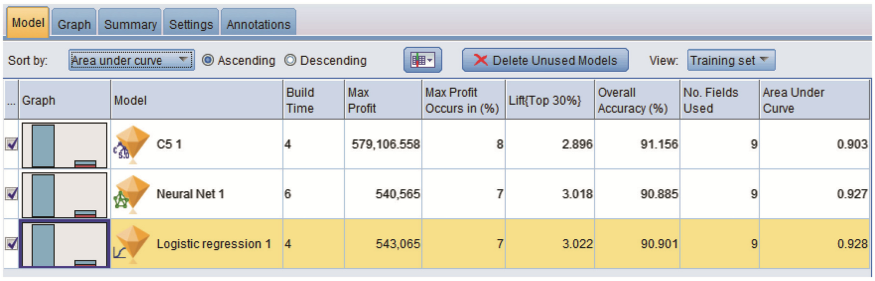


```
CLICKS_IN_3 <= 756 [ Mode: 0 ] ⇨ 0
CLICKS_IN_3 > 756 [ Mode: 1 ] ⇨ 1
```

**Fig. 8.** Tree output of C51 Model

Scenario #2: Q05 is executed with target item category “Toys & Games”. The target variable is 1 if the number of clicks by a customer is greater than average number of clicks in “Toys & Games” category and 0 otherwise. Clicks in this category are “not” part of input feature vector (Fig. 8).

SPSS Modeler shows that three models (C51, Neural Network and Logistic Regression) have relatively high accuracy amongst several classification algorithms. The accuracy level is less than 100% (Fig. 9).

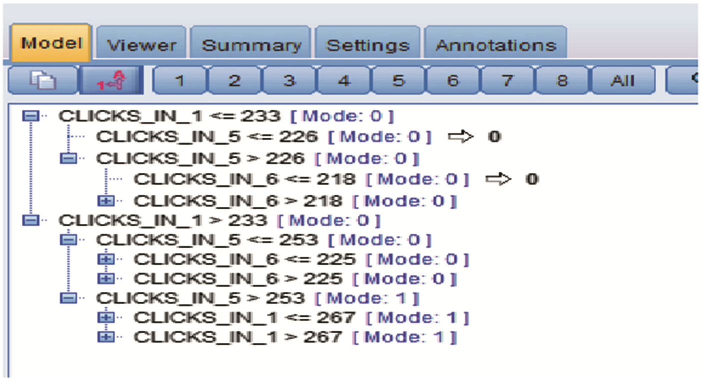


Sort by: Area under curve ◉ Ascending ○ Descending View: Training set

| ... | Graph | Model | Build Time | Max Profit | Max Profit Occurs in (%) | Lift{Top 30%} | Overall Accuracy (%) | No. Fields Used | Area Under Curve |
|---|---|---|---|---|---|---|---|---|---|
| ✓ | | C5 1 | 4 | 579,106.558 | 8 | 2.896 | 91.156 | 9 | 0.903 |
| ✓ | | Neural Net 1 | 6 | 540,565 | 7 | 3.018 | 90.885 | 9 | 0.927 |
| ✓ | | Logistic regression 1 | 4 | 543,065 | 7 | 3.022 | 90.901 | 9 | 0.928 |

**Fig. 9.** Top 3 classification models filtered by SPSS, sorted by Area Under Curve

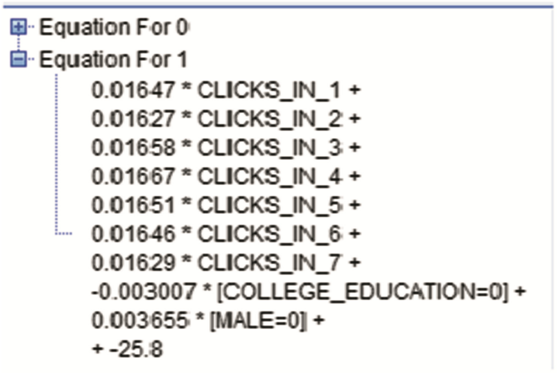
**C51 Model.** The Fig. 10 shows the decision tree structure produced by C51 Model [6]. It is worth noting that the optimal tree depth selected for producing high accuracy is 25. Adding a tree based classification algorithm to BigBench would be an interesting add-on.



```
⊟ CLICKS_IN_1 <= 233 [ Mode: 0 ]
    CLICKS_IN_5 <= 226 [ Mode: 0 ] ⇨ 0
  ⊟ CLICKS_IN_5 > 226 [ Mode: 0 ]
      CLICKS_IN_6 <= 218 [ Mode: 0 ] ⇨ 0
    ⊞ CLICKS_IN_6 > 218 [ Mode: 0 ]
⊟ CLICKS_IN_1 > 233 [ Mode: 0 ]
  ⊟ CLICKS_IN_5 <= 253 [ Mode: 0 ]
    ⊞ CLICKS_IN_6 <= 225 [ Mode: 0 ]
    ⊞ CLICKS_IN_6 > 225 [ Mode: 0 ]
  ⊟ CLICKS_IN_5 > 253 [ Mode: 1 ]
    ⊞ CLICKS_IN_1 <= 267 [ Mode: 1 ]
    ⊞ CLICKS_IN_1 > 267 [ Mode: 1 ]
```

**Fig. 10.** Tree output by C51 Model

**Logistic Regression Model.** The Fig. 11 shows the parameters chosen by Logistic regression for the feature vector. Demographics have negligible impact on the target variable (Fig. 12).



```
Equation For 0
Equation For 1
    0.01647 * CLICKS_IN_1 +
    0.01627 * CLICKS_IN_2 +
    0.01658 * CLICKS_IN_3 +
    0.01667 * CLICKS_IN_4 +
    0.01651 * CLICKS_IN_5 +
    0.01646 * CLICKS_IN_6 +
    0.01629 * CLICKS_IN_7 +
    -0.003007 * [COLLEGE_EDUCATION=0] +
    0.003655 * [MALE=0] +
    + -25.8
```

**Fig. 11.** Parameters of Logistic Regression

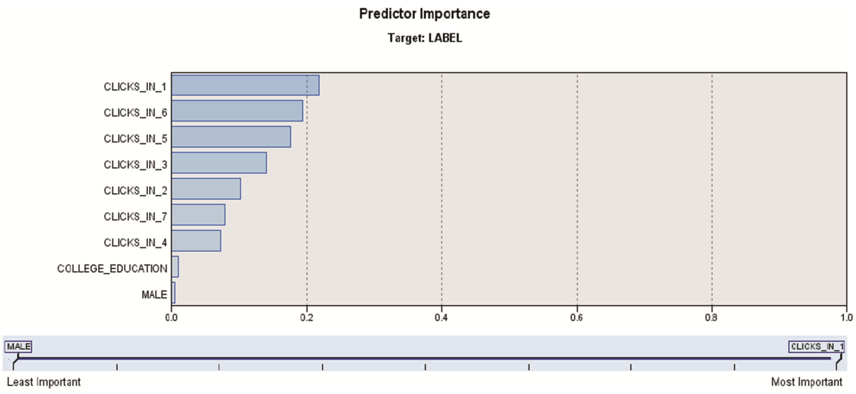


**Fig. 12.** Predictor importance using C51 algorithm

**Predictor Importance.** Both logistic regression and C51 show that the number of clicks in different categories dictates the interest of visitor in "Toys & Games" category. Demographics have the least impact on the target variable (Fig. 13).

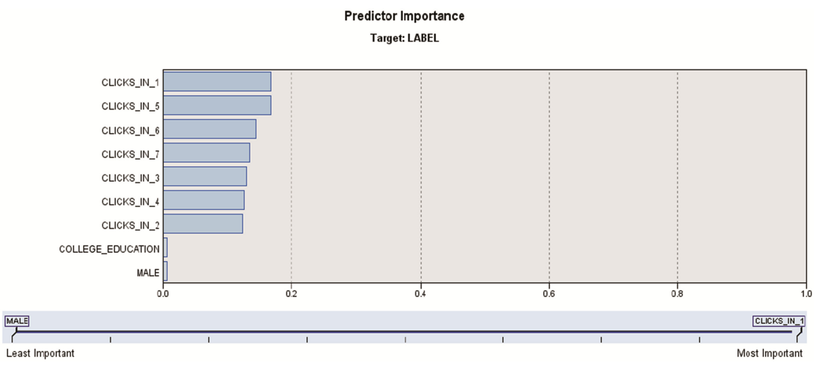
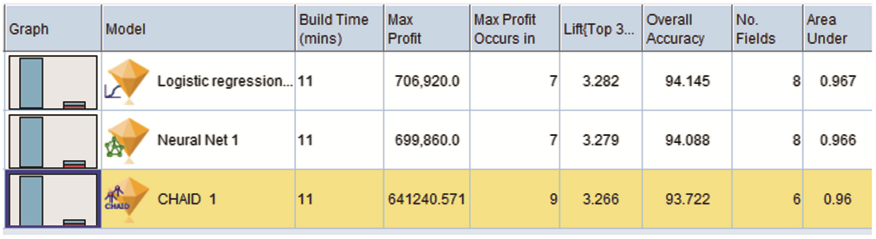


**Fig. 13.** Predictor importance using Logistic Regression

Similar results were observed when the target item category was “Books” and CLICKS_IN_3 was “not” included in the input feature vector (Fig. 14).



| Graph | Model | Build Time (mins) | Max Profit | Max Profit Occurs in | Lift{Top 3... | Overall Accuracy | No. Fields | Area Under |
|---|---|---|---|---|---|---|---|---|
| | Logistic regression... | 11 | 706,920.0 | 7 | 3.282 | 94.145 | 8 | 0.967 |
| | Neural Net 1 | 11 | 699,860.0 | 7 | 3.279 | 94.088 | 8 | 0.966 |
| | CHAID 1 | 11 | 641240.571 | 9 | 3.266 | 93.722 | 6 | 0.96 |

**Fig. 14.** Top 3 Classification models filtered by SPSS

**Key Inferences.**

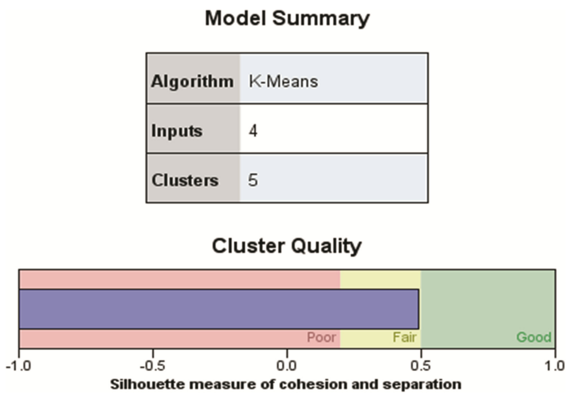
As mentioned, not including the deterministic clicks in the input feature vector will exercise and stress the machine learning algorithms in a more realistic way. This clearly reflects in the tree depth – 25 in Scenario #2 versus only 1 in Scenario #1 Another benefit of Scenario #2 is the ability to introduce more algorithms such as Trees and Neural Networks to the BigBench ML mix.

### 4.2 K-Means Clustering

In SPSS, the quality of clustering is measured by the Silhouette coefficient. Silhouette coefficient combines the concepts of cluster cohesion (favoring models which contain tightly cohesive clusters) and cluster separation (favoring models which contain highly separated clusters).

Q20 performs Customer segmentation for return analysis: Customers are separated along the following dimensions: return frequency, return order ratio (total number of orders partially or fully returned versus the total number of orders), return item ratio (total number of items returned versus the number of items purchased) and return amount ratio (total monetary amount of items returned versus the amount purchased).

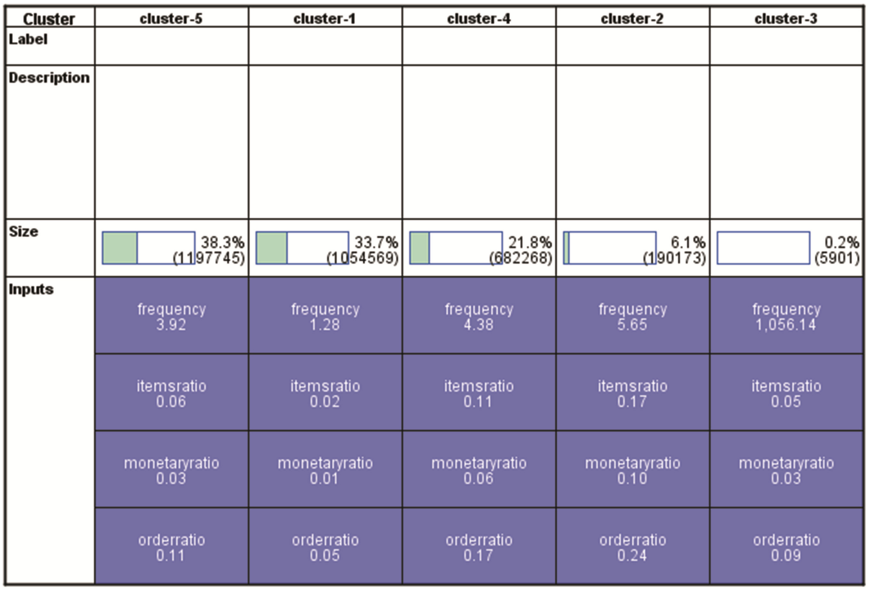
As shown in the Fig. 15, Q20 is on the threshold of Good (0.5) on the Silhouette scale.



Model Summary

| | |
|---|---|
| Algorithm | K-Means |
| Inputs | 4 |
| Clusters | 5 |

**Fig. 15.** Cluster Quality of Q20 K-Means Clustering: twenty iterations & five Clusters



| Cluster | cluster-5 | cluster-1 | cluster-4 | cluster-2 | cluster-3 |
|---|---|---|---|---|---|
| Label | | | | | |
| Description | | | | | |
| Size | 38.3% (1197745) | 33.7% (1054569) | 21.8% (682268) | 6.1% (190173) | 0.2% (5901) |
| Inputs | frequency 3.92 | frequency 1.28 | frequency 4.38 | frequency 5.65 | frequency 1,056.14 |
| | itemsratio 0.06 | itemsratio 0.02 | itemsratio 0.11 | itemsratio 0.17 | itemsratio 0.05 |
| | monetaryratio 0.03 | monetaryratio 0.01 | monetaryratio 0.06 | monetaryratio 0.10 | monetaryratio 0.03 |
| | orderratio 0.11 | orderratio 0.05 | orderratio 0.17 | orderratio 0.24 | orderratio 0.09 |

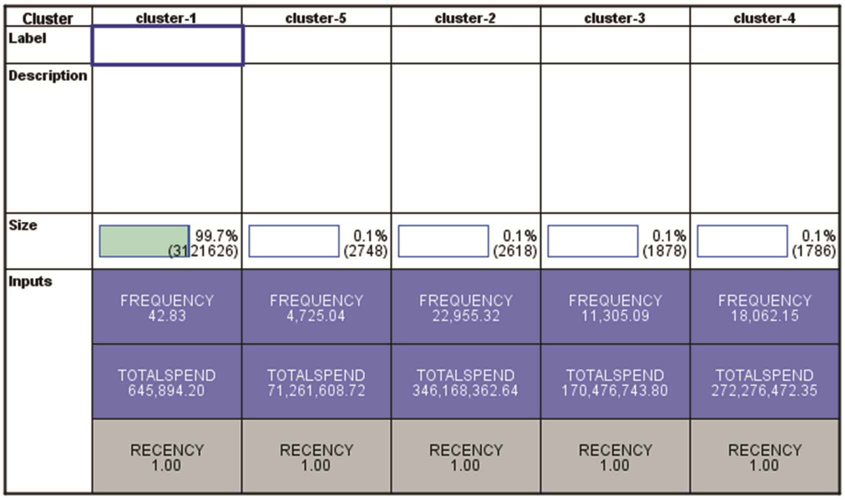
**Fig. 16.** Q20 Clusters ordered by size and predictor importance; Cluster size = 5

**Predictor Importance.** In context of clustering algorithms, predictor importance indicates how well the variable can differentiate different clusters. For both range (numeric) and discrete variables, the higher the importance measure, the less likely the variation for a variable between clusters is due to chance and more likely due to some underlying difference.

In Q20, all inputs have equal importance. The size of the largest cluster is 1.2 million (38% of population) while the size of smallest cluster is 5901 (0.2% of the population). The numbers beneath the feature names are the cluster centers (Fig. 16).

Q25 performs customer segmentation based on recency of last visit, frequency of visits and monetary amount spent. Recency doesn't have any impact on the clustering, while frequency of visits and amount spent are equally important in determining the cluster.

Although Q25 fares very good on the Silhouette scale, there is a skew in cluster size with 99.7% of population belonging to a single cluster (Fig. 17).



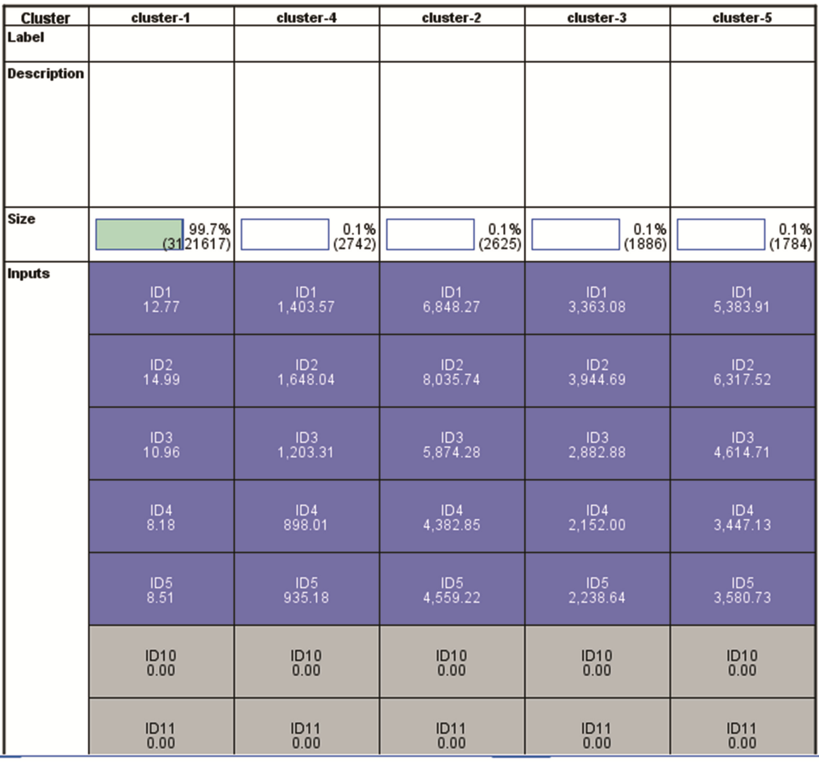
| Cluster | cluster-1 | cluster-5 | cluster-2 | cluster-3 | cluster-4 |
|---|---|---|---|---|---|
| Label | | | | | |
| Description | | | | | |
| Size | 99.7% (3121626) | 0.1% (2748) | 0.1% (2618) | 0.1% (1878) | 0.1% (1786) |
| Inputs | FREQUENCY 42.83 | FREQUENCY 4,725.04 | FREQUENCY 22,955.32 | FREQUENCY 11,305.09 | FREQUENCY 18,062.15 |
| | TOTALSPEND 645,894.20 | TOTALSPEND 71,261,608.72 | TOTALSPEND 346,168,362.64 | TOTALSPEND 170,476,743.80 | TOTALSPEND 272,276,472.35 |
| | RECENCY 1.00 | RECENCY 1.00 | RECENCY 1.00 | RECENCY 1.00 | RECENCY 1.00 |

**Fig. 17.** Q25 Clusters ordered by size and predictor importance; Cluster size = 5

Q26 clusters customers into book buddies/club groups based on their store book purchasing histories. For this query, we observe non-zero store sales values for books in 5 out of 15 categories. 10 categories show 0 sales and hence they do not have any impact on the clustering.

The clustering characteristics for Q26 are similar to Q25. Although it fares very well on Silhouette scale, 99.7% of the population belongs to a single cluster (Fig. 18).

In our performance experiments, we observe that K-Means runs for the default number of iterations, twenty, for convergence. Also, K-Means requires the number of clusters to be specified. Given these factors, the skew in the cluster size should not impact the extent to which K-Means is exercised. During our experiments, we found that the Spark cache size impacts the K-Means queries significantly and hence they were effective in assessing the benefits of an optimized data connector. K-Means runs for twenty iterations and if the data doesn't fit in the memory, the analytics engine has to repeatedly fetch the input vector from the data source. The optimizations done in the dashDB Spark data connector improved performance significantly.



| Cluster | cluster-1 | cluster-4 | cluster-2 | cluster-3 | cluster-5 |
|---|---|---|---|---|---|
| Label | | | | | |
| Description | | | | | |
| Size | 99.7% (3121617) | 0.1% (2742) | 0.1% (2625) | 0.1% (1886) | 0.1% (1784) |
| Inputs | ID1 12.77 | ID1 1,403.57 | ID1 6,848.27 | ID1 3,363.08 | ID1 5,383.91 |
| | ID2 14.99 | ID2 1,648.04 | ID2 8,035.74 | ID2 3,944.69 | ID2 6,317.52 |
| | ID3 10.96 | ID3 1,203.31 | ID3 5,874.28 | ID3 2,882.88 | ID3 4,614.71 |
| | ID4 8.18 | ID4 898.01 | ID4 4,382.85 | ID4 2,152.00 | ID4 3,447.13 |
| | ID5 8.51 | ID5 935.18 | ID5 4,559.22 | ID5 2,238.64 | ID5 3,580.73 |
| | ID10 0.00 | ID10 0.00 | ID10 0.00 | ID10 0.00 | ID10 0.00 |
| | ID11 0.00 | ID11 0.00 | ID11 0.00 | ID11 0.00 | ID11 0.00 |

**Fig. 18.** Q26 Clusters ordered by size and predictor importance; Cluster size = 5

## 5 Conclusion and Future Work

With the increasing importance of Machine Learning in big data scenarios, a broader coverage of commonly used Machine Learning algorithms along with more realistic scenarios like tuning the parameters of machine learning using cross validation and loading an existing model to predict outcome on real time data will make it even more appealing and relevant. Our experiments show that the performance characteristics of ML algorithms vary and hence will be useful in stressing the different system components. The paper highlights the importance of broadening the scope of ML algorithms and use cases in BigBench and provides concrete recommendations supported by experiments. It also talks about how the existing BigBench ML queries, K-Means and Logistics Regression, have been effective in proving the performance benefits of dashDB Spark connector.

In future, we plan to study the performance characteristics of other ML algorithms like Trees and Neural Networks on large data sets using BigBench. We'd also like BigBench to consider adopting our recommendations.

**Acknowledgement.** We would like to thank Berni Schiefer, Steve Rees, Torsten Steinbach, John Poelman and Manish Anand for providing their valuable feedback.

## References

1. Apache Spark. http://spark.apache.org/
2. dashDB. http://www.ibm.com/analytics/us/en/technology/cloud-data-services/dashdb/
3. dashDB Local. http://www.ibm.com/analytics/us/en/technology/cloud-data-services/dashdb-local/
4. UCI Machine Learning Repository. http://archive.ics.uci.edu/ml/
5. IBM SPSS. http://www.ibm.com/analytics/us/en/technology/spss/spss.html
6. ftp://public.dhe.ibm.com/software/analytics/spss/documentation/modeler/16.0/en/modeler_applications_guide_book.pdf
7. Ghazal, A., et al.: BigBench: towards an industry standard benchmark for big data analytics. In: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data. ACM (2013)
8. Chowdhury, B., Rabl, T., Saadatpanah, P., Du, J., Jacobsen, H.-A.: A BigBench implementation in the hadoop ecosystem. In: Rabl, T., Jacobsen, H.-A., Raghunath, N., Poess, M., Bhandarkar, M., Baru, C. (eds.) WBDB 2013. LNCS, vol. 8585, pp. 3–18. Springer, Heidelberg (2014). doi:10.1007/978-3-319-10596-3_1
9. Baru, C., et al.: Discussion of BigBench: a proposed industry standard performance benchmark for big data. In: Nambiar, R., Poess, M. (eds.) TPCTC 2014. LNCS, vol. 8904, pp. 44–63. Springer, Cham (2015). doi:10.1007/978-3-319-15350-6_4
10. Nambiar, R., Poess, M. (eds.): TPCTC 2013. LNCS, vol. 8391. Springer, Heidelberg (2014). doi:10.1007/978-3-319-04936-6
11. Meng, X., et al.: Mllib: Machine learning in apache spark. JMLR **17**(34), 1–7 (2016)
12. Agrawal, D., et al.: SparkBench – a spark performance testing suite. In: Nambiar, R., Poess, M. (eds.) TPCTC 2015. LNCS, vol. 9508, pp. 26–44. Springer, Heidelberg (2016). doi:10.1007/978-3-319-31409-9_3
13. Su, X., Khoshgoftaar, T.M.: A survey of collaborative filtering techniques. Adv. Artif. Intell. **2009**, 19 (2009). Article ID 421425, doi:10.1155/2009/421425
14. Koren, Y., Bell, R., Volinsky, C.: Matrix factorization techniques for recommender systems. Computer **42**(8), 30–37 (2009)
15. Zhou, Y., Wilkinson, D., Schreiber, R., Pan, R.: Large-scale parallel collaborative filtering for the netflix prize. In: Fleischer, R., Xu, J. (eds.) AAIM 2008. LNCS, vol. 5034, pp. 337–348. Springer, Heidelberg (2008). doi:10.1007/978-3-540-68880-8_32
16. Jain, P., Netrapalli, P., Sanghavi, S.: Low-rank matrix completion using alternating minimization. In: Proceedings of the Forty-Fifth Annual ACM Symposium on Theory of Computing. ACM (2013)
17. Transaction Processing Performance Council. http://www.tpc.org
18. Zaharia, M., Chowdhury, M., Das, T., Dave, A., Ma, J., McCauley, M., Franklin, M., Shenker, S., Stoica, I.: Resilient distributed datasets: a fault-tolerant abstraction for in-memory cluster computing. In: Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation. USENIX Association, p. 2 (2012)
19. Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.: Spark: cluster computing with working sets. In: Proceedings of the 2nd USENIX Conference on Hot Topics in Cloud Computing, Boston, 22–25 June 2010, p. 10 (2010)

20. Pilászy, I., Zibriczky, D., Tikk, D.: Fast als-based matrix factorization for explicit and implicit feedback datasets. In: Proceedings of the Fourth ACM Conference on Recommender Systems. ACM (2010)
21. Feuerverger, A., He, Y., Khatri, S.: Statistical significance of the Netflix challenge. Stat. Sci. **27**, 202–231 (2012)
22. Hastie, T., et al.: Matrix completion and low-rank SVD via fast alternating least squares. J. Mach. Learn. Res. **16**, 3367–3402 (2015)