Machine Learning-based Product Recommendation using Apache Spark

Lin Chen *, Rui Li *, Yige Liu *, Ruixuan Zhang *, Diane Myung-kyung Woodbridge
{lchen74,rli33,yliu225,rzhang45,dwoodbridge}@usfca.edu
Master of Science in Analytics (MSAN) Program,
University of San Francisco,
San Francisco, California

***Abstract*—There have been growing interests in the area of recommender systems using machine learning techniques. As there are a great number of explicit and implicit features that can be used for estimating user preference, it requires scalable and accurate algorithms along with a system with high availability and scalability. Alternating least square matrix (ALS) algorithm is an enhanced version of latent factor models using matrix factorization with good scalability and predictive accuracy. Apache Spark is an open-source distributed platform for processing big data, achieving good speed and scalability suitable for iterative machine learning algorithms. Amazon offers cloud computing services with various functionality including data storage and processing engines and is highly available and scalable. In this study, we applied the ALS algorithm using Apache Spark running on an Amazon Web Service (AWS) Elastic Map Reduce (EMR) cluster for recommending a product with a good accuracy and enhanced scalability.**

***Keywords*-collaborative learning, machine learning, Apache Spark, Amazon Web Service (AWS)**

I. Introduction

Recommender systems have become an emerging research area with the growth of e-commerce and web services including advertisements for providing personalized recommendations to a user. For example, Amazon, one of the largest online retailers in the world, has been using robust data-driven analytics to optimize its market growth for over 304 million active customers with 12.2 million products carried [1], [2]. When customers browse, buy and review products, the system provides them product recommendations to increase sales and improve customers experience based on their history.

For enhancing the outcome of a recommender system, many researchers have been developed recommender algorithms that utilize both explicit and implicit user feedbacks [3]. While explicit feedbacks include a user's product ratings and reviews, implicit feedbacks include purchase histories, click patterns, search histories, etc. However, as explicit feedbacks are not always available and are limited most of the time, inferring user preference using implicit feedbacks is inevitable. However, including implicit feedbacks increases the number of features to consider and requires both an algorithm and a system that are highly scalable.

Various collaborative filtering (CF) algorithms using user ratings of products to predict user preference have shown good predictive accuracy. Collaborative filtering and its variations have been optimized to work on big data sets, showing good scalability [4]. Recently, as implicit feedbacks became available and are commonly used in recommender systems, there have been many active ongoing research on more accurate and scalable algorithms for processing both explicit and implicit feedbacks [5][6].

Distributed computing is designed for running big data on a cluster of many machines to make computation reliable, fast and inexpensive. On distributed computing, a MapReduce job splits the data into smaller chunks into different partitions and a map task such as filtering and sorting jobs processes it in a parallel manner. The output of a map task becomes an input of a reduce operation which performs a summary operation. Spark extends the MapReduce model for efficient data sharing and combines structured data processing, graph algorithms and machine learning all in a single framework. Spark runs programs up to 100 times faster than Hadoop MapReduce and provides easy development environment [7].

This study was designed to develop a scalable recommender system using the alternative least square matrix algorithm which is a collaborative filtering algorithm and Apache Spark, a scalable cluster computing framework. In our study, the Apache Spark was running on Amazon Web Service (AWS) Elastic Map Reduce (EMR) and the data was uploaded from AWS Simple Storage Service (S3), a durable and scalable data storage, onto the EMR cluster. EMR provides a Hadoop framework for processing vast amounts of data easy, fast, and cost-effective across dynamically scalable Amazon Elastic Computing Cloud (EC2) instances and can interact with data in AWS S3. For enhancing computation performance, we performed various levels of configurations in parallelism, driver and executor memory. In our study, we utilized Amazon book review data to validate its performance and scalability.

* Theses authors contributed equally.

978-1-5386-0435-9/17/$31.00 ©2017 IEEE

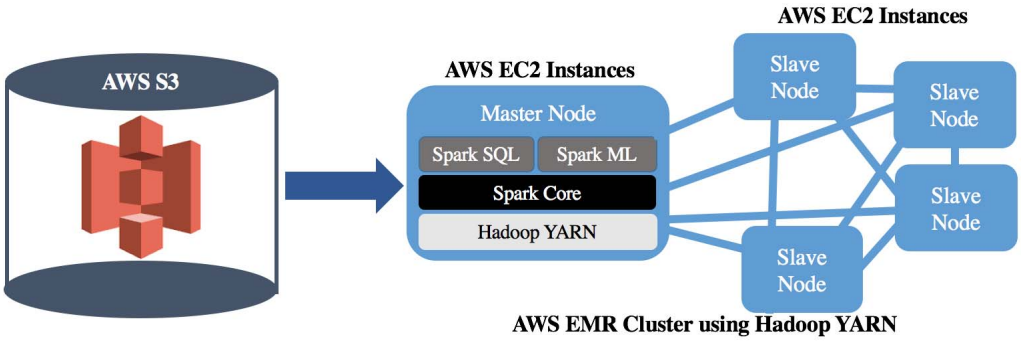

Figure 1: System workflow

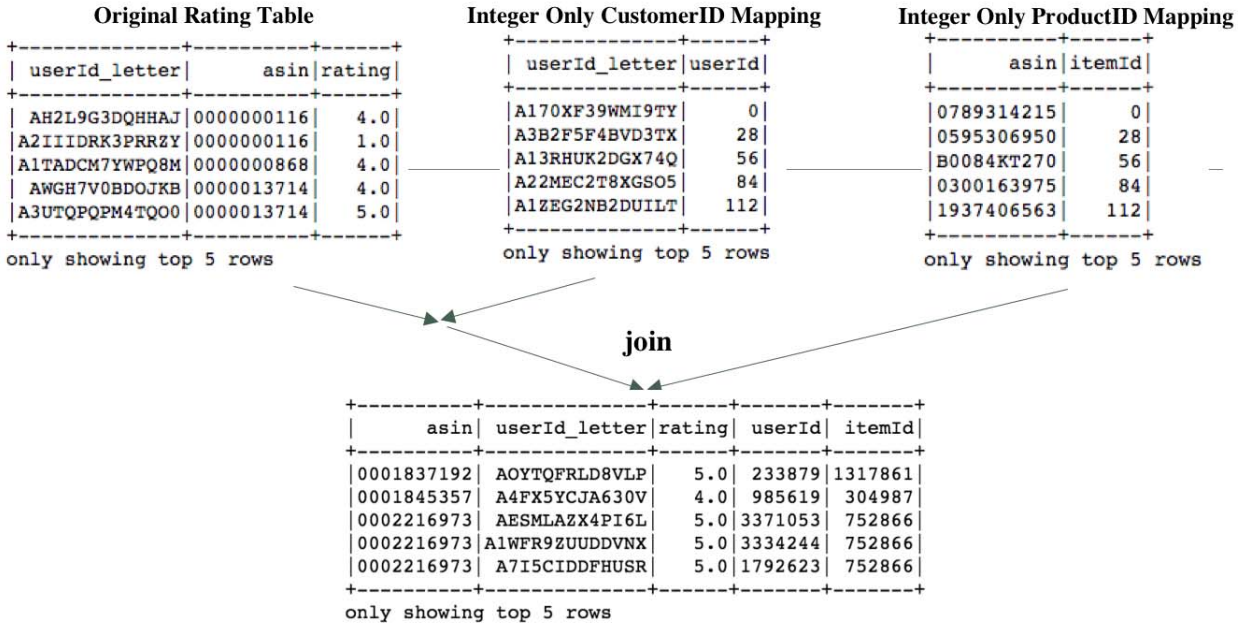

Figure 2: Joined DataFrame with newly generated unique IDs and ratings.

II. System Overview

A. System Workflow

In this study, we developed a pipeline to load and process data in a cluster (Figure 1). AWS provides high availability and scalability and makes its components including storage and processing engines to be compatible. AWS S3 is a durable and scalable object storage system that could be automatically extendable, if it requires more data storage. AWS EC2 is a cloud server and provides auto scaling features that scales up and down by increasing and decreasing EC2 instances based on demands to maintain high throughput.

In our study, data is stored in an S3 bucket which we can easily load data from and transfer to an AWS EMR cluster publicly or using user authentication. Apache Spark is installed on AWS EC2 nodes managed by the EMR cluster using a Hadoop YARN (Yet Another Resource Negotiator) cluster manager. YARN is a Hadoops resource manager and execution system and can run other types of jobs other than MapReduce including Spark. YARN consists of a resource manager and multiple node managers along with a client process, a driver and executors [8]. In the YARN cluster, users can manage memory properties including driver and executor memory. The master runs the driver which requests memory and CPU resources from the cluster manager and sends tasks to executors. The executor runs tasks in parallel and its memory property affects the amount of data Spark can cache.

In our study, the data process pipeline was built with six steps: 1) Data transfer, 2) RDD creation, 3) DataFrame creation, 4) DataFrame processing, 5) recommender

algorithm training and 6) prediction.
Spark Core is a foundation of the system and provides basic Spark components including resilient distributed datasets (RDDs) and functions. RDD is an abstraction of a distributed collection of data with operations applicable to the data and provides networking, security, scheduling and data shuffling functions. In this work, we created RDDs from the data in the S3 bucket and used it for DataFrame conversion.
Spark SQL provides functions for manipulating large sets of distributed and structured data including DataFrames and DataSets, where a DataSet is a distributed collection of data and a DataFrame is a subset of a DataSet. A DataFrame includes columns and values and is similar to a table in a relational database management system (RDBMS). Spark translates them to operations on RDDs and executes as usual Spark jobs [7][9]. Spark SQL uses SQL-like functionalities and query languages. In our system, each of the RDDs is piped to DataFrame generation using SparkSQL and DataFrame schema definition. Then the DataFrames are joined as one DataFrame using their keys. An example of a joined DataFrame is depicted in Figure 2.
For training recommender model, we utilized the alternating least square matrix model (ALS). The algorithm details are described in Section II-B. We split the data into training and testing sets. Training data is for building a model, while testing data is for evaluating the accuracy of the model. For testing, to make a recommendation for each customer, we returned the predicted ratings in descending order and provided the corresponding item details. We used root mean square error to evaluate the model performance and tuned parameters through a grid search.

B. Algorithm

In order to recommend a product, we applied the Alternating Least Squares (ALS) matrix factorization method [5], [6], [10], which is an advanced version of collaborative filtering method. Collaborative filtering (CF) is a recommendation method based on user behavior similarity. CF provides a recommendation to that user or others who have similar tastes by measuring r_{ui}, which is an observation of item i from user u such as rating and implicit user behaviors. r_{ui} could be explicit ratings including reviews or implicit ratings such as browsing or purchase frequencies. Using k items rated by u that are most similar to i and similarity measure s_{ij} between item i and j, item-oriented CF calculates a predicted rating $\hat{r}_{ij}$.

$$\hat{r}_{ij} = \frac{\sum_{j \in S^k(i;u)} s_{ij} r_{uj}}{\sum_{j \in S^k(i;u)} s_{ij}} \tag{1}$$

Item-oriented CF models do not distinguish between explicit user preference and implicit information.
As an explicit review is not always available, a recommendation algorithm also needs to infer user preference based on implicit user behaviors. For instance, if a user tends to browse books from a same author often, we can assume that the user likes the author. A latent factor (LF) model is developed for uncovering these latent features that explains observed ratings. As an example, a model could be induced by singular vector decomposition (SVD) of a user u with a user-factor vector $x_u \in R$ and item i with a item-factors vector $y_i \in R$ given in Equation 2.

$$min_{x,y} \sum (r_{ui} - x_u^T y_i)^2 + \lambda(\|x_u\|^2 + \|y_i\|^2) \tag{2}$$

where λ is a regularization factor and parameters are learned by stochastic gradient descent (SGD). LF model shows better accuracy and scalability than conventional CF models.
The Alternating least square (ALS) matrix factorization algorithm is an improved version which applies both explicit and implicit user feedback and denotes the strengths in observations of implicit feedback. The ALS method uses binary variable of the preference of user u to item i, p_{ui}, where

$$p_{ui} = \begin{cases} 1 & r_{ui} > 0 \\ 0 & r_{ui} = 0 \end{cases} \tag{3}$$

Equation 3 means that if a user u rates or browses item i ($r_{ui} > 0$), this means that u likes i. Otherwise, there is no preference. For $p_{ui} = 1$, we can calculate confidence level c_{ui}, where c_{ui} increases according to r_{ui}.

$$c_{ui} = 1 + \alpha r_{ui} \tag{4}$$

where α is a constant and Hu's experiments demonstrate good results with $\alpha = 40$ [5].
Using the preference and confidence matrices, the algorithm finds latent factors of a user's preference to an item factored by x_u and y_i and recommends items. User and item factors are obtained from the objective function in Equation 5.

$$min_{x,y} \sum c_{ui}(p_{ui} - x_u^T y_i)^2 + \lambda(\sum_u \|x_u\|^2 + \sum_i \|y_i\|^2) \tag{5}$$

where λ is data-dependent and determined by cross-validation. All possible u and i pairs should be calculated for unobserved data and the possible combinations of users and items could be easily over a few billion. The ALS matrix algorithm applies an efficient optimization process by fixing either user factors (x_u) or item factors (y_i) and recomputing the other factor alternately and achieves linear computing time.

Data Set	Input Format	Size
Book Rating	.csv	22,507,155 ratings (873.8 MB)
Product Review	.json	8,898,041 reviews (9.8 GB)

Table I: Amazon Book Review and Rating Data

```
AH2L9G3DQHHAJ,0000000116,4.0,1019865600
A2IIIDRK3PRRZY,0000000116,1.0,1395619200
A1TADCM7YWPQ8M,0000000868,4.0,1031702400
AWGH7V0BDOJKB,0000013714,4.0,1383177600
A3UTQPQPM4TQO0,0000013714,5.0,1374883200
A8ZS0I5L5V31B,0000013714,5.0,1393632000
ACNGUPJ3A3TM9,0000013714,4.0,1386028800
A3BED5QFJWK88M,0000013714,4.0,1350345600
A2SUAM1J3GNN3B,0000013714,5.0,1252800000
APOZ15IEYQRRR,0000013714,5.0,1362787200
AYEDW3BFK53XK,0000013714,5.0,1325462400
A1KLCGLCXYP1U1,0000013714,3.0,1376092800
A37W6POFWIVG13,0000013714,5.0,1316131200
A2EIPZNHAEXZHJ,0000013714,4.0,1325030400
A1VAFVJFT58YI3,0000013714,5.0,1384300800
A9KTKY6BUR8U6,0000013714,1.0,1357516800
A2742OG8PK8KU6,0000013714,5.0,1358380800
A38AAPXSJN4C5G,0000015393,4.0,1239494400
A14A5O8VJK5NLR.0000029831.4.0.1393286400
```

(a) Example: User rating data.

```
{ "itemID": "0000031852",
  "title": "Girls Ballet Tutu Zebra Hot Pink",
  "price": 3.17,
  "Url": "http://ecx.images-amazon.com/images/I/51fAmVkTbyL._SY300_.jpg",
  "salesRank": {"Toys & Games": 211836},
  "brand": "Coxlures",
  "categories": [["Sports & Outdoors", "Other Sports", "Dance"]],
  "related": {
             "also_bought": ["B00JHONN1S", "B002BZX8Z6"],
             "bought_together": ["B002BZX8Z6"]
             }
}
```

(b) Example: Product review data.

Figure 3: Example input data.

userId_letter	title	rating
AM3YQ21A0JZSH	Tell Me	5.0
AM3YQ21A0JZSH	Oxford Blood (The...	5.0
AM3YQ21A0JZSH	null	5.0
AM3YQ21A0JZSH	Darkmoon (The Cai...	5.0
AM3YQ21A0JZSH	To Katie With Love	5.0
AM3YQ21A0JZSH	The Kings of Char...	5.0
AM3YQ21A0JZSH	null	5.0
AM3YQ21A0JZSH	Dreaming of the W...	5.0
AM3YQ21A0JZSH	Descent of Blood ...	5.0
AM3YQ21A0JZSH	The Silver Sphere	5.0
AM3YQ21A0JZSH	Side Jobs: Storie...	5.0
AM3YQ21A0JZSH	null	5.0
AM3YQ21A0JZSH	Vanilla On Top (W...	5.0
AM3YQ21A0JZSH	After Dark: The D...	5.0
AM3YQ21A0JZSH	Out of the Shadow...	5.0
AM3YQ21A0JZSH	Hot Secrets (Tall...	5.0
AM3YQ21A0JZSH	To Catch a Vampir...	5.0

(a) Training data set.

userId_letter	title	prediction
AM3YQ21A0JZSH	Night Walker (The...	4.9055037
AM3YQ21A0JZSH	Hiding in the Sha...	4.7850604
AM3YQ21A0JZSH	Mind Over Monster...	4.5004263
AM3YQ21A0JZSH	Touching Evil: A ...	4.318152
AM3YQ21A0JZSH	null	4.1369963

(b) Test data set.

Figure 4: Example : Subset of training and test data from a user who likes vampire stories.

III. Experiment Output

The purpose of the experiment is to use customers' previous ratings (r) of products to calculate predicted ratings ($\hat{r}$) of other products. The system provides recommendations to a user based on the highest predicted ratings using ALS developed on the Apache Spark and AWS frameworks. We chose Amazon review data to train a model and provide personalized recommendations. We used PySpark (version 2.1.0) on EMR (version 5.2.1) and the Spark Python API to process our data and applied the ALS algorithm to build a recommendation system.

A. data

For training and testing a recommender model, we utilized Amazon book reviews and product review metadata collected between May 1996 and July 2014 (Table I) by McAuley [11][12].

The book rating data includes a user ID, item ID, timestamp and rating without any missing values in the dataset or any outliers in the rating column. The user ID and item ID are both strings composed of numbers and letters, while the ratings are categorical integer values between 1 and 5. The number of unique User ID in the dataset is 8,026,324. For recommending items and their details to a user, we mapped the item ID with item names using product review metadata which includes the information of over 9.40 million Amazon products. The product review metadata contains an item ID, item name, price, product URL, items others also bought together, etc. (Figure 3).

B. Results

1) Recommender System: The product ratings (.csv) in Table I were loaded as RDD objects with the number of partitions equivalent to 3 times of the number of cores. The product review RDDs from the .json file in Table I are in a key-value pair format and its key is also contained in the first RDD. The RDDs were built for the later DataFrame conversion. The book rating DataFrame contains the information for both building the ALS model and locating detailed information of the recommended item. Once a

Running on / Performance	Local machine (4 cores, 16 GB RAM)	YARN x 5 Instances (32 cores, 244 GB)	YARN x 1 Instance (32 cores, 244 GB)	YARN x 1 Instance (8 cores, 61GB)
Running Time /model	30 mins (iter=20)	10 mins (iter=20)	10 mins (iter=20)	15 mins (iter=20)
Cost	0	$2.66 / model	$0.6 / model	$0.2 / model
RMSE	1.5	1.5	1.5	4.3

Table II: Performance comparison based on machine types.

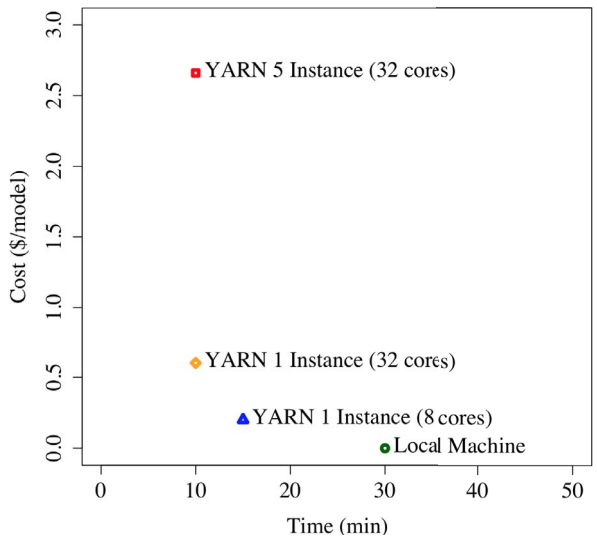

Figure 5: Cost comparison based on machine types.

recommendation is available from the model, the predicted item ID was used to retrieve detailed information including book titles from the product review DataFrame.

ALS in Spark ML was applied to build the recommendation engine [10]. ALS learns a set of latent factors for users and items and generate predicted ratings, aiming at minimizing the error between actual ratings and predicted ratings. The final output of the model is a matrix of predicted ratings, formed as a user and item ID combination.

We used root mean squared error (RMSE) to evaluate the model performance. RMSE measures an average distance between the predicted rating and the actual rating.

$$RMSE = \sqrt{\frac{\sum (r_{ui} - \hat{r}_{ui})^2}{k}} \tag{6}$$

The achieved RMSE value using the experiment data with different configurations is described in Table II.

The parameters associated with the model are as followed: $rank = 30$, $maxIter = 20$, $regParam = 0.1$. $rank$ is the number of latent variables in the model. As the algorithm generally converges to a promising solution within 20 iterations, we set $maxIter = 20$. $regParam$ specifies the regularization parameter.

Finally, for each User ID, we ranked the predicted ratings for all the items in descending order, and we only recommended top-rated items.

For example, the user AM3YQ21A0JZSH, who tends to give higher ratings on vampire-related books, received book recommendations in Figure 4. We can see that these books are also vampire mysteries.

2) Performance: Based on the trials of different settings on EMR, we found that the selection of cluster size and number of nodes should be based on the characteristics of a dataset and specific to a utilized algorithm. Increasing the numbers of nodes on AWS EMR might require more time due to the cross-node connections and network traffics. The number of cores and the RAM size of each cluster should also refer to the requirement of the algorithm and the size of the data.

- Size: The cluster should have enough driver and execution memory to process the data.
- Algorithm: The cluster should have the ability to process data with a required number of iterations.

As to this practice, as long as one machine can process the program well, using just one machine is optimal for cost and time as it has less data transfer. If building a better model (one with a lower RMSE for this case) is the goal, we recommend choosing a machine that is capable of running the algorithm. If cost is a concern, choosing a cluster with multiple cheap machines might be also the choice. As AWS provides various server options with different CPU, memory, storage, and networking capacity, a developer should choose one that is suitable for her or his own needs.

We compared performance and price with different options in Table II and Figure 5. The local machine used for this study was a MacBook Pro with 16 GB memory, quad-core Intel Core i7 at 2.5 GHz using a Spark standalone cluster manager.

Fig 6 shows the system performance, as the size of dataset changes. The experiment was done on a single YARN instance with 32 cores.

There are two types of memory setting we considered for PySpark configuration: execution memory and storage

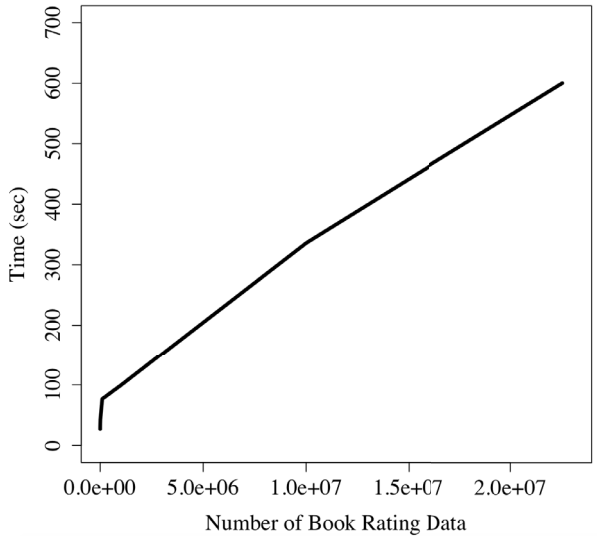

Figure 6: System performance with different data sizes.

memory. Since the algorithm for our model requires multiple iterations, if the storage memory is too small, there would not be enough space to cache the training data which was repeatedly used for modeling. On the other side, if the execution memory is not sufficient, the task might fail for the reason of a container killed by YARN for exceeding memory limits. This scenario could be prevented by setting the spark.yarn.executor.memoryOverhead configuration when submitting a Spark application.
One example of configuration for submitting an application is shown below.

```
–master yarn
–deploy-mode client
–driver-memory 1024M
–executor-memory 2048M
–num-executors 2
–conf spark.yarn.executor.memoryOverhead=512
```

Setting the memory size and memoryOverhead for an executor is an important step when submitting an application to clusters. Configurations for applications running on a cluster might take a few times to make sure to make the most use of the machines.

IV. Conclusion

In this work, we developed a product recommender system using Apache Spark and its ALS algorithm. In order to enhance scalability of the algorithm, we developed the system on the Amazon Web Service (AWS) framework. We also compared its performance using different settings including Spark standalone and Hadoop YARN cluster managers with different configurations. The experiment results showed a decent root mean square error as an output of the recommender model with efficient run time. The experiment also showed that running on a YARN cluster outperforms even with reasonable cost for the dataset we utilized.

References

[1] Statistica, *Statistics and Facts about Amazon*. [Online]. Available: https://www.statista.com/topics/846/amazon/

[2] 360pi, *How Many Products Does Amazon Actually Carry? And in What Categories?* [Online]. Available: http://360pi.com/press_release/many-products-amazon-actually-carry-categories/

[3] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013.

[4] G. Takács, I. Pilászy, B. Németh, and D. Tikk, "Scalable collaborative filtering approaches for large recommender systems," *Journal of machine learning research*, vol. 10, no. Mar, pp. 623–656, 2009.

[5] Y. Hu, Y. Koren, and C. Volinsky, "Collaborative filtering for implicit feedback datasets," in *Data Mining, 2008. ICDM'08. Eighth IEEE International Conference on*. Ieee, 2008, pp. 263–272.

[6] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems," *Computer*, vol. 42, no. 8, 2009.

[7] A. Spark, "Apache spark: Lightning-fast cluster computing," 2016. [Online]. Available: http://spark.apache.org

[8] M. B. Petar Zecevic, *Spark in Action*. Manning Publications, 2016.

[9] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi *et al.*, "Spark sql: Relational data processing in spark," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 1383–1394.

[10] Apache Spark, *Collaborative Filtering - spark.ml*. [Online]. Available: https://spark.apache.org/docs/2.0.0-preview/ml-collaborative-filtering.html

[11] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel, "Image-based recommendations on styles and substitutes," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pp. 43–52.

[12] J. McAuley, R. Pandey, and J. Leskovec, "Inferring networks of substitutable and complementary products," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2015, pp. 785–794.