NNMF IN GOOGLE TENSORFLOW AND APACHE SPARK: A COMPARISON STUDY

A Thesis

Presented to

The Faculty of the Department of Computer Science

Sam Houston State University

In Partial Fulfillment

of the Requirements for the Degree of

Master of Science

---

by

Qizhao Li

August, 2019

NNMF IN GOOGLE TENSORFLOW AND APACHE SPARK: A COMPARISON STUDY

by

Qizhao Li

APPROVED:

Hyuk Cho, PhD
Thesis Director

Bing Zhou, PhD
Committee Member

Qingzhong Liu, PhD
Committee Member

John Pascarella, PhD
Dean, College of Science and Engineering Technology

ABSTRACT

Li, Qizhao , *NNMF IN GOOGLE TENSORFLOW AND APACHE SPARK: A COMPARISON STUDY*. Master of Science (Computer Science), August, 2019, Sam Houston State University, Huntsville, Texas.

Data mining is no longer a new term as it has been already pervasive in all aspects of our lives. New computing platforms for specific usages are proposed continuously. Therefore, the awareness of the characteristics and the capacity of existing and newly proposed platforms becomes a critical task for researchers and practitioners, who want to use existing algorithms and also develop new ones on the recent platforms.

Particularly, this thesis aims to implement and compare a set of popular matrix factorization algorithms on recent computing platforms. Specifically, the three matrix factorization algorithms, including classic Non-negative Matrix Factorization (NNMF), CUR Matrix Decomposition, and Compact Matrix Decomposition (CMD), are implemented on the two computing platforms, including Apache Spark and Google TensorFlow.

As rank k approximation with Singular Value Decomposition (SVD) is an optimal baseline, both CUR and CMD approximation are less accurate than the SVD approximation. The experimental result shows that CMD in TensorFlow performs better in terms of matrix approximation than the other two non-negative matrix factorization algorithms (NNMF, and CUR) in the same experiment setup. Also, as the number of rows or columns selected for CUR and CMD increases, the approximation error decreases.

KEY WORDS: TensorFlow, Apache Spark, Non-Negative Matrix Factorization, CUR Matrix Decomposition, Compact Matrix Decomposition, Approximation Performance

TABLE OF CONTENTS

Page

ABSTRACT iii

TABLE OF CONTENTS iv

LIST OF TABLES vi

LIST OF FIGURES vii

I INTRODUCTION 1

II BACKGROUND 3

Singular Value Decomposition (SVD) 3

Non-negative Matrix Factorization (NNMF) 4

CUR Matrix Decomposition 6

Compact Matrix Decomposition (CMD) 9

Apache Spark 11

Google TensorFlow 13

III METHODOLOGY 16

Algorithms Implementation and Evaluation 16

Computing Platforms 18

Google Colab 19

IV EXPERIMENT RESULT 20

Experimental Setting 20

Approximation Performance within Platform 21

Approximation Performance between Platforms 23

Overall Approximation Performance 26

Running Time Performance 27
Approximation Performance over Varying k 28
Overall Approximation Performance over Varying k 30
V CONCLUSION & FUTURE WORK 31
Conclusion 31
Future Work 31
REFERENCES 33
APPENDIX 38
VITA 46

LIST OF TABLES

Table **Page**

1 Multiplicative Update Rules [1].. 5

2 Initial Subspace Construction [7].. 8

3 CMD Subspace Construction [7]. 11

LIST OF FIGURES

Figure **Page**

1 Rank k Approximation via SVD [43]. 3
2 CUR Decomposition [31]. 8
3 Performance Comparison of Three Algorithms in Space, Time and Estimation Cost [7]. 10
4 Illustration of CMD Steps [32]. 10
5 Apache Spark Construction [39]. 12
6 TensorFlow Toolkit [38]. 14
7 Online Job Listing [28]. 15
8 New GitHub Activity [28]. 15
9 Illustration of (a) CUR and (b) CMD [7]. 16
10 Comparison in Apache Spark. 21
11 Comparison in Google TensorFlow. 22
12 Comparison of NNMF between TensorFlow and Spark. 23
13 Comparison of CUR between TensorFlow and Spark. 24
14 Comparison of CMD between TensorFlow and Spark. 25
15 Approximation Performance Comparison. 26
16 Running Time Performance Comparison. 27
17 Approximation Performance over Varying k in Spark. 28
18 Approximation Performance over Varying k in TensorFlow. 29
19 Approximation Performance over Varying k. 30

CHAPTER I

Introduction

Data mining is no longer a new term as it has already been implemented and used in every aspect of our daily activities. New computing platforms for specific and general propose are being proposed continuously to achieve effective and efficient computation. Some disappeared after months, but others have been further developed more comprehensively. Therefore, the awareness of the characteristics and capacity of those existing platforms becomes a critical task for researchers and practitioners, who want to use existing algorithms or develop new one on the state-of-the-art computing platforms.

Among numerous computing algorithms, we focus on the three matrix factorization algorithms: Non-Negative Matrix factorization [1], CUR Matrix Decompositions [6], and Compact Matrix Decomposition [7]. We implement the three algorithms on the two recently developed computing platforms: Apache Spark [3] and Google TensorFlow [2]. Apache Spark and Google TensorFlow were released on May 2014 and November 2015, respectively. Both have been popular in the data mining, machine learning, and data science communities and made a great contribution to both research and real-life applications. For example, NNMF is widely used in astronomy, text mining, spectral data analysis, bioinformatics, and nuclear imaging, etc. Uber uses Spark Streaming, Kafka, and HDFS (Hadoop Distributed File System) for building a continuous ETL (Extract Transform Load) pipeline [41]. GE Healthcare trained a neural network using TensorFlow to identify anatomy on MRIs of the brain [40].

The remaining chapters are organized as follow. Chapter II briefly discusses fundamentals for the matrix factorization algorithms and the two state-of-the-art

computing platforms. Chapter III describes the algorithm implementation in Google Colab environment. Chapter IV discusses the experimental setting and the matrix approximation result. Chapter V concludes with future research direction.

CHAPTER II

Background

Singular Value Decomposition (SVD)

Singular Value Decomposition (SVD) [42] is used as the standard optimal baseline matrix factorization in the study. SVD is a factorization of a matrix in linear algebra. It is the generalization of the eigendecomposition of a positive semidefinite normal matrix.

Suppose A is an m x n matrix, which is the field of either real numbers or complex numbers. The SVD of A exists and is a factorization of the form:

$$A = USV^T, \quad (1)$$

where the columns of U are the left singular vectors, each of which is orthonormal, S has singular values in its diagonal entries, and V^T has rows that are the right singular vectors. SVD is largely used in the mathematical area such as pseudo inverse, solving homogeneous linear equations, and separable models. The SVD can be used for both rectangular and square matrices.

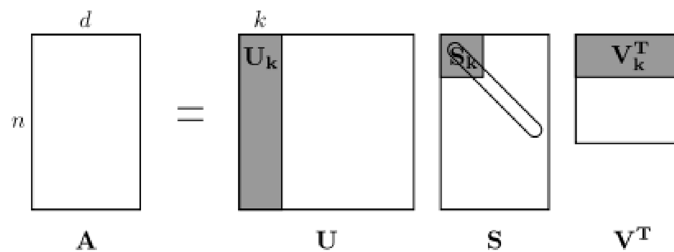

Figure 1. **Rank k Approximation via SVD [43].**

In order to have best approximate a matrix *A* by a rank k matrix, the SVD gives a rigorously justified solution. As shown in Figure 1, the *S* is non-zero only on its diagonal and the diagonal entries of *S* are sorted from high to low. The rank k approximation is $A_k = U_k S_k V_k^T$.

Today, singular value decomposition has spread through many branches of science, in particular psychology and sociology, climate and atmospheric science, and astronomy. It is also extremely useful in machine learning and in both descriptive and predictive statistics.

Non-negative Matrix Factorization (NNMF)

In 1999, Lee and Seung introduced a new algorithm for machine learning: Non-Negative Matrix Factorization(NNMF) [5]. For a given non-negative matrix *V*, NNMF targets to find non-negative matrix factors, *W* and *H*, such that:

$$V \approx WH \qquad (2)$$

Given a set of multivariate m-dimensional data vectors, the vectors are placed in the columns of an *m* x *n* matrix *V*, where *n* is the number of examples in the data set. The given matrix V is then factorized into an *m* x *r* matrix *W* and an *r* x *n* matrix *H*. Usually *r* is chosen to be smaller than *n* or *m* so that *W* and *H* are smaller than the original matrix *V* [1]. NNMF aims at minimizing the Euclidian distance between *V* and *WH* and can be used as an effective technique for dimension reduction and unsupervised clustering [12].

Table 1

Multiplicative Update Rules [1].

$H_{\alpha\mu} \leftarrow H_{\alpha\mu} \frac{(W^T V)_{\alpha\mu}}{(W^T WH)_{\alpha\mu}}$	$W_{i\alpha} \leftarrow W_{i\alpha} \frac{(VH^T)_{i\alpha}}{(H^T HW)_{i\alpha}}$
$H_{\alpha\mu} \leftarrow H_{\alpha\mu} \frac{\sum i\, W_{i\alpha} V_{i\mu} / (WH)_{i\mu}}{\sum k\, W_{k\alpha}}$	$W_{i\alpha} \leftarrow W_{i\alpha} \frac{\sum \mu\, H_{\alpha\mu} V_{i\mu} / (WH)_{i\mu}}{\sum v\, WH_{\alpha v}}$

As shown in Table 1, NNMF consists of the two multiplicative update rules for updating *H* and *W*. The Euclidean distance is invariant under these updates if and only if *W* and *H* are at a stationary point of the distance [1].

Compared with SVD, the NNMF factors only positive entries. SVD factors can be related to eigenfunction of a system when original matrix represented a system about which one is interested in terms of signal processing perspective. NNMF can also be used but the associating physical relationships are more indirect. Moreover, the SVD yields unique factors whereas NNMF factors are non-unique, which makes NNMF more suitable for the privacy protection algorithms.

NNMF has also been implemented and used in numerous applications. In biological data mining, CloudNMF [12] and bioNMF [13] are developed with NNMF. In text mining, a document-term matrix is constructed with the weights of various terms from a set of documents. The matrix is factored into a term-feature and a feature-document matrix. The features are derived from the contents of the documents, and the feature-document matrix describes data clusters of related documents [30]. In astronomy,

NNMF is a promising method for dimensional reduction in the sense that astrophysical signals are non-negative. NNMF has been applied to the spectroscopic observations [34] and the direct imaging observations as a method to study the common properties of astronomical objects and post-process the astronomical observations. In direct imaging, various statistical methods have been adopted to reveal the faint exoplanets and circumstellar disks from bright the surrounding stellar lights, which has a typical contrast from 10^5 to 10^{10} [35]. However, the light from the exoplanets or circumstellar disks are usually over-fitted, where forward modeling has to be adopted to recover the true flux. Forward modeling is currently optimized for point sources, not for extended sources, especially for irregularly shaped structures such as circumstellar disks. In this situation, NNMF has been an excellent method, being less over-fitting in the sense of the non-negativity and sparsity of the NNMF modeling coefficients. Therefore, forward modeling can be performed with a few scaling factors, rather than a computationally intensive data re-reduction on generated models.

The NNMF on TensorFlow is implemented by eesungkim and available on GitHub [11].

CUR Matrix Decomposition

CUR Matrix Decomposition [6] was proposed by M. Mahoney and P. Drineas on January 12, 2009. CUR matrix decomposition is a low-rank matrix decomposition, which is explicitly expressed in terms of a small number of actual columns and actual rows of the data matrix.

The algorithm preferentially chooses actual columns and rows that exhibit high "statistical leverage." Thus, CUR in a very precise statistical sense, exerts a disproportionately large effect on the best low-rank fit of the data matrix [6].

$$A \approx CUR \tag{3}$$

$$A \approx C \cdot U \cdot R$$

$$A \approx C \cdot U \cdot R$$

A C U R

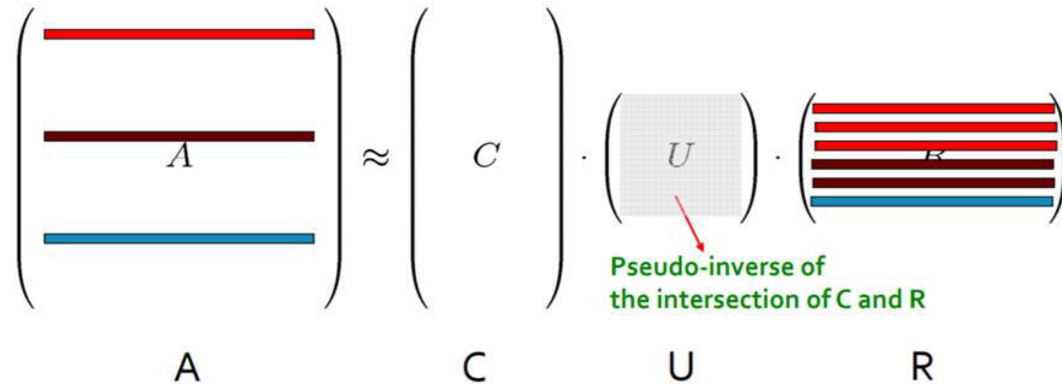

***Figure 2*. CUR Decomposition [31].**

As illustrated in Figure 2, given an *m* x *n* matrix *A*, CUR algorithm decomposes it as a product of three matrixes, *C*, *U*, and *R*, where *C* consists of a small number of actual columns of *A*, *R* consists of a small number of actual rows of *A*, and *U* is a small carefully constructed matrix that guarantees that the product *CUR* is "close" to the original matrix *A* [6]. As described in Table 2, the algorithm carefully selects columns and rows and constructs the subspace of *C* and *R*.

Table 2

Initial Subspace Construction [7].

Input: matrix $A \in \mathbb{R}^{m \text{ x } n}$, sample size *c*
Output: $C_d \in \mathbb{R}^{m \text{ x } n}$
1. for $x = 1 : n$ [column distribution]
2. $P(x) = \sum_i A(i,x)^2 / \sum_{i,j} A(i,j)^2$
3. for $i = 1 : c$ [sample columns]
4. Pick $j \in 1$: n based on distribution *P(x)*
5. Compute $C_d(:,i) = A(:,j)/\sqrt{cP(J)}$

Compared with SVD, CUR decomposition is less accurate in terms of low-rank approximation. However, it is unique in terms of constructing the factors as it takes the

actual columns and rows from a given non-negative matrix and constructs the corresponding non-negative factor matrices.

For a specific application of CUR, Intelligent Transportation Systems (ITSs) often operate on large road networks, and typically collect traffic data with high temporal resolution. Consequently, ITSs need to handle massive volumes of data, and methods to represent that data in more compact representations are sorely needed [37]. CUR matrix decomposition can lead to low-dimensional models where the components correspond to individual links in the network and thus the resulting models can be easily interpreted and can also be used for compressed sensing of the traffic network [37].

Compact Matrix Decomposition (CMD)

Compact Matrix Decomposition (CMD) [7] requires much less space and computation time then CUR; thus, CMD is more efficient. Figure 3 illustrates the performance comparisons among SVD, CUR, and CDM regrading to space, time, and estimation cost. As shown in Figure 3, CMD has significantly outperformed CUR decomposition and the singular value decomposition (SVD) of space requirement and computational time.

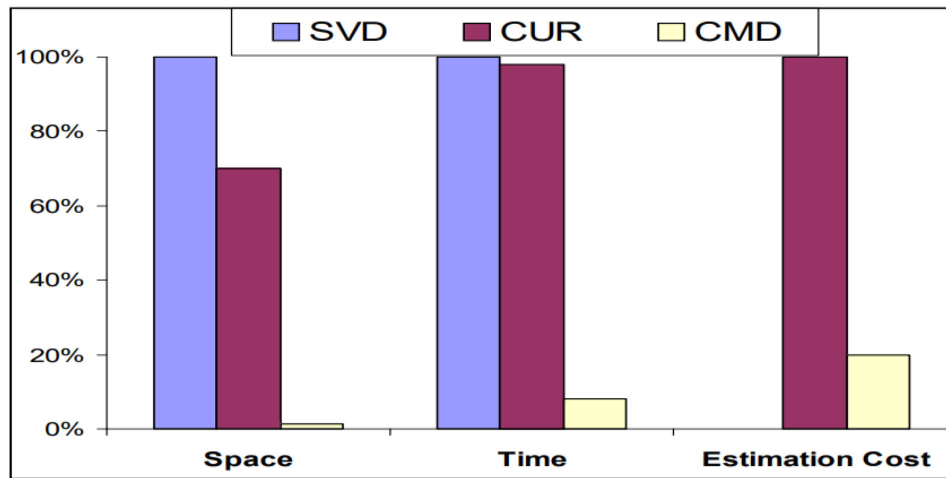

Figure 3. **Performance Comparison of Three Algorithms in Space, Time and Estimation Cost [7].**

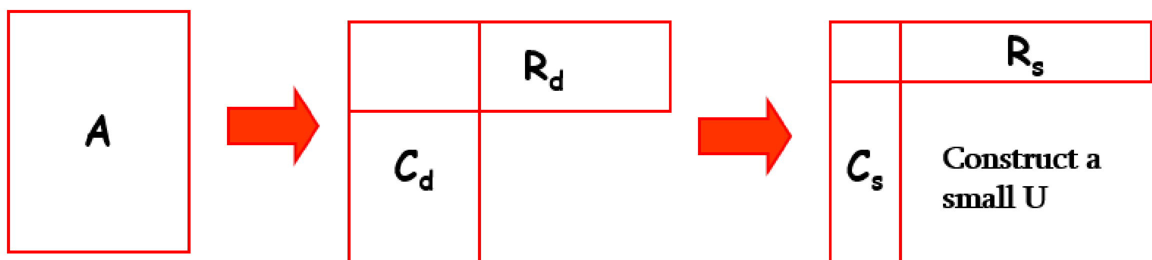

Figure 4. **Illustration of CMD Steps [32].**

Table 3

CMD Subspace Construction [7].

Input: matrix $A \in \mathbb{R}^{m \times n}$, sample size c
Output: $C_d \in \mathbb{R}^{m x n}$
1. Compute C_d using the initial subspace construction
2. Let $C \in \mathbb{R}^{m \times c'}$ be the unique columns of C_d
3. for $i = 1 : c'$
4. Let u be the number of $C(:, i)$ in C_d
5. Compute $C_s(:, i) \leftarrow \sqrt{u} \cdot C(:, i)$

Table 3 shows how CMD algorithm constructs a low dimensional subspace represented with a set of unique columns. Each column is selected by sampling the input matrix A, and then scaling it up based on the square root of the number of times it is selected [7]. Figure 4 illustrates the construction of smaller row and column components by removing redundancy from each dimension.

Apache Spark

Apache Spark [3] is a fast, open-source, in-memory data processing engine that allows data workers to efficiently execute machine learning, streaming or SQL workloads which require fast iterative access to datasets. Spark was originally developed at the AMPLab (Algorithms Machines People Lab) at UC Berkeley. Later on, the Spark codebase was donated to the Apache Software Foundation, which has maintained the Spark since then.

Apache Spark has the following characteristics [4]. Spark introduced an abstraction called resilient distributed datasets (RDDs). An RDD is a read-only collection of objects separated across a set of machines that can be rebuilt if a partition is

lost. Users can explicitly cache an RDD in memory across machines and reuse it in multiple MapReduce-like parallel operations. RDDs achieve fault tolerance through a notion of lineage: if a partition of an RDD is lost, the RDD has enough information about how it was derived from other RDDS to be able to rebuild that partition.

As shown in Figure 5, Apache Spark is contributed by five parts: Spark Core, Spark SQL, Spark Streaming, Spark MLlib, and GraphX. Spark Core is the foundation of the overall project. It provides distributed task-dispatching scheduling and basic I/O functionalities. Spark SQL is a component on top of Spark Core that introduces a data abstraction called DataFrames, which provides support for structured and semi-structured data. Spark Streaming uses Spark Core's fast scheduling capability to perform streaming analytics. Spark MLlib is a distrusted machine learning framework on top of Spark Core and GraphX is the graph processing framework on top of Apache Spark.

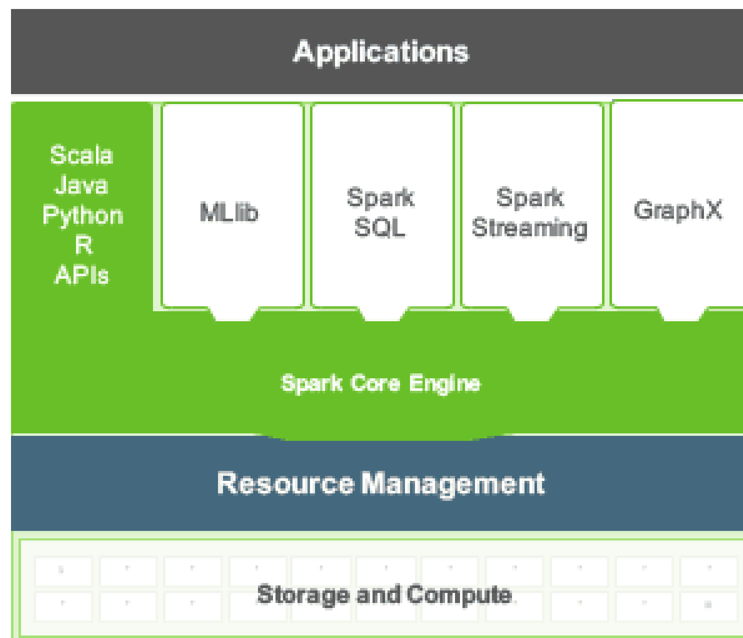

***Figure 5.* Apache Spark Construction [39].**

Apache Spark is also be used in many companies such as Baidu, eBay Inc, IBM Almaden, Yahoo!, etc. [29].

Google TensorFlow

Google TensorFlow [2] is an open source interface released by Google for expressing and implementing data mining and machine learning algorithms. The computation expressed using TensorFlow can be performed on a wide variety of heterogeneous systems with little or small change. The range is from mobile devices, such as phones and tablets, up to the large-scale distributed system of hundreds of machines. TensorFlow can be characterized as a flexible system and can be used in a wide variety of algorithms [2]. TensorFlow takes computations described using a dataflow-like model and maps them in different devices. An operation has a name and represents an abstract computation. An operation can also have attributes and all the attributes must be provided or referred at graph-construction time in order to instantiate a node to perform the operation. In a TensorFlow graph, each node has zero or more inputs and zero or more outputs, and represents the instantiation of an operation. Values that flow along normal edges in a graph are tensors [2]. A variable in TensorFlow is a special kind of operation that returns a handle to persistent mutable tensor that survives across executions of a graph.

TensorFlow provides a variety of different toolkits that allow users to construct models at users' preferred level of abstraction. As shown in Figure 6, users can use lower-level APIs to build models by defining a series of mathematical operations.

Alternatively, users can use higher-level APIs, each of which is called tf.estimator, to specify predefined architectures, such as linear regressors or neural networks.

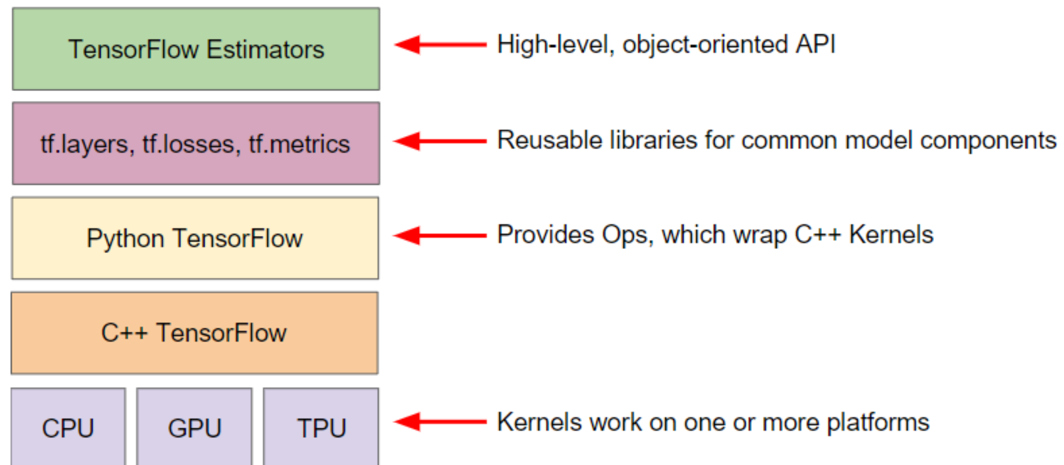

***Figure 6.* TensorFlow Toolkit [38].**

As TensorFlow is developed by Google, it has one of the largest developer community. Therefore, TensorFlow is widely used among all kinds of big companies such as Airbnb [22], Intel [23], Lenovo [24], PayPal [25], Twitter [26], Qualcomm [27], etc. TensorFlow is also the most popular deep learning framework. As illustrated in Figure 7 and Figure 8, TensorFlow has the most GitHub activity in each category and the most mentioned skill in machine learning related job listing [28].

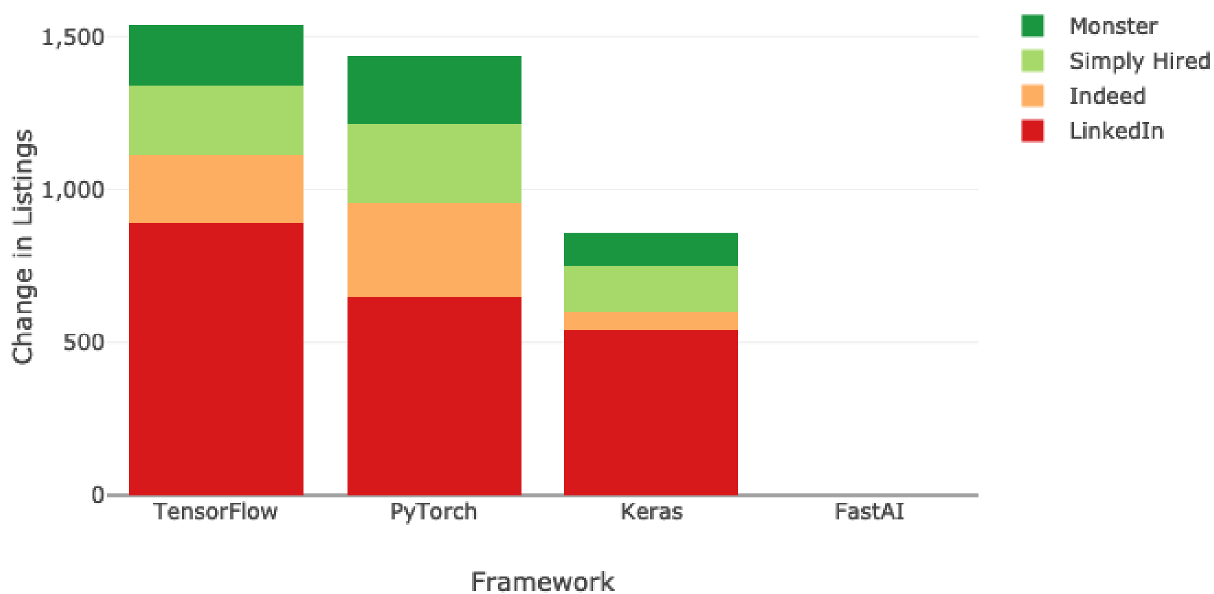

Figure 7. **Online Job Listing [28].**

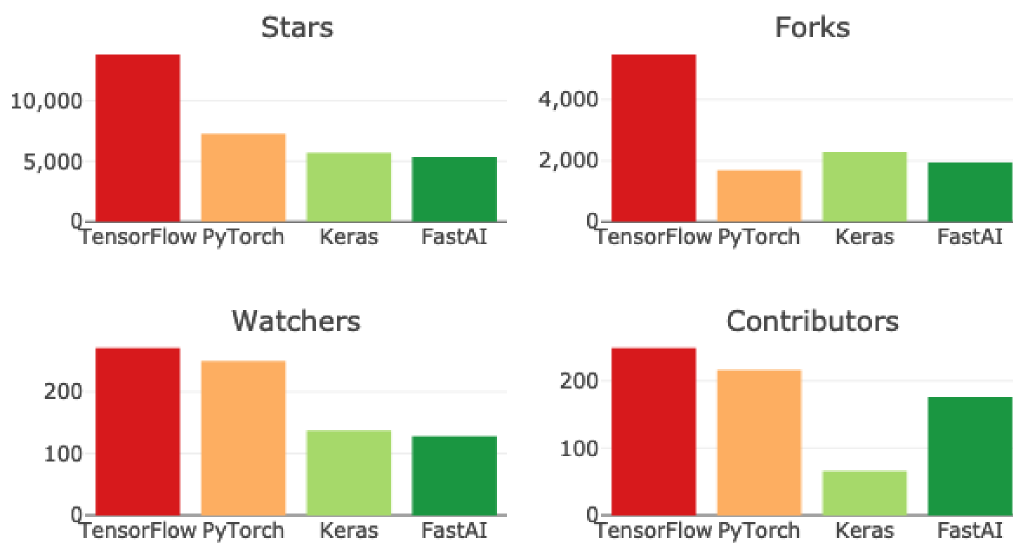

Figure 8. **New GitHub Activity [28].**

CHAPTER III

Methodology

Algorithms Implementation and Evaluation

The input data for testing is a single standard tab-delimited text file which contains the non-negative matrix with or without labels. In the experimental study, the performance is measured as follows:

1) The measurement of approximation performance. As discussed, CMD is an improved version of CUR matrix decomposition. Particularly, compared with CUR, CMD has significant improvement in terms of both storage and computing performance. CMD carefully removes duplicate columns and row after sampling, and thus it reduces both the storage space required as well as the computational effort. The key step of subspace construction is to scale up the columns that are sampled multiple times while removing the duplicates [7].

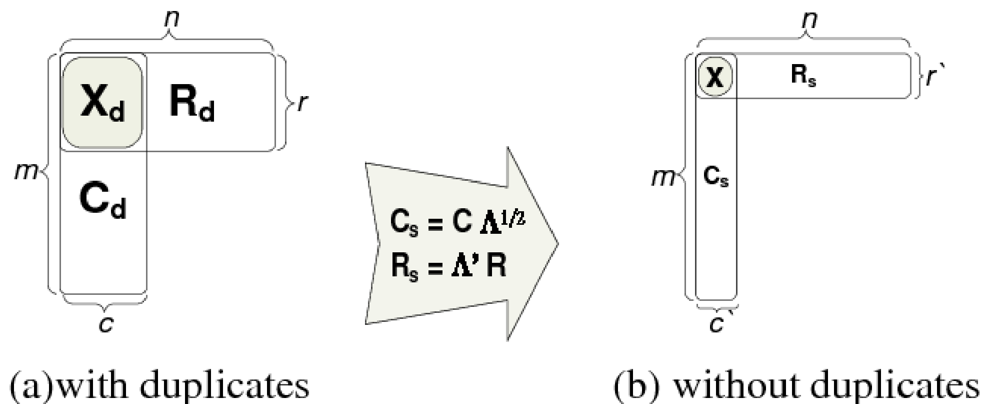

Figure 9. **Illustration of (a) CUR and (b) CMD [7].**

As shown in Figure 9, CMD carefully removes duplicate columns and rows after sampling. Intuitively, the directions of those duplicate columns are more important than

the other columns. Thus, the key step of subspace contraction is to scale up the columns that are sampled multiple times while removing the duplicates [7]. As illustrated in Figure 9, each column is selected by sampling the input matrix A, and the scaling it up based on the square root of the number of times it is selected.

2) The matrix norm as a metric for matrix approximation measurement. In linear algebra, a vector norm is a function that assigns a strictly positive length or size to each vector in a vector space except for the zero vector, which is assigned a length of zero.

3) The performance in each computing platform. The usual matrix 2-norm of the different between the original matrix and the approximated matrix by each matrix decomposition is used to measure the reconstruction error. By the positivity property of matrix norm, we know that the two inequalities (4) and (5) hold. Therefore, the other two inequalities (6) and (7) are used to compare the approximation performance. As shown in (6), if the left hand side (LHS) is greater than the right hand side (RHS), then the approximation performance with CUR is worse than the one with CMD. Contrarily, as shown in (7), if the LHS is less than the RHS, then the approximation performance with CUR is better than the one with CMD.

$$||A - CUR||_2 \geq 0 \tag{4}$$

$$||A - CMD||_2 \geq 0 \tag{5}$$

$$||A - CUR||_2 > ||A - CMD||_2 \tag{6}$$

$$||A - CUR||_2 < ||A - CMD||_2 \tag{7}$$

4) The performance between Apache Spark and Google TensorFlow platforms. The result of the approximation performance of different platforms, TensorFlow and Spark, can be measured as follows. The reconstruction error $||A - CUR_s||_2$ is compared

with the reconstruction error $||A - CUR_T||_2$ to determine the performance of CUR decomposition on different platforms. Similarly, $||A - CMD_S||_2$ is compared with $||A - CMD_T||_2$. Note the subscripts, S and T, are used to denote Apache Spark platform and Google TensorFlow platform, respectively.

Computing Platforms

The reason for selecting Apache Spark and Google TensorFlow as the test platforms is because they are state-of-the-art data mining and machine learning computing platforms in the market. Both are recently developed and actively maintained by a wide range of open-source developer communities. Although other computing platforms are available for data mining and machine learning societies, Spark and TensorFlow are promising directions in the data mining and machine learning in terms of both speed-up and scalability. Spark and TensorFlow do share some similarities in the technical aspect. Both frameworks can perform distributed operations on large datasets. They both take a set of input operations, compile these operations to a DAG (Directed Acyclic Graph), and then ship the DAG to a pool of executors and execute the DAG on a subset of the data.

However, with all the similarities, Spark and TensorFlow are still different. Spark utilizes the RDD [8] primitive for distributing any map/reduce-like operation. Spark's framework is heavily optimized for caching distributed datasets and minimizing communication between executors [10]. On the other hand, TensorFlow is a specialized tool for performing numerical operations on data, utilizing a tensor as its primitive. TensorFlow's distributed master mode [9] is different from Spark's in terms that it

partitions one DAG between multiple executors, sets up Remote Procedure Calls (RPCs) between these executors at the graph partitions, launches a parameter server for executors to read/write weight updates, and provides efficient implementations for CPU and GPU executors. For specific numerical tasks such as minimization of an objective given a huge volume of data, TensorFlow's architecture is more efficient than Spark [10].

Currently, many existing data mining and machine learning algorithms and applications have been implemented and used in numerous research and application communities. For example, Apache MRQL [14] is a Spark-based query processing deoptimization system for large-scale and distributed data analysis. RankBrain [15] is a TensorFlow-based large-scale deployment of deep neural search ranking on Google.com. Meanwhile, popular algorithms such as K-Nearest Neighbor (KNN) [16], K-means [17], and Support Vector Machine (SVM) [18] have already been implemented on TensorFlow. Although KNN is not implemented on Spark because Spark's characteristic, K-Means [19], SVM [20], and PageRank [21] are available on Spark. To best of our knowledge, CUR and CMD algorithms have not been fully implemented on Spark and TensorFlow as well as compared between the two computing platforms, yet.

Google Colab

Google Colab is a free cloud service based on Google Drive and Jupyter notebook environment. It provides the free access to GPUs for users to develop their own application using popular Apache Spark, Google TensorFlow, and Deep Learning libraries such as Keras, TensorFlow, PytTorch, and OpenCV. Currently, it supports Python 2.7 and 3.6, but not R and Scala, yet. Therefore, all the implemented algorithms are executed in Google Colab environment.

CHAPTER IV

Experiment Result

Experimental Setting

The following hardware, software, and data are used for the experimental study.

CPU: Intel Core i7-6700k at 4.4 GHz.

Ram: 16GB.

GPU: Nvidia GeForce GTX 980ti.

Operating System: Windows 10 Pro 64-bit.

Dataset: The datasets used in the experiment are all non-negative symmetric matrix from dimension 5 to dimension 50.

All the methods are tested in an identical environment on Google Colab with the same input matrix and parameter values.

Approximation Performance within Platform

Apache Spark.

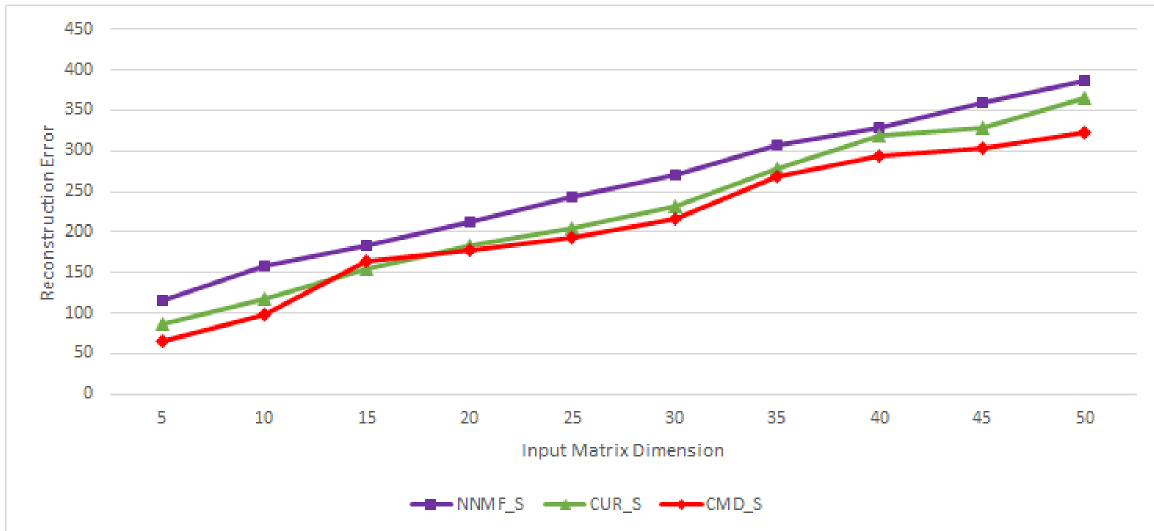

Figure 10. **Comparison in Apache Spark.**

Note that k is set as ‘input matrix dimension - 1’. With the growth of the input matrices dimension in Apache Spark, the reconstruction errors for all the three algorithms increase. Overall, CMD shows the better approximation performance than the others.

Google TensorFlow.

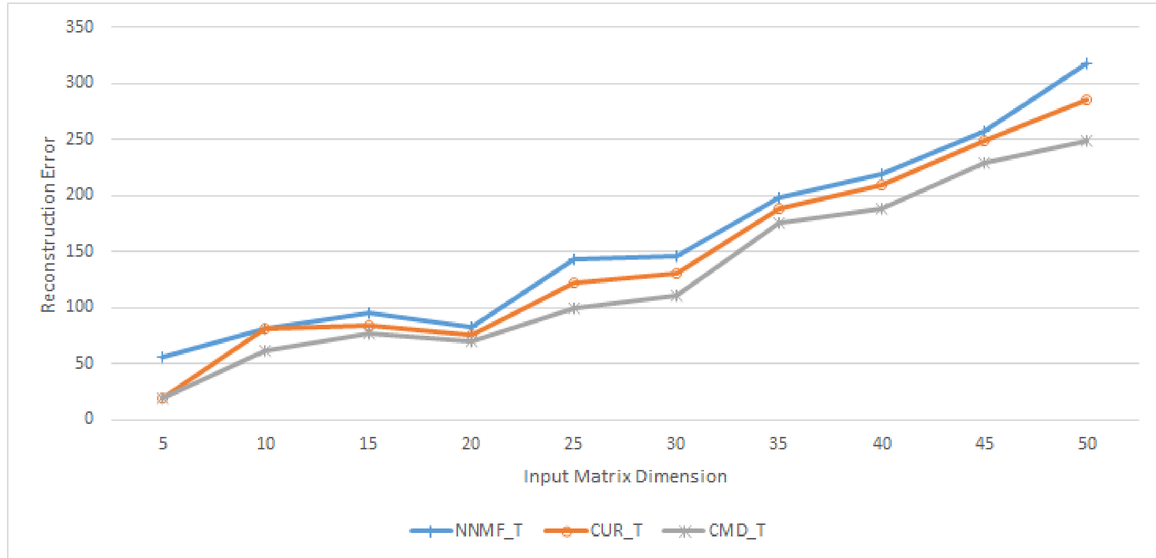

Figure 11. **Comparison in Google TensorFlow**.

Note that *k* is set as 'input matrix dimension - 1'. With the growth of the input matrices dimension in Google TensorFlow, the reconstruction errors for all the three algorithms increase. Overall, CMD shows the better approximation performance than the others

Approximation Performance between Platforms

NNMF.

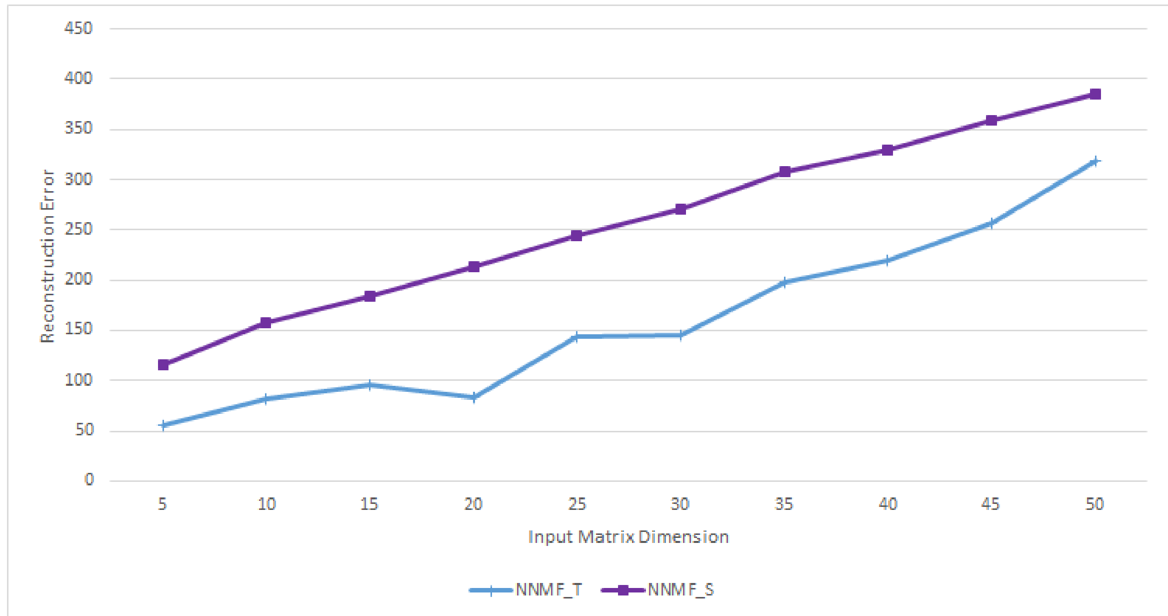

Figure 12. **Comparison of NNMF between TensorFlow and Spark**.

Note that k is set as 'input matrix dimension - 1'. As before, with the growth of the input matrices dimension, the reconstruction errors for NNMF increase. NNMF in Google TensorFlow has better approximation performance than the one in Apache Spark.

CUR.

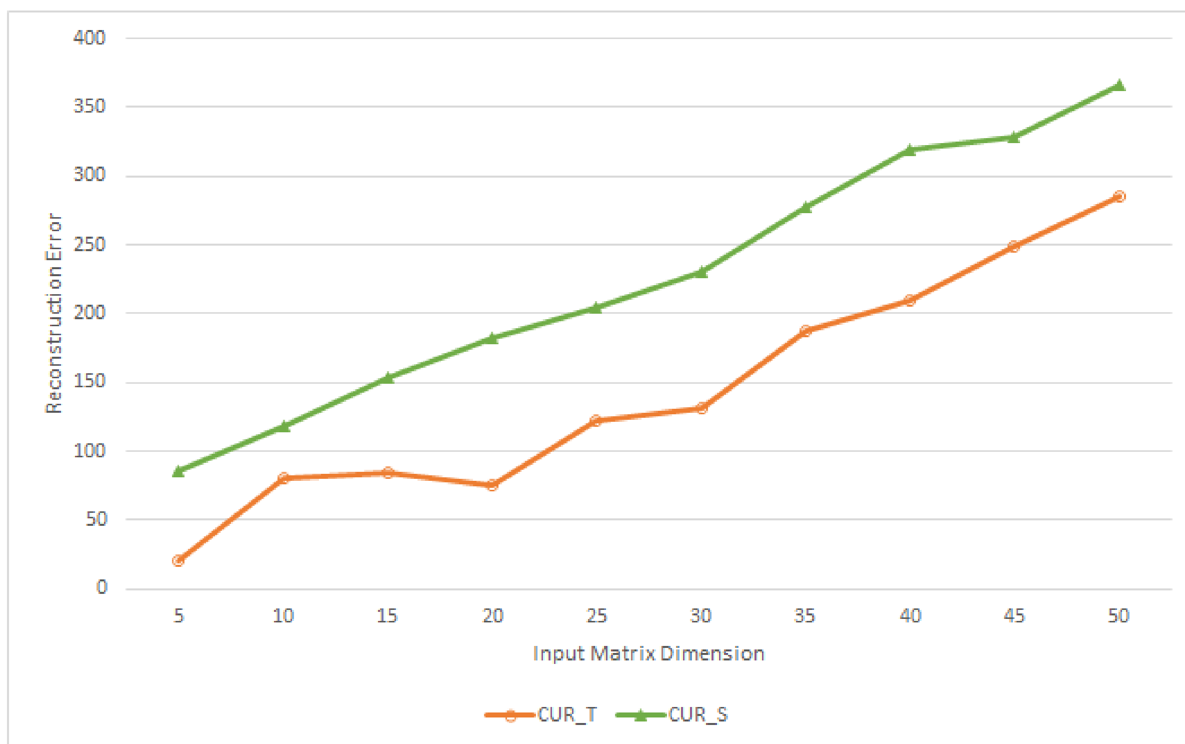

Figure 13. **Comparison of CUR between TensorFlow and Spark**.

Note that k is set as 'input matrix dimension - 1'. As before, with the growth of the input matrices dimension, the reconstruction errors for CUR increase. CUR in Google TensorFlow has better approximation performance than the one in Apache Spark.

CMD.

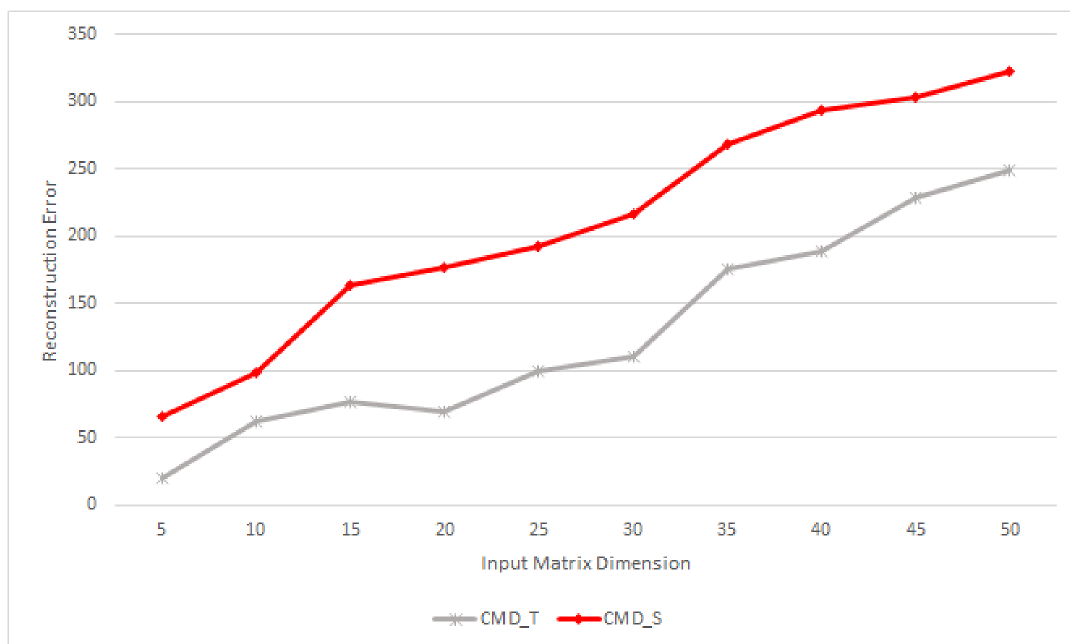

Figure 14. **Comparison of CMD between TensorFlow and Spark.**

Note that k is set as 'input matrix dimension - 1'. As before, with the growth of the input matrices dimension, the reconstruction errors for CMD increase. CMD in Google TensorFlow has better approximation performance than the one in Apache Spark.

Overall Approximation Performance

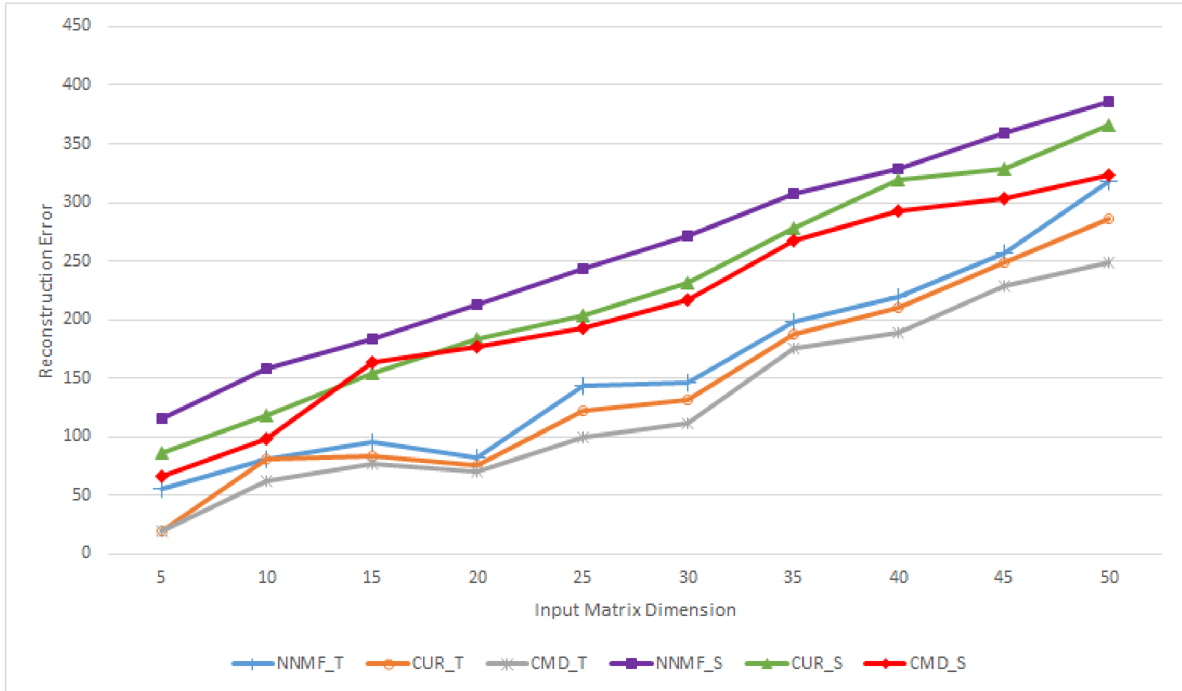

Figure 15. **Approximation Performance Comparison**.

Note that *k* is set as 'input matrix dimension - 1'. As discussed before, with the growth of the input matrices dimension, the reconstruction errors for all the algorithms increase. As shown in Figure 15, the algorithms in Google TensorFlow perform better than the ones in Apache Spark. Particularly, CMD in Google TensorFlow results in the best approximation performance.

Running Time Performance

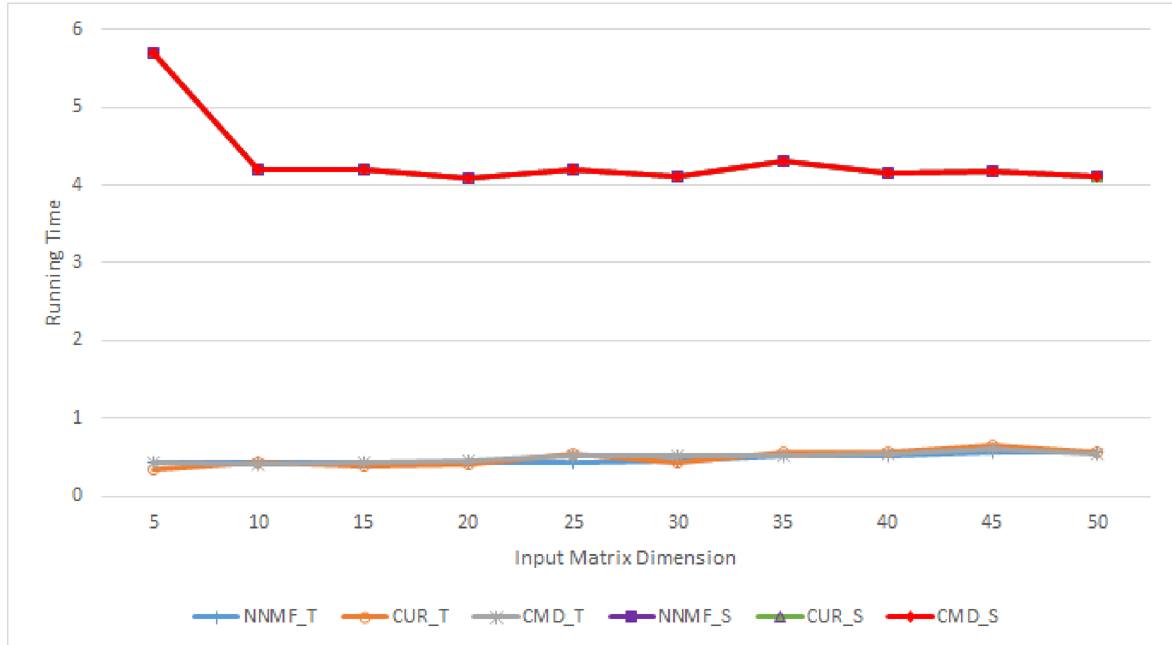

Figure 16. **Running Time Performance Comparison**.

Note that *k* is set as 'input matrix dimension - 1'. The algorithms on Google TensorFlow take less running/execution time than the ones on Apache Spark.

Approximation Performance over Varying k

Apache Spark.

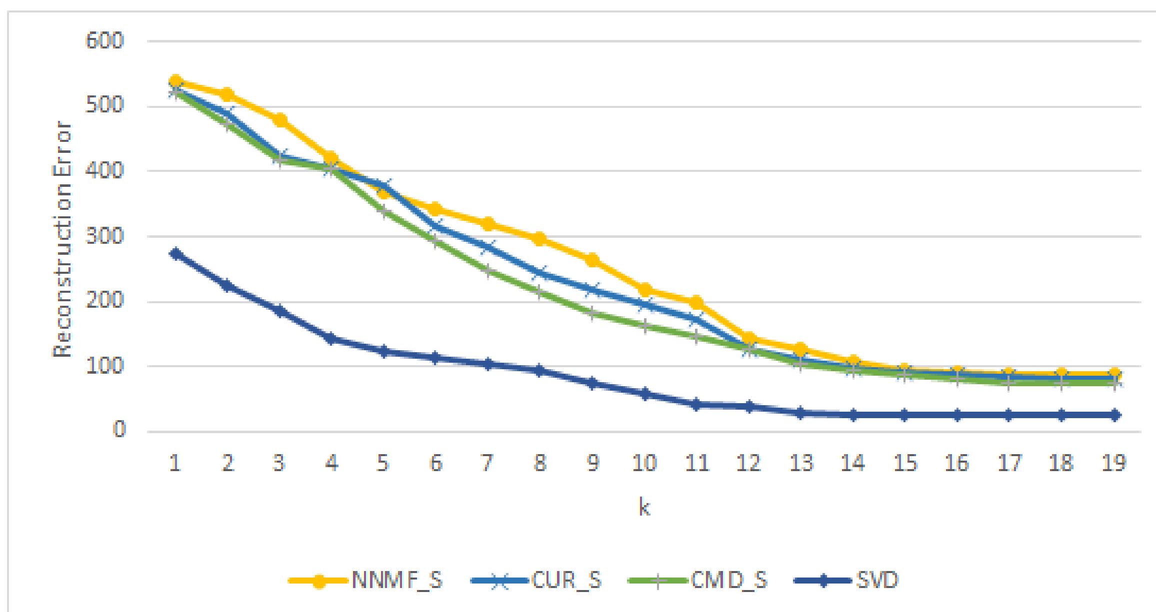

Figure 17. **Approximation Performance over Varying k in Spark**.

Figure 17 shows the reconstruction errors of the four algorithms (NNMF, CUR, CMD, and SVD) in Apache Spark. As the number (denoted as k) of vectors selected for every algorithm increase, the reconstruction errors decreases. It is expected to get the best matrix approximation with SVD as it is the optimal rank-k approximation algorithm. Overall, CMD shows better approximation performance than the other two algorithms (NNMF and CUR).

Google TensorFlow.

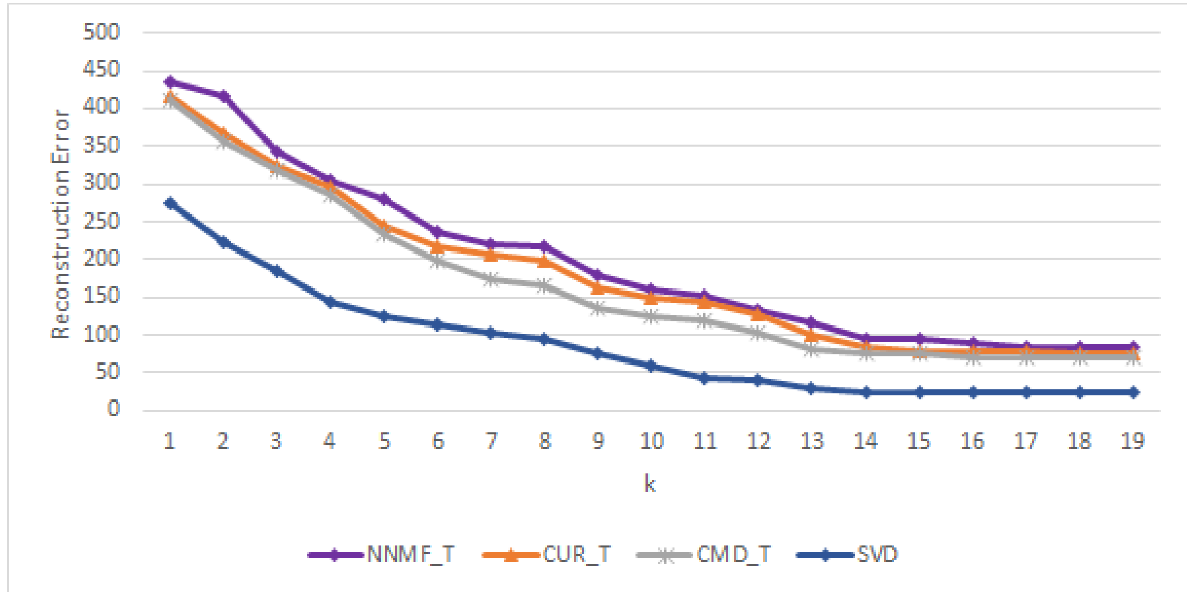

Figure 18. **Approximation Performance over Varying k in TensorFlow**.

Figure 18 depicts the reconstruction errors of the four algorithms (NNMF, CUR, CMD, and SVD) in Google TensorFlow. As for the cases in Apache Spark, the reconstruction errors with all the algorithms decrease with the increase of the number (denoted as k) of vectors selected for every algorithm. Also, as expected, SVD results in the best matrix approximation over all the considered range of k values. Overall, CMD shows better approximation performance than the other two algorithms (NNMF and CUR).

Overall Approximation Performance over Varying k

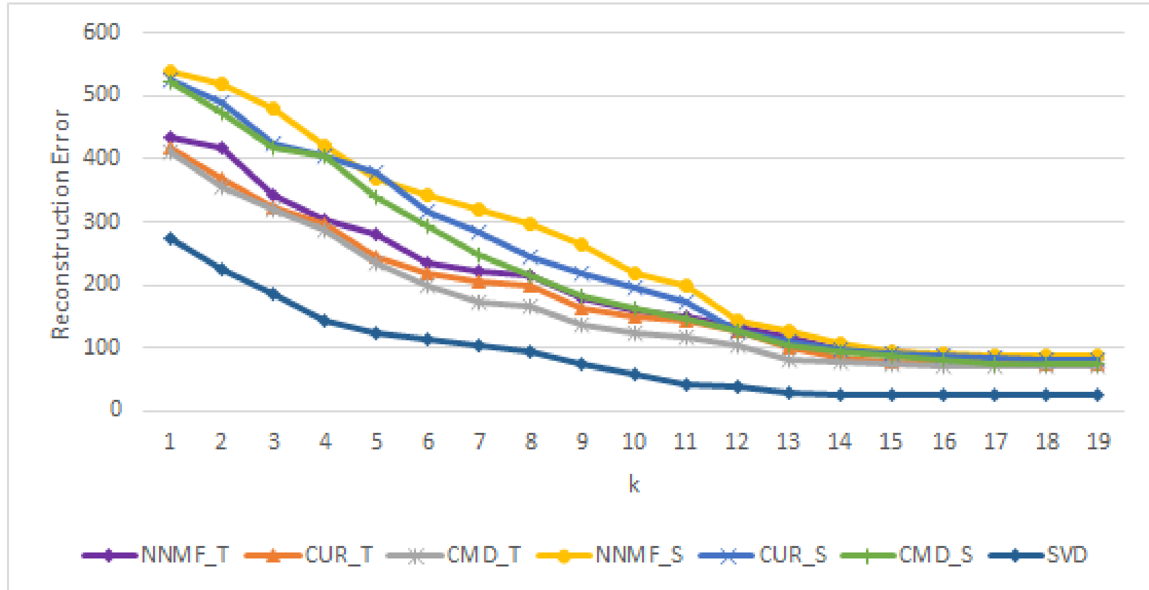

Figure 19. **Approximation Performance over Varying k.**

Figure 19 illustrates the approximation performance of all the four matrix approximation algorithms (NNMF, CUR, CMD, and SVD) and the two computing platforms (Apache Spark and Google TensorFlow). As discussed, SVD performs best. Overall, CMD in Google TensorFlow shows better performance than the other cases except the one with SVD.

CHAPTER V

Conclusion & Future Work

Conclusion

In this study, the three popular non-negative matrix factorization algorithms, NNMF, CUR, and CMD are implemented in the two new computing platforms, Apache Spark and Google TensorFlow. The approximation performance is measured by the usual matrix 2-norm between the original matrix and the matrix reconstructed by each algorithm. Along with the optimal approximation by SVD, the matrix approximation performance by all the six cases are compared. Note that currently all the six cases result in much less accurate approximation, compare with the optimal approximation by the SVD. CMD in Google TensorFlow shows the best approximation performance among all the six cases, which are from the three algorithms (NNMF, CUR, and CMD) and the two platforms (Apache Spark and Google TensorFlow). The experimental result also shows that the approximation improves in terms of the reconstruction error, as the number of rows and columns selected for CUR and CMD.

Future Work

Although it is the first that compares the three non-negative matrix factorization algorithms together on the two computing platforms, this study should be considered preliminary as more systematic and comprehensive study is necessary. Large-scale data should be used to validate the correctness of algorithm implementation as well as the characteristics of the algorithms with more comprehensive parameter setting.

To be more specific, the following should be investigated in future study:

- For example, large scaled data in real-life problems
- Investigation of reconstruction error of each algorithm over exhaustive k values
- Minimization of the gap from SVD's optimal rank-k approximation over the whole range of the number of rows and columns selected for CUR and CMD
- Comparison of actual running time as well as storage requirements for every algorithm
- Handling both dense and sparse matrices
- Section of different rows and columns instead of the fixed parameter value of k for both the dimensions
- Comprehensive experimental study on the speed-up and the scalability of each algorithm with large-scale data

REFERENCES

[1] D. D. Lee and H. S. Seung, ‘Algorithms for Non-negative Matrix Factorization,’ Advances in Neural Information Processing Systems 13 (NIPS 2000), pp. 535-541.

[2] M. Abadi et al, ‘TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,’ Preliminary whtie Paper, November 9, 2015.

[3] ‘What is Apache Spark,’ Hortonworks, [Online]. Available: https://hortonworks.com/apache/spark/ [Accessed: 2 Otc. 2018].

[4] M. Zaharia, M. Chowdhury et al, ‘Spark: Cluster Computing with Working Sets,’ USENIX Workshop on Hot Topics in Cloud Computing.

[5] D. D. Lee and H. S. Seung, ‘Learning the parts of objects by non-negative matrix factorization,’ *Nature*, vol. 401, no. 6755, pp. 788–791, 1999.

[6] M. Mahoney and P. Drineas, ‘CUR matrix decompositions for improved data analysis,’ PNAS, vol. 106, no.3, pp. 697-702, January 12, 2009.

[7] J. Sun, Y. Xie, H. Zhang, and C. Faloutsos, ‘Less is more: compact matrix decomposition for large sparse graphs,’ Statistical Analysis and Data Mining, vol.1, pp.6-22, 2008.

[8] ‘RDD Programming Guide,’ [Online]. Available: https://spark.apache.org/docs/latest/rdd-programming-guide.html, [Accessed: 8 Otc. 2018].

[9] ‘TensorFlow Architecture,’ [Online]. Available: https://www.tensorflow.org/extend/architecture#distributed_master, [Accessed: 8 Otc. 2018].

[10] ‘Why do people intergrate Spark with TensorFlow even if there is a distributed TensorFlow framwork,’ [Online]. Available: https://www.quora.com/Why-do-people-integrate-Spark-with-TensorFlow-even-if-there-is-a-distributed-TensorFlow-framework, [Accessed: 8 Otc. 2018].

[11] eesungkim _ ,[Online] Available: https://github.com/eesungkim/NMF-Tensorflow/blob/master/nmf.py. [Accessed: 8 Otc. 2018].

[12] R. Liao, Y. Zhang, J. Guan, and S. Zhou, ‘CloudNMF: A MapReduce Implementation of Nonnegative Matrix Factorization for Large-scale Biological Datasets,’ Genomics, Proteomics & Bioinformatics, vol. 12, no. 1, pp. 48–51, 2014.

[13] E. Mejia-Roa, P. Carmona-Saez, R. Nogales, C. Vicente, M. Vazquez, X. Y. Yang, C. Garcia, F. Tirado, and A. Pascual-Montano, ‘bioNMF: a web-based tool for nonnegative matrix factorization in biology,’ Nucleic Acids Research, vol. 36, no. Web Server, pp.523-528, 2008.

[14] [Online], available: https://wiki.apache.org/mrql. Accessed: [10 Otc. 2018].

[15] ‘Google RankBrain: The Defintive Guide,’ [online]. Available: https://backlinko.com/google-rankbrain-seo. Accessed: [10 Otc. 2018].

[16] ‘Simplest TensorFlow examole(KNN),’ [online]. Available: https://ensemblearner.github.io/blog/2017/04/01/knn. Accessed: [10 Otc. 2018].

[17] ‘tf.contrib.learn.KMeansClustering,’ [online]. Available: https://www.tensorflow.org/api_docs/python/tf/contrib/learn/KMeansClustering. Accessed: [10 Otc. 2018].

[18] ‘tf.contrib.learn.SVM,’ [online]. Available: https://www.tensorflow.org/api_docs/python/tf/contrib/learn/SVM. Accessed: [10 Otc. 2018].

[19] [Online], available: https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-mllib/spark-mllib-KMeans.html. Accessed: [10 Otc. 2018].

[20] [Online], available: https://spark.apache.org/docs/2.2.0/api/python/pyspark.ml.html#pyspark.ml.classification.LinearSVC. Accessed: [10 Otc. 2018].

[21] [Online], Available: https://raw.githubusercontent.com/abbas-taher/pagerank-example-spark2.0-deep-dive/master/SparkPageRank.scala. Accessed: [10 Otc. 2018].

[22] S. Yao, ‘Categorizing Listing Photos at Airbnb,’ Accessed: [Online], Available: https://medium.com/airbnb-engineering/categorizing-listing-photos-at-airbnb-f9483f3ab7e3. Accessed: [20 June 2019].

[23] ‘TensorFlow* Optimizations for the Intel® Xeon® Scalable Processor,’ [Online], https://www.intel.ai/tensorflow-optimizations-intel-xeon-scalable-processor/#gs.kffjdj. Accessed: [20 June 2019].

[24] [Online], https://www.lenovo.com/us/en/data-center/software/Lenovo-Intelligent-Computing-Orchestration/p/WMD00000356. Accessed: [20 June 2019].

[25] [Online], https://medium.com/paypal-engineering. Accessed: [20 June 2019].

[26] ‘Ranking Tweets with TensorFlow,’ [Online], https://medium.com/tensorflow/ranking-tweets-with-tensorflow-932d449b7c4. Accessed: [20 June 2019].

[27] 'TensorFlow machine learning now optimized for the Snapdragon 835 and Hexagon 682 DSP,' [Online], https://www.qualcomm.com/news/onq/2017/01/09/tensorflow-machine-learning-now-optimized-snapdragon-835-and-hexagon-682-dsp. Accessed: [20 June 2019].

[28] J. Hale, 'Which Deep Learning Framework is Growing Fastest?,' [Online], https://towardsdatascience.com/which-deep-learning-framework-is-growing-fastest-3f77f14aa318. Accessed: [20 June 2019].

[29] [Online], https://spark.apache.org/powered-by.html. Accessed: [20 June 2019].

[30] S. Arora, R. Ge, and Y. Halperm, 'A practical algorithm for topic modeling with provable guarantees,' in Proc. Int. Conf. Machine Learning, vol. 28, no. 2, 2013, pp. 280–288.

[31] M. Mahoney, [Online], https://web.stanford.edu/group/mmds/slides/mahoney-mmds.pdf. Accessed: [20 June 2019].

[32] J. Sun, 'Less is More: Compact Matrix Decomposition for Large Sparse Graphs,' [Online], https://slideplayer.com/slide/5221432/. Accessed: [20 June 2019].

[33] [Online], https://research.google.com/colaboratory/faq.html. Accessed: [20 June 2019].

[34] O. Berné, C. Joblin, Y. Deville, J. D. Smith, M. Rapacioli, J. P. Bernard, J. Thomas, W. Reach, A. Abergel. 'Analysis of the emission of very small dust particles from Spitzer spectro-imagery data using blind signal separation methods,' Astronomy & Astrophysics, vol. 469, pp.575-586, July II 2007.

[35] D. Lafrenière, C.Marois, and R. Doyon. 'HST/NICMOS Detection of HR 8799 b in 1998.' The Astrophysical Journal Letters, vol. 694, pp. 148-152, April 2009.

[36] B. Ren, L. Pueyo, and J. Debes. [2018]. ‘Non-negative Matrix Factorization: Robust Extraction of Extended Structures,’ The Astrophysical Journal, vol. 852.

[37] N. Mitrovic, [Online], http://www.mit.edu/~jaillet/general/itsc13-cur.pdf. Accessed: [24 June 2019].

[38] ‘First Steps with TensorFlow: Toolkit,’ https://developers.google.com/machine-learning/crash-course/first-steps-with-tensorflow/toolkit. Accessed: [29 June 2019].

[39] [Online], https://hortonworks.com/apache/spark/. Accessed: [29 June 2019].

[40] [Online], https://medium.com/tensorflow/intelligent-scanning-using-deep-learning-for-mri-36dd620882c4/. Accessed: [30 June 2019].

[41] [Online], https://intellipaat.com/tutorial/spark-tutorial/apache-spark-applications/. Accessed: [29 June 2019].

[42] G.H. Golub and C.F.Van, ‘Matrix Computation (3rd Edition),’ Baltimore, Maryland: The Johns Hopkins University Press. ISBN 978-0-8018-5414-9.

[43] T. Roughgarden and G. Valiant, ‘CS168: The Modern Algorithmic Toolbox Lecture #9: The Singular Value Decomposition (SVD) and Low-Rank Matrix Approximations,’ [Online], http://theory.stanford.edu/~tim/s15/l/l9.pdf. Accessed: [29 June 2019].

APPENDIX

The original NNMF[1] implemented in Google TensorFlow[5].

```
import numpy as np
import tensorflow as tf

class NMF:
    """Compute Non-negative Matrix Factorization (NMF)"""
    def __init__(self, max_iter=200, learning_rate=0.01,display_step=10, optimizer='mu',
initW=False):

        self.max_iter = max_iter
        self.learning_rate= learning_rate
        self.display_step = display_step
        self.optimizer = optimizer

    def NMF(self, X, r_components, learning_rate, max_iter, display_step, optimizer,
initW, givenW ):
        m,n=np.shape(X)
        tf.reset_default_graph()
        V = tf.placeholder(tf.float32)

        initializer = tf.random_uniform_initializer(0,1)
        if initW is False:
            W =  tf.get_variable(name="W", shape=[m, r_components], initializer=initializer)
            H =  tf.get_variable("H", [r_components, n], initializer=initializer)
        else:
            W =  tf.constant(givenW, shape=[m, r_components], name="W")
            H =  tf.get_variable("H", [r_components, n], initializer=initializer)

        WH =tf.matmul(W, H)
        cost = tf.reduce_sum(tf.square(V - WH))

        if optimizer=='mu':
            """Compute Non-negative Matrix Factorization with Multiplicative Update"""
            Wt = tf.transpose(W)
            H_new = H * tf.matmul(Wt, V) / tf.matmul(tf.matmul(Wt, W), H)
            H_update = H.assign(H_new)

            if initW is False:
                Ht = tf.transpose(H)
                W_new = W * tf.matmul(V, Ht)/ tf.matmul(W, tf.matmul(H, Ht))
                W_update = W.assign(W_new)

        elif optimizer=='pg':
```

```
            """optimization; Projected Gradient method """
            dW, dH = tf.gradients(xs=[W, H], ys=cost)
            H_update_ = H.assign(H - learning_rate * dH)
            H_update = tf.where(tf.less(H_update_, 0), tf.zeros_like(H_update_), H_update_)

            if initW is False:
                W_update_ = W.assign(W - learning_rate * dW)
                W_update = tf.where(tf.less(W_update_, 0), tf.zeros_like(W_update_),
W_update_)

        with tf.Session() as sess:
            sess.run(tf.global_variables_initializer())
            for idx in range(max_iter):
                if initW is False:
                    W=sess.run(W_update, feed_dict={V:X})
                    H=sess.run(H_update, feed_dict={V:X})
                else:
                    H=sess.run(H_update, feed_dict={V:X})
                if (idx % display_step) == 0:
                    costValue = sess.run(cost,feed_dict={V:X})
                    print("|Epoch:","{:4d}".format(idx), " Cost=","{:.3f}".format(costValue))
        return W, H

    def fit_transform(self, X,r_components, initW, givenW):
        """Transform input data to W, H matrices which are the non-negative matrices."""
        W, H =  self.NMF(X=X, r_components = r_components,
learning_rate=self.learning_rate,
                    max_iter = self.max_iter, display_step = self.display_step,
                    optimizer=self.optimizer, initW=initW, givenW=givenW  )
        return W, H

    def inverse_transform(self, W, H):
        """Transform data back to its original space."""
        return np.matmul(W,H)

def main():
    V = np.array([[1, 1], [2, 1], [3, 1.2], [4, 1], [5, 0.8], [6, 1]])
    model = NMF(max_iter=200,learning_rate=0.01,display_step=10, optimizer='mu')
    W, H = model.fit_transform(V, r_components=2, initW=False, givenW=0)
    print(W)
    print(H)
    print(V)
    print(model.inverse_transform(W, H))

if __name__ == '__main__':
        main()
```

CUR_T.py

```
import numpy as np
from typing import Tuple
import tensorflow as tf
import pandas as pd
np.random.seed(0)

Matrix = np.ndarray
Vector = np.ndarray

M = np.array(
[[3, 4, 8, 9, 52, 12],
[4, 4, 9, 44, 26, 69],
[5, 5, 5, 15, 95, 26],
[4, 4, 5, 4, 36, 69],
[59, 5, 4, 99, 62, 36],
[5, 4, 43, 65, 86, 69]]

    , dtype=np.float64).T
print('CUR', M.shape)

def cur_decomposition(M: Matrix, r: int):

    m, n = M.shape
    r_probs, c_probs = probabilities(M)
    C, c_idx = select_C(M, r, c_probs)
    R, r_idx = select_R(M, r, r_probs)
    U = make_U(M, c_idx, r_idx)
    return C, U, R

def probabilities(M: Matrix):
    squared = np.square(M)
    row_sum = np.sum(squared, 1)
    col_sum = np.sum(squared, 0)
    denom = np.sum(row_sum)
    row_probs = row_sum / denom
    col_probs = col_sum / denom
    return row_probs, col_probs

def select_C(M, r, probs):
    return select_part(M, r, probs, 1)

def select_R(M, r, probs):
    return select_part(M, r, probs, 0)
```

```
def select_part(M: Matrix, r: int, probs: Vector, axis: int):

    size = M.shape[axis]
    idx = np.random.choice(size, size=r, p=probs)
    selected = np.take(M, idx, axis)
    scale = probs[idx]
    scale = np.sqrt(scale * r)
    scale = np.expand_dims(scale, axis - 1)
    return selected / scale, idx

def make_U(M: Matrix, c: Vector, r: Vector):
    W = select_W(M, c, r)
    x, e, y = np.linalg.svd(W)
    inv_e = psuedo_inverse(e)

    Us = np.matmul(y.T ,np.diag(np.square(inv_e)))
    U = np.matmul(Us, x.T)

    return U

def select_W(M: Matrix, c: Vector, r: Vector):
    return M[r, :][:, c]

def psuedo_inverse(sigma: Vector):
    zeros = sigma == 0
    # Get mask where 0's in sigma are 1
    num = (zeros) == 0
    # Replace zero with one
    denum = zeros + sigma
    # do inverse because 0 is now 0/1
    return num / denum

C1, U1, R1 = cur_decomposition(M,M.shape[1]-1)

C1 = np.divide(C1, C1.max())
U1 = np.divide(U1, U1.max())
R1 = np.divide(R1, R1.max())

C = tf.Variable(C1)
U = tf.Variable(U1)
R = tf.Variable(R1)

CU = tf.matmul(C, U)
CUR = tf.matmul(CU,R)

A_orig_df = pd.DataFrame(M)
```

```
A_df_masked = A_orig_df.copy()
np_mask = A_df_masked.notnull()

tf_mask = tf.Variable(np_mask.values)
A = tf.constant(A_df_masked.values)

#cost of Frobenius norm
cost = tf.reduce_sum(tf.pow(tf.boolean_mask(A, tf_mask) - tf.boolean_mask(CUR,
tf_mask), 2))

# Clipping operation. This ensure that C, U, and R learnt are non-negative
clip_C = C.assign(tf.maximum(tf.zeros_like(C), C))
clip_U = U.assign(tf.maximum(tf.zeros_like(U), U))
clip_R = R.assign(tf.maximum(tf.zeros_like(R), R))
clip = tf.group(clip_C, clip_U, clip_R)

# Learning rate
lr = 0.0001
# Number of steps
train_step = tf.compat.v1.train.GradientDescentOptimizer(lr).minimize(cost)
init = tf.compat.v1.global_variables_initializer()

#tensorflow
steps = 1000
with tf.compat.v1.Session() as sess:
    sess.run(init)
    for i in range(steps):
        sess.run(train_step)
        sess.run(clip)

    learnt_C = sess.run(C)
    learnt_U = sess.run(U)
    learnt_R = sess.run(R)

pred_CU = np.dot(learnt_C, learnt_U)
pred = np.dot(pred_CU, learnt_R)
pred_df = pd.DataFrame(pred)

L2norm = np.linalg.norm(pred_df.round()-A_orig_df)
print('\n2-Norm', L2norm)
```

CMD_T.py

```
import numpy as np
from typing import Tuple
import tensorflow as tf
import pandas as pd
np.random.seed(0)

Matrix = np.ndarray
Vector = np.ndarray

M = np.array(
[[3, 4, 8, 9, 52, 12],
[4, 4, 9, 44, 26, 69],
[5, 5, 5, 15, 95, 26],
[4, 4, 5, 4, 36, 69],
[59, 5, 4, 99, 62, 36],
[5, 4, 43, 65, 86, 69]]
    , dtype=np.float64).T

print('CMD', M.shape)

def cur_decomposition(M: Matrix, r: int):
    m, n = M.shape
    r_probs, c_probs = probabilities(M)
    C, c_idx = select_C(M, r, c_probs)
    R, r_idx = select_R(M, r, r_probs)
    U = make_U(M, c_idx, r_idx)
    return C, U, R

def probabilities(M: Matrix):
    squared = np.square(M)
    row_sum = np.sum(squared, 1)
    col_sum = np.sum(squared, 0)
    denom = np.sum(row_sum)
    row_probs = row_sum / denom
    col_probs = col_sum / denom
    return row_probs, col_probs

def select_C(M, r, probs):
    return select_part(M, r, probs, 1)

def select_R(M, r, probs):
    return select_part(M, r, probs, 0)

def select_part(M: Matrix, r: int, probs: Vector, axis: int):
```

```
    size = M.shape[axis]
    idx = np.random.choice(size, size=r, p=probs)
    selected = np.take(M, idx, axis)
    scale = probs[idx]
    scale = np.sqrt(scale * r)
    scale = np.expand_dims(scale, axis - 1)
    return selected / scale, idx

def make_U(M: Matrix, c: Vector, r: Vector):
    W = select_W(M, c, r)
    x, e, y = np.linalg.svd(W)
    inv_e = psuedo_inverse(e)

    Us = np.matmul(y.T ,np.diag(np.square(inv_e)))
    U = np.matmul(Us, x.T)

    return U

def select_W(M: Matrix, c: Vector, r: Vector):
    return M[r, :][:, c]

def psuedo_inverse(sigma: Vector):
    zeros = sigma == 0
    # Get mask where 0's in sigma are 1
    num = (zeros) == 0
    # Replace zero with one
    denum = zeros + sigma
    # do inverse because 0 is now 0/1
    return num / denum

C1, U1, R1 = cur_decomposition(M,M.shape[1]-1)

C2 = np.array(list(set([tuple(t) for t in C1])))
R2 = np.array(list(set([tuple(t) for t in R1])))

C2 = np.divide(C2, C2.max())
R2 = np.divide(R2, R2.max())

temp_U = np.random.randn(C2.shape[1], R2.shape[0]).astype(np.float64)
U2 = np.divide(temp_U, temp_U.max())

C = tf.Variable(C2)
U = tf.Variable(U2)
R = tf.Variable(R2)
```

```
CU = tf.matmul(C, U)
CUR = tf.matmul(CU,R)

A_orig_df = pd.DataFrame(M)
A_df_masked = A_orig_df.copy()
#A_df_masked.iloc[0,0]=np.NAN
np_mask = A_df_masked.notnull()

tf_mask = tf.Variable(np_mask.values)
A = tf.constant(A_df_masked.values)

#cost of Frobenius norm
cost = tf.reduce_sum(tf.pow(tf.boolean_mask(A, tf_mask) - tf.boolean_mask(CUR,
tf_mask), 2))

# Clipping operation. This ensure that C, U, and R learnt are non-negative
clip_C = C.assign(tf.maximum(tf.zeros_like(C), C))
clip_U = U.assign(tf.maximum(tf.zeros_like(U), U))
clip_R = R.assign(tf.maximum(tf.zeros_like(R), R))
clip = tf.group(clip_C, clip_U, clip_R)

# Learning rate
lr = 0.0001
# Number of steps
train_step = tf.compat.v1.train.GradientDescentOptimizer(lr).minimize(cost)
init = tf.compat.v1.global_variables_initializer()

#tensorflow
steps = 1000
with tf.compat.v1.Session() as sess:
    sess.run(init)
    for i in range(steps):
        sess.run(train_step)
        sess.run(clip)
    learnt_C = sess.run(C)
    learnt_U = sess.run(U)
    learnt_R = sess.run(R)

pred_CU = np.dot(learnt_C, learnt_U)
pred = np.dot(pred_CU, learnt_R)
pred_df = pd.DataFrame(pred)

L2norm = np.linalg.norm(pred_df.round()-A_orig_df)
print('\n2-Norm', L2norm)
```

VITA

EDUCATION

Master of Science student in Computer Science at Sam Houston State University, September 2017 - present. Thesis title: "NNMF in Google TensorFlow and Apache Spark: A Comparison Study."

Bachelor of Science (May 2017) in Computer Science at Sam Houston State University, Huntsville, Texas.

ACADEMIC EMPLOYMENT

Graduate Teaching Assistant, Department of Computer Science, Sam Houston State University, September 2017 – present.

ACADEMIC AWARDS

Academic Affairs Scholars, Sam Houston State University, May 2014.