CS 136 FINAL PROJECT: A Quantum-Inspired Classical Algorithm to Build Recommender Systems

By: Ary Swaminathan, Savvy Raghuvanshi, Raahul Acharya

December 2018

1 Introduction

A recommendation system suggests products to users based on data preferences. User-user and product-product recommendation systems are usually modeled by completing the entries of an m x n matrix M where M_{ij} represents user i's preferences towards product j. Another significant characteristic of the matrix is its small rank k, which is essential when performing matrix factorization of M into smaller matrices of a minimum dimension k.

In this project, we will be implementing the first classical algorithm to produce a recommendation in O(poly(k)polylog(m,n)) time, which is an exponential improvement to previous classical algorithms which run in time linear to m and n.

This project stemmed from Ewin Tang's paper "A Quantum-Inspired Classical Algorithm for Recommendation Systems" which outlined a classical algorithm inspired by the quantum algorithm of Kerenidis and Prakash (Quantum Recommendation Systems. In 8th Innovations in Theoretical Computer Science Conference (ITCS 2017), volume 67, pages 49:1–49:21, Dagstuhl, Germany, 2017.).

An essential characteristic of the algorithm involves seeking only a randomized sample from the user's preferences instead of reconstructing all of them, which would, of course, require large amounts of data computation. It samples the high-value entries from a low-rank approximation of an input matrix, which is nothing more than an approximated factorization of this input matrix into smaller, more tractable matrices in time independent of m and n, the number of users and number of items, respectively.

As a consequence of implementing this algorithm, we prove that Kerenidis and Prakash's quantum algorithm, which displays one of the most significant examples of quantum speedup, does not in fact provide an exponential speedup over classical algorithms.

To complete the entries of a large user preference matrix, we function off the standard assumptions that users tend to fall in a small number of classes based upon their preferences, using these classes we can build future recommendations for products the users have not interacted with.

Current algorithms tend to reconstruct full rows of user's preferences, taking $\Omega(n)$ time. This is addressed in this new algorithm by only sampling the high value entries in a given user's data, and using these to build future recommendations.

2 Algorithm

2.1 Sketch

Our algorithm is based off the following two theorems produced in Tang's paper.

Theorem 1: Suppose we are given as an input a matrix A supporting query and ℓ^2-norm sampling operations, a row/user $i \in [m]$, a singular threshold σ, and a sufficiently small $\epsilon > 0$. There is a classical algorithm whose output distribution is O(ϵ)-close in total variation distance to the distribution given by ℓ^2-norm sampling from the ith row of a low-rank approximation D of A in query and time complexity, where the run time of this algorithm is independent of m and n, as shown below:

$O(poly(\frac{||A||_F}{\sigma}, \frac{1}{\epsilon}, \frac{||A_i||}{||D_i||}, log(\frac{1}{\delta})))$

where δ is the probability of failure, $||A||_F$ is the Frobenius norm of A, $||A_i||$ is the ℓ^2 norm of row i in A, and similarly for $||D_i||$.

This theorem is essentially stating that, given a user i, we can output a list of preferences whose values are ϵ-bounded in size to the given user i preferences in A, and therefore this implies that the values of this fully-completed list of approximated recommendations are sufficiently close to what the user would actually rate the products he has not seen before.

Theorem 2: Applying theorem 1 to the recommendation systems model with a quantum state preparation data structure achieves the same bounds of quality recommendations as the quantum-based algorithm (that this classical algorithm was based off) up to constant factors and for a sufficiently small ϵ.

This theorem essentially explains how the classical algorithm we will be implementing can achieve the same bounds of accuracy as the quantum-version of this algorithm proposed, and combined with Theorem One, can do so in the same order of time. Note that it requires the usage of the same quantum state preparation structure, but this can be averted by implementing the lightweight Binary Search Tree (BST) we will discuss.

The algorithm works as follows:

1. Begins with an input matrix A stored in the BST data structure that suuports powerful ℓ^2-norm sampling operations in constant time.

2. Apply a subsampling strategy, called ModFKV, to find a low-rank approximation of A. This algorithm does not output the completed recommendation matrix in full, and rather outputs a succinct description of the matrix, which is a set of approximately orthonormal singular vectors $\hat{V}$,

where the low-rank approximation D of A then becomes $A\hat{V}\hat{V}^T$, which is the projection of the rows of A onto the low-dimensional subspace spanned by $\hat{V}$. Interestingly, we are able to sample from and query these singular vectors quickly.

3. Now given the desired low-rank approximation D, we look to sample from a particular row of it- representing extracting high-value entries from the a given user i's preferences. This is equivalent to sampling from $A_i\hat{V}\hat{V}^T$. To perform this sampling, we first estimate $A\hat{V}$, estimating dot products that can be done with the sampling access to A allowed through our data structure. We then use this output to perform the same operations and give an approximate sample from $A\hat{V}\hat{V}^T$. This sample is the desired output.

In the following sections we will discuss in more details the significant areas of this algorithm.

2.2 Data Structure

Since we are interested in achieving sublinear bounds for this classical algorithm, we must expedite the process by concerning ourselves with how the input is given.

If the input matrix $A \in \mathbb{R}^{mxn}$ is given in a classical unordered list of entries (i,j,A_{ij}), clearly linear time is required to parse through the input to alter it into a usable form. Even if this input is relatively structured (for example, given the entries of A sorted by row and column) then we still cannot sample the low-rank approximation of a generic matrix in sublinear time because of the time needed to locate non-zero entries.

Therefore, for our algorithm, we implemented the following data structure with low overhead cost.

- We built a binary search tree for a given vector $v \in \mathbb{R}^n$, which is equivalent to the row of user i's preferences towards n products.
- The n leaf nodes store, instead of v_i for $i \in [n]$, v_i^2 along with $sgn(v_i)$
- Building from this, each interior node stores the sum of the values of its two children. Ex: The parent node of v_0^2 and v_1^2 holds a value equivalent to $v_0^2 + v_1^2$. This allows for updates to be done quickly by updating all of the nodes above a particular leaf.
- Following this process, root node of a particular vector v simply holds $||v||^2$
- A significant characteristic of this data structure is its sampling efficiency. Sampling a high-value entry from D_v, which is the distribution on vector v, can simply be done by starting from the top node of the tree $||v||^2$ and randomly recursing on a child with the probability proportional to the child's weight. Note: We define $D_v(i) = \frac{v_i^2}{||v||^2}$

- This data structure allows for:
  1. Reading an updating an entry of the vector in O($log^2(n)$) time, where n is the number of components of the vector. This, as described above, is equivalent to updating all parent nodes above a particular leaf.
  2. Finding $||v||^2$ in O(1) time- simply pull the root node of the tree.
  3. Sampling from D_v in O($log^2(n)$) time again, by randomly recursing through all levels of a tree until you reach a leaf node.
- Now, when given a full m x n matrix A, we can extend this data structure to allow for the same operations in the same time constraints by building n BST's with the norms of each row squared as their roots, and then extending this BST by taking binary sums until the root node of this new BST is simply $||A||_F^2$.

2.3 ModFKV

ModFKV is the essentially the low-rank approximation algorithm that produces the matrix D from which we will sample finally, making it central to this writeup. Recall that ModFKV produces the D from its description by projecting input matrix A onto a subspace defined by vectors $\hat{V} \in \mathbb{R}^{nxk}$ where k is the small rank we mentioned earlier in this paper. That is,
$D = A\hat{V}\hat{V}^T = A\Sigma_{t=1}^{k}(\hat{V}^{(t)})(\hat{V}^{(t)})^T$
where $\hat{V}^{(t)}$ represents the t'th column vector in $\hat{V}$.

This description implies the columns of $\hat{V}$ to be a linear combination of rows of A. Let S be the submatrix of A by restricting the rows to $i_0...i_p$ for some p and then renormalizing row r by $\frac{1}{\sqrt{pD_{\tilde{A}(r)}}}$ where $\tilde{A}$ is equivalent to A restricted to the first p rows.

Then $\hat{V}^{(i)} := \frac{S^T u^{(i)}}{\sigma^{(i)}}$ where $u^{(i)}$ is the i'th left singular column vector of U given the singular value decomposition(SVD) $A = U\Sigma V$ (which can be produced trivially by already known algorithms.

In summary, what ModFKV does is subsamples a matrix, computes the large singular vectors of the matrix, and then outputs these, under the assumption that these singular vectors give a good description of the matrix.

ModFKV works, in full, as follows (Note: this is taken fully from Ewin Tang's Paper):

1. **Input:** Matrix $A \in \mathbb{R}^{mxn}$ supporting quick norm sampling operations outlined in Data Structures section, with a threshold σ (which described lower-bound value for entries of the sampled matrix), and error parameters ϵ and κ.
2. **Output** Description of D, low-rank approximation of A.
3. Set $K = \frac{||A||_F^2}{\sigma^2}$

4. Set $\hat{\epsilon} = \frac{\kappa\epsilon^2}{\sqrt{K}}$

5. Set $p = 10^7 max\{\frac{K^4}{\hat{\epsilon}^3}, \frac{K^2}{\hat{\epsilon}^4}\}$

6. Sample rows $i_1 ... i_p$ from $D_{\tilde{A}}$

7. Let F denote the distribution given by choosing an $s \sim_u [p]$, and choosing a column from $D_{\tilde{A}_{i_s}}$

8. Sample columns $j_1 ... j_p$ from F

9. Let the resulting p x p submatrix, with row r normalized by $\frac{1}{\sqrt{pD_{\tilde{A}(r)}}}$ and column c be normalized by $\frac{1}{\sqrt{pD_{\tilde{A}(c)}}}$, be denoted W.

10. Compute the left singular vectors of W $u^{(1)} ... u^{(k)}$ that correspond to singular values $\sigma^{(1)} ... \sigma^{(k)}$ larger than σ (again produced by SVD of A)

11. Output $i_1 ... i_p$, $u^{(1)} ... u^{(k)}$, and $\sigma^{(1)} ... \sigma^{(k)}$ as a description of output matrix D.

3 Testing and Drawbacks

A large drawback about this algorithm's applicability are its run-time constants. While for a significantly large input matrix A (say more than 10 million users and/or products), it can outperform most current singular value decomposition algorithms, for any test sets that must run in < 1 hour, this algorithm is not so practical.

Earlier in this writeup. We mentioned that the run time of this classical algorithm that produced a low-ranked sampling was:

O$(poly(\frac{||A||_F}{\sigma}, \frac{1}{\epsilon}, \frac{||A_i||}{|||D_i||}, log(\frac{1}{\delta})))$

The poly() function applied here omits one major subtlety- the actual degree of this polynomial. The true run time, presented below, is more informative of the drawbacks of this algorithm.

O$(max\{\frac{||A||^{30}}{\sigma^{30}\epsilon^{18}\kappa^6}, \frac{||A||^{24}}{\sigma^{24}\epsilon^{18}\kappa^8}\})$

where κ is sampling error.

As can be seen, with a max degree of 30, this algorithm poses a large barrier of computational time for numerous entries. Therefore, we were not able to display the outputs of our implementation on the Netflix Dataset as we intended, and, in fact, were not able to implement it on any meaningful dataset of user preferences. We were, however, able to test its accuracy on small, self-constructed matrices, and through that were assured that our implementation sufficed.