Spark and Matrix Factorization

Reza Zadeh
Problem

Data growing faster than processing speeds

Only solution is to parallelize on large clusters

> Wide use in both enterprises and web industry

How do we program these things?
Traditional Network Programming

Message-passing between nodes (e.g. MPI)

Very difficult to do at scale:

» How to split problem across nodes?
  • Must consider network & data locality
» How to deal with failures? (inevitable at scale)
» Even worse: stragglers (node not failed, but slow)
» Ethernet networking not fast
» Have to write programs for each machine

Rarely used in commodity datacenters
Spark Computing Engine

Extends a programming language with a distributed collection data-structure
   » “Resilient distributed datasets” (RDD)

Open source at Apache
   » Most active community in big data, with 50+ companies contributing

Clean APIs in Java, Scala, Python, R
Key Idea

Resilient Distributed Datasets (RDDs)

» Collections of objects across a cluster with user controlled partitioning & storage (memory, disk, …)
» Built via parallel transformations (map, filter, …)
» The world only lets you make make RDDs such that they can be:

   Automatically rebuilt on failure
A General Platform

Standard libraries included with Spark

Spark SQL structured
Spark Streaming real-time
GraphX graph
MLlib machine learning

Spark Core
Benefit of iterations: Optimization
Optimization

At least two large classes of optimization problems humans can solve:

- Convex Programs
- Spectral Problems (SVD)
Deep Dive: Singular Value Decomposition
Singular Value Decomposition

\[ A_{m \times n} = U_{m \times k} \Sigma_{k \times k} V^T_{k \times n} \]
Singular Value Decomposition

Two cases: Tall and Skinny vs roughly Square

computeSVD function takes care of which one to call, so you don’t have to.
Tall and Skinny SVD

- Given $m \times n$ matrix $A$, with $m \gg n$.
- We compute $A^T A$.
- $A^T A$ is $n \times n$, considerably smaller than $A$.
- $A^T A$ is dense.
- Holds dot products between all pairs of columns of $A$.

\[
A = U \Sigma V^T \quad A^T A = V \Sigma^2 V^T
\]
Tall and Skinny SVD

\[ A^T A = V \Sigma^2 V^T \]  \quad \text{Gets us \ V and the singular values}

\[ A = U \Sigma V^T \]  \quad \text{Gets us \ U by one matrix multiplication}
Square SVD via ARPACK

Very mature Fortran77 package for computing eigenvalue decompositions

JNI interface available via netlib-java

Distributed using Spark
Square SVD via ARPACK

Only needs to compute matrix vector multiplies to build Krylov subspaces

\[ K_n = [b \quad Ab \quad A^2b \quad \cdots \quad A^{n-1}b] \]

The result of matrix-vector multiply is small

The multiplication can be distributed
All pairs Similarity
All pairs Similarity

All pairs of similarity scores between n vectors

Compute via DIMSUM:

“Dimension Independent Similarity Computation using MapReduce”

Will be in Spark 1.2 as a method in RowMatrix
All-pairs similarity computation

- Given $m \times n$ matrix $A$, with $m \gg n$.

$$A = \begin{pmatrix}
a_{1,1} & a_{1,2} & \ldots & a_{1,n} \\
a_{2,1} & a_{2,2} & \ldots & a_{2,n} \\
& & \ddots & \\
a_{m,1} & a_{m,2} & \ldots & a_{m,n}
\end{pmatrix}$$

- $A$ is tall and skinny, example values $m = 10^{12}$, $n = 10^6$.
- $A$ has sparse rows, each row has at most $L$ nonzeros.
- $A$ is stored across hundreds of machines and cannot be streamed through a single machine.
Intuition

Sample columns that have many non-zeros with lower probability.

On the flip side, columns that have fewer non-zeros are sampled with higher probability.
Spark implementation

// Load and parse the data file.
val rows = sc.textFile(filename).map { line =>
    val values = line.split(' ').map(_.toDouble)
    Vectors.dense(values)
}
val mat = new RowMatrix(rows)

// Compute similar columns perfectly, with brute force.
val simsPerfect = mat.columnSimilarities()

// Compute similar columns with estimation using DIMSUM
val simsEstimate = mat.columnSimilarities(threshold)
Future of MLlib
General Linear Algebra

CoordinateMatrix
RowMatrix
BlockMatrix

Local and distributed versions.
Operations in-between.  

Goal: version 1.3
Spark and ML

Spark has all its roots in research, so we hope to keep incorporating new ideas!