Singular Value Decomposition

Reza Zadeh

@Reza_Zadeh | http://reza-zadeh.com

Optimization

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

» Convex

» Spectral

Distributed Singular Value Decomposition

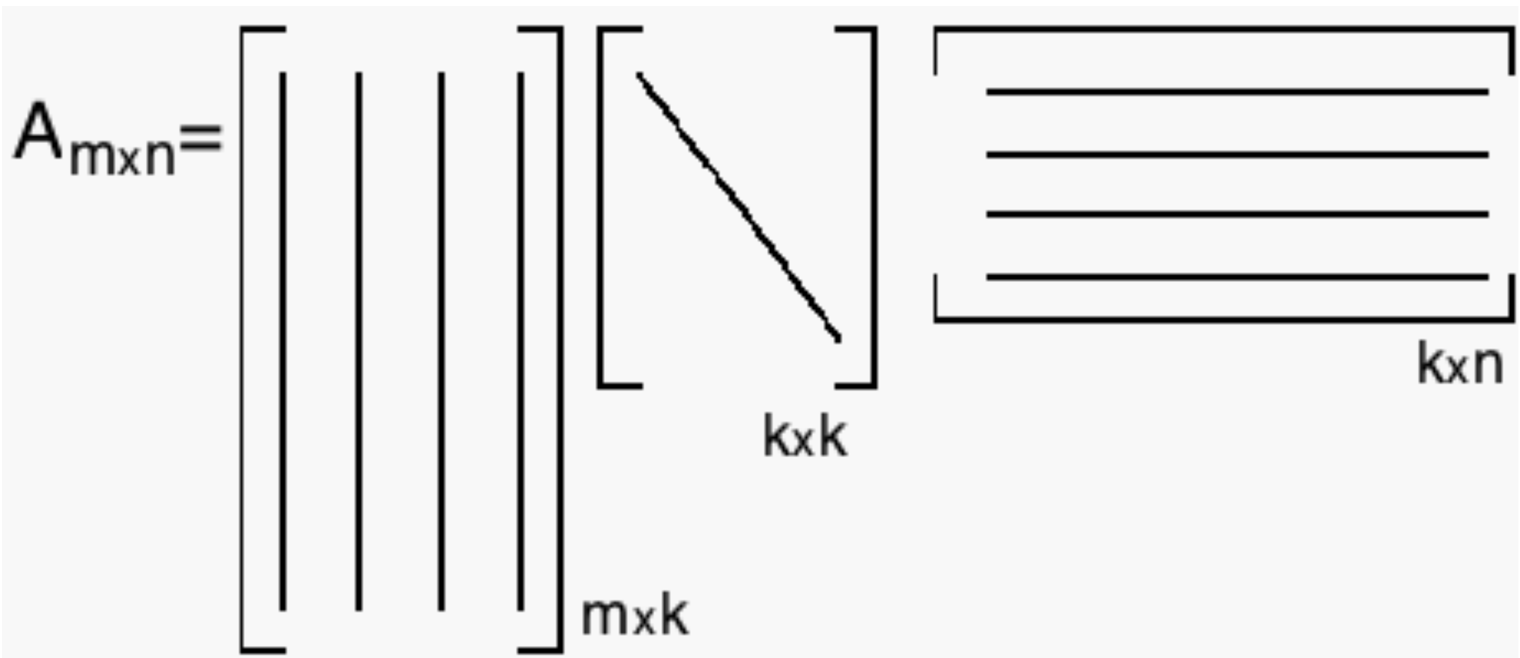
Distributing Matrices

How to distribute a matrix across machines?

- » By Entries (CoordinateMatrix)
- » By Rows (RowMatrix)
- » By Blocks (BlockMatrix) As of version 1.3

All of Linear Algebra to be rebuilt using these partitioning schemes

Singular Value Decomposition



Singular Value Decomposition

Two cases

- Tall and Skinny
- Short and Fat (not really)
- Roughly Square

SVD method on RowMatrix takes care of which one to call.

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.

$$A = U \Sigma V^T \qquad\qquad A^T A = V \Sigma^2 V^T$$

Tall and Skinny SVD

$$A^T A = V \Sigma^2 V^T$$

Gets us V and the singular values

$$A = U \Sigma V^T$$

Gets us U by one matrix multiplication

Square SVD

ARPACK: Very mature Fortran77 package for computing eigenvalue decompositions

JNI interface available via netlib-java

Distributed using Spark – how?

Square SVD via ARPACK

Only interfaces with distributed matrix via matrix-vector multiplies

$$K_n = \begin{bmatrix} b & Ab & A^2 b & \cdots & A^{n-1} b \end{bmatrix}$$

The result of matrix-vector multiply is small.

The multiplication can be distributed.

Square SVD

Matrix size	Number of nonzeros	Time per iteration (s)	Total time (s)
23,000,000 x 38,000	51,000,000	0.2	10
63,000,000 x 49,000	440,000,000	1	50
94,000,000 x 4,000	1,600,000,000	0.5	50

With 68 executors and 8GB memory in each, looking for the top 5 singular vectors

Optimization Example: Gradient Descent

ML Objectives

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

Scaling

1) Data size

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

2) Model size

3) Number of models

Data Scaling

$$w \leftarrow w - \alpha \cdot \sum_{i=1}^{n} g(w; x_i, y_i)$$

```
val points = spark.textFile(...).map(parsePoint).cache()
var w = Vector.zeros(d)
for (i <- 1 to numIterations) {
  val gradient = points.map { p =>
    (1 / (1 + exp(-p.y * w.dot(p.x)) - 1) * p.y * p.x
  ).reduce(_ + _)
  w -= alpha * gradient
}
```

Separable Updates

Can be generalized for

» Unconstrained optimization

» Smooth or non-smooth

» LBFGS, Conjugate Gradient, Accelerated Gradient methods, …

Model Scaling

Model is distributed (an RDD)

Linear Models only need dot products with training data computed (Block Matrices). How?

Model Scaling

More complicated models (e.g. large NN) need parameter servers

Lots of Models

Easy, often embarrassingly parallel

Shipping the work to the cluster is hardest part, but that’s usually taken care of by data-flow language