SVD Summary

Distributed Computing
School of Data Science
University of Virginia

Last updated: October 2, 2025

Matrix Factorization

SVD factors an *m x n* rectangular matrix into three matrices w special structure:

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

Matrix Factorization

SVD factors an *m x n* rectangular matrix into three matrices w special structure:

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

U is an orthonormal *m x m* matrix; its columns are called *left singular vectors*
Σ is a rectangular *m x n* diagonal matrix. It has nonnegative values in descending order
V is an orthonormal *n x n* matrix; its columns are called *right singular vectors*

Diagonal elements of $\sum$ called *singular values*

Dimension Reduction

Select top *k* singular values and associated singular vectors

This yields matrix approximation:

$$\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^T$$

Dimension Reduction

Select top *k* singular values and associated singular vectors

This yields matrix approximation:

$$\hat{A} = \hat{U}\hat{\Sigma}\hat{V}^T$$

$\hat{U}$ has dimensions *m x k*
$\hat{\Sigma}$ has dimensions *k x k*
$\hat{V}^T$ has dimensions *k x n*

These matrices can be substantially smaller
Only need to store diagonal elements of $\hat{\Sigma}$

SVD in Spark

Dataset stored as RowMatrix, which is distributed across partitions

Large computations (matrix-matrix, matrix-vector products across all data) are distributed

Small computations (eigenvalue, singular value) done on driver

SVD in Spark: Special Cases

Calculation breaks into special cases and general case.

> **Special Cases**: n is small $(n < 100)$ or k is large compared with n $(k > n/2)$

1. Compute Gramian in distributed manner

$$G = A^\top A \in \mathbb{R}^{n\times n}$$

- Feasible since *n* relatively small

- Each partition of A contributes to local $A_i^\top A_i$

- The dense Gramian is stored on **driver** (since it's small)

SVD in Spark: Special Cases

Step 2 done on driver

2. Decompose Gramian: $G = V \Lambda V^\top$

- V = eigenvectors of G,
- $\Lambda = \mathrm{diag}(\lambda_1, \ldots, \lambda_n)$ are eigenvalues.

Then $\Sigma = \mathrm{diag}(\sqrt{\lambda_1}, \ldots, \sqrt{\lambda_n})$

Note diagonal elements of $\sum$ are singular values σ_i

SVD in Spark: Special Cases

Step 3 done on workers

3. Compute left singular vectors $u_i = \frac{1}{\sigma_i} A v_i$

Now we have all of the factors to reconstruct the matrix

SVD in Spark: General Case

Gramian won't fit in driver memory, so earlier strategy won't work

Instead, at a high level, start with relationship $A^\top Av = \sigma^2 v$

SVD in Spark: General Case

Gramian won’t fit in driver memory, so earlier strategy won’t work

Instead, at a high level, start with relationship $A^\top Av = \sigma^2 v$

Then compute $(A^T A)v$ in a distributed way

Builds a **Krylov subspace** by repeatedly multiplying the matrix by a vector:

$$\mathcal{K}_k(A, v) = \text{span}\{v, Av, A^2v, \ldots, A^{k-1}v\}$$

Project onto subspace for small tridiagonal matrix *T*

SVD in Spark: General Case

Send *T* to ARPACK to compute top eigenvalues, eigenvectors on driver

ARPACK:

> FORTRAN software library for solving large-scale eigenvalue problems

> Highly optimized for sparse and large matrices

```
! svd_example.f90
!
! A modern Fortran example demonstrating how to use the LAPACK routine
! DGESVD to compute the Singular Value Decomposition (SVD) of a matrix.
!
! Link with a LAPACK and BLAS implementation:
! e.g., gfortran svd_example.f90 -llapack -lblas

program svd_solver
  implicit none

  ! Dimensions of the matrix A
  integer, parameter :: m = 4, n = 3

  ! Input matrix A and workspace for DGESVD.
  ! DGESVD will overwrite A with intermediate values.
  real(kind=8), dimension(m, n) :: a

  ! Singular values stored in s.
  real(kind=8), dimension(min(m, n)) :: s
```