Generalizing Discriminant Analysis Using the Generalized Singular Value Decomposition

Paper by Peg Howland and Haesun Park

Abhishek Kalokhe and Rajeshwari Devaramani
MS Computational Science and Engineering
Georgia Institute of Technology
December 4, 2023

1 Abstract

Discriminant analysis, a long-standing method for extracting features that maintain class separability, has traditionally relied on an optimization problem involving covariance matrices representing scatter within and between clusters. However, its application has been constrained by the necessity for one of these matrices to be nonsingular, limiting its use to datasets with specific dimensional characteristics. In this study, we explore various optimization criteria and broaden the applicability of discriminant analysis by leveraging the generalized singular value decomposition to overcome the requirement for nonsingularity. This results in an enhanced discriminant analysis approach that can be employed even when the sample size is smaller than the dimension of the sample data. To assess the effectiveness of this modified approach, we compare classification results from the reduced representation with alternative methods. We conclude with a discussion of the relative merits of these approaches.

2 Literature Review

To understand the implementation of the paper [2], we must have an understanding of the Generalized Singular Value Decomposition. We have two formulations of GSVD. The formulation by Van Loan [4] has assumed restrictions over the size of one of the matrices, whereas the formulation by Paige and Saunders [3] is a more general formulation which can be defined for any two matrices with the same number of columns.

2.1 Generalized Singlar Value Decomposition

2.1.1 Van Loan

Suppose two matrices $K_A \in \mathbb{R}^{p\times m}$ with $p \geq m$ and $K_B \in \mathbb{R}^{n\times m}$ are given. Then, there exist orthogonal matrices $U \in \mathbb{R}^{p\times p}$ and $V \in \mathbb{R}^{n\times n}$ and a nonsingular matrix $X \in \mathbb{R}^{m\times m}$ such that

$$U^T K_A X = \mathrm{diag}(\alpha_1, \dots, \alpha_m)$$

$$V^T K_B X = \mathrm{diag}(\beta_1, \dots, \beta_q)$$

where $q = \min(n, m)$, $\alpha_i \geq 0$ for $1 \leq i \leq m$, and $\beta_i \geq 0$ for $1 \leq i \leq q$.

2.1.2 Paige and Saunders

Given $K_A \in \mathbb{R}^{p\times m}$ and $K_B \in \mathbb{R}^{n\times m}$, there exist orthogonal matrices $U \in \mathbb{R}^{p\times p}$, $V \in \mathbb{R}^{n\times n}$, $W \in \mathbb{R}^{t\times t}$, and $Q \in \mathbb{R}^{m\times m}$ such that

$$U^T K_A Q = \Sigma_A (W^T R\,,\, 0)$$

$$V^T H^T w Q = \Sigma_B (W^T R\,,\, 0)$$

where

$$K = \begin{pmatrix} K_A \\ K_B \end{pmatrix}$$

and $t = \text{rank}(K)$, $R \in \mathbb{R}^{t \times t}$ is nonsingular with singular values equal to the nonzero singular values of K.

Relating to Van Loan's formulation: We get,

$$U^T K_A X = (\Sigma_A, 0) \quad \text{and} \quad V^T K_B X = (\Sigma_B, 0),$$

where,

$$X_{m \times m} = Q \begin{pmatrix} R^{-1}W & 0 \\ 0 & I \end{pmatrix}$$

.

where,

$$r = \text{rank}\begin{pmatrix} K_A \\ K_B \end{pmatrix} - \text{rank}(K_B) \quad \text{and} \quad s = \text{rank}(K_A) + \text{rank}(K_B) - \text{rank}\begin{pmatrix} K_A \\ K_B \end{pmatrix}$$

Therefore,

$$K_A^T K_A = X^{-T} \begin{pmatrix} \Sigma_A^T \Sigma_A & 0 \\ 0 & 0 \end{pmatrix} X^{-1}$$

and

$$K_B^T K_B = X^{-T} \begin{pmatrix} \Sigma_B^T \Sigma_B & 0 \\ 0 & 0 \end{pmatrix} X^{-1},$$

and

$$\beta_i^2 K_A^T K_A x_i = \alpha_i^2 K_B^T K_B x_i \quad \text{for} \quad 1 \le i \le t.$$

$$[1 \ge \alpha_{r+1} \ge \cdots \ge \alpha_{r+s} \ge 0, \quad 0 < \beta_{r+1} \le \cdots \le \beta_{r+s} < 1] \text{ and } \alpha^2 + \beta^2 = 1$$

3 Introduction

The goal of the paper is to combine the features i.e reduce the number of dimensions of the original data in a way that maintains the cluster structure of the data.

- **Assumption:** The data is clustered.
- **What we want to achieve?**

$$G^T : a \in \mathbb{R}^{m \times 1} \to y \in \mathbb{R}^{l \times 1}$$

We represent the vectorized dataset as matrix A:
$A = (A_1, A_2, ..., A_k)$ where $A_i \in \mathbb{R}^{m \times n_i}$ and $\sum_{i=1}^{k} n_i = n$. Here, the data vectors $a_1, a_2, ..., a_n$ are the columns of

matrix A. Let N_i denote the set of column indices that belong to cluster i. The centroid $c^{(i)}$ is computed by taking the average of the columns in cluster.

$$c^{(i)} = \frac{1}{n_i} \sum_{j \in N_i} a_j$$

and the global centroid is defined as,

$$c = \frac{1}{n} \sum_{j=1}^{n} a_j$$

We define scatter matrix S_W, S_B and S_M as:

$$S_W = \sum_{i=1}^{k} \sum_{j \in N_i} (a_j - c^{(i)})(a_j - c^{(i)})^T$$

$$S_B = \sum_{i=1}^{k} \sum_{j \in N_i} (c^{(i)} - c)(c^{(i)} - c)^T$$

$$= \sum_{i=1}^{k} n_i (c^{(i)} - c)(c^{(i)} - c)^T$$

$$S_M = \sum_{i=1}^{k} \sum_{j \in N_i} (a_j - c)(a_j - c)^T$$

The relation between them is defined as $S_M = S_B + S_W$.

As we apply G^T to the matrix A, it transforms the scatter matrices to $l \times l$ matrices,

$$S_W^Y = G^T S_W G, S_B^Y = G^T S_B G, S_M^Y = G^T S_M G$$

When cluster quality is high, each cluster is tightly grouped, but well separated from the other clusters.

$$\text{trace}(S_W) = \sum_{i=1}^{k} \sum_{j \in N_i} (a_j - c(i))^T (a_j - c(i)) = \sum_{i=1}^{k} \sum_{j \in N_i} \|a_j - c(i)\|^2$$

measures the closeness of the columns within the clusters, and

$$\text{trace}(S_B) = \sum_{i=1}^{k} \sum_{j \in N_i} (c(i) - c)^T (c(i) - c) = \sum_{i=1}^{k} \sum_{j \in N_i} \|c(i) - c\|^2$$

measures the separation between clusters.

Optimal transformation: maximize trace(S_Y^B) and minimize trace(S_Y^W)

$$\max_{\text{G}} \text{trace}((G^T S_2 G)^{-1} (G^T S_1 G))$$

4 Problem Formulation

We aim at solving the above mentioned problem while considering different S_1 and S_2 matrices from S_W, S_B and S_M. We want to find G such that the above mentioned trace is maximized, which in turn results in compact clusters as by maximizing the above problem we maximize the separation between the data points between the clusters and minimize the separation of data points within the cluster.

We are specifically interested in finding the solution for a special case when S_2 is singular and it's inverse does not exist.

5 Methods

5.1 Generalization of Linear Discriminant Analysis

We leverage the Generalized Singular Value Decomposition (GSVD) to broaden several criteria within discriminant analysis. Furthermore, we establish equivalence across different choices of scatter matrices.

5.1.1 Optimization of J_1 = trace($S_2^{-1} S_1$) Criteria

$$\textbf{Optimize } J_1(G) = \text{trace}((G^T S_2 G)^{-1} (G^T S_1 G))$$

over G, where S_1 and S_2 are chosen from S_W, S_B and S_M. Assume S_2 to be nonsingular, it is symmetric positive definite. There exists a nonsingular matrix $X \in \mathbb{R}^{m \times m}$

$$X^T S_1 X = \Lambda = \text{diag}(\lambda_1 ... \lambda_m) \text{ and } X^T S_2 X = I_m$$

(Symmetric Definite Generalized Eigenvalue Problem). Letting x_i denote the ith column of X, we have

$$S_1 x_i = \lambda_i S_2 x_i$$

which means λ_i and x_i are an eigenvalue-eigenvector pair of $S_2^{-1}S_1$. $\lambda_i \geq 0$ for $1 \leq i \leq m$ (S_1 is positive semidefinite) Largest $q = \text{rank}(S_1)$ λ_is are non-zero.

$$J_1(G) = \text{trace}(\tilde{G}^T\tilde{G})^{-1}\tilde{G}^T\Lambda\tilde{G})$$

where $\tilde{G} = X^{-1}G$. $\tilde{G}$ has full rank provided G does, so we can write $\tilde{G} = QR$, where $Q \in \mathbb{R}^{m\times l}$ has orthonormal columns and R is nonsingular. We get,

$$J_1(G) = \text{trace}(Q^T\Lambda Q)$$

Once we have simultaneously diagonalized S_1 and S_2, the maximization of $J_1(G)$ depends only on an orthonormal basis for range($X^{-1}G$), i.e,

$$\begin{aligned} \underset{G}{max} J_1(G) &= \underset{Q^TQ=I_l}{max} \text{trace}(Q^T\Lambda Q) \\ &\leq \lambda_1 + ... + \lambda_q \\ &= \text{trace}(S_2^{-1}S_1) \end{aligned}$$

For any l satisfying $l \geq q$, this upper bound on $J_1(G)$ is achieved for

$$Q = \begin{pmatrix} I_l \\ 0 \end{pmatrix} \text{ or } G = X\begin{pmatrix} I_l \\ 0 \end{pmatrix} R$$

Tranformation G is not unique as J_1 satisfies invariance property $J_1(G) = J_1(GW)$ for any nonsingular matrix $W \in \mathbb{R}^{l\times l}$. Hence, maximum $J_1(G)$ is also achieved for

$$G = X\begin{pmatrix} I_l \\ 0 \end{pmatrix}$$

This means that, for $l \geq \text{rank}(S_1)$,

$$\text{trace}((G^TS_2G)^{-1}G^TS_1G) = \text{trace}(S_2^{-1}S_1)$$

whenever $G \in \mathbb{R}^{m\times l}$ consists of l eigenvectors of $S_2^{-1}S_1$ corresponding to the l largest eigenvalues. According to our partitioning of A into k clusters, we define $m \times n$ matrices,

$$\begin{aligned} H_W &= (A_1 - c^{(1)}e^{(1)^T}, A_2 - c^{(2)}e^{(2)^T}, ..., A_k - c^{(k)}e^{(k)^T}) \\ H_B &= ((c^{(1)} - c)e^{(1)^T}, (c^{(2)} - c)e^{(2)^T}, ..., (c^{(k)} - c)e^{(k)^T}) \\ H_M &= (a_1 - c, ..., a_n - c) = A - ce^T = H_W + H_B \end{aligned}$$

where $e^{(i)} = (1, ..., 1)^T \in \mathbb{R}^{n_i\times 1}$ and $e = (1, ..., 1)^T \in \mathbb{R}^{n\times 1}$
Scatter matrices are expressed as

$$S_W = H_WH_W^T,\ S_B = H_BH_B^T,\ S_M = H_MH_M^T$$

J_1 cannot be applied when the number of available data vectors n is smaller than the dimension m of the data. Therefore, we generalize by expressing λ_i as α_i^2/β_i^2 in

$$S_1 x_i = \lambda_i S_2 x_i$$

to,

$$\beta^2 S_i x_i = \alpha_i^2 S_2 x_i$$

5.1.2 Generalization of J_1 = trace($S_2^{-1}S_1$) Criteria for Singular S_2

Case 1:

$$(S_1, S_2) = (S_B, S_W).$$

To approximate G that satisfies both

$$\max_G \text{trace}(G^T S_B G) \quad \text{and} \quad \min_G \text{trace}(G^T S_W G),$$

For nonsingular S_W, the generalized singular vectors are eigenvectors of $S_W^{-1}S_B$, so we choose the $x_i's$ which correspond to the k - 1 largest λ_i's, where $\lambda_i = \alpha_i^2/\beta_i^2$. When $m \geq n$, the scatter matrix S_W is singular. Hence, the eigenvectors of $S_W^{-1}S_B$ are undefined, and classical discriminant analysis fails. If a generalized singular vector x_i lies in the null space of S_W. From the above generalized equation, we see that either x_i also lies in the null space of S_B, or the corresponding β_i equals zero. When

$$x_i \in \text{null}(S_W) \cap \text{null}(S_B),$$

This will be the case for the rightmost m - t columns of X. To determine whether these columns should be included in G, consider

$$\text{trace}(G^T S_B G) = \sum_j g_j^T S_B g_j \quad \text{and} \quad \text{trace}(G^T S_W G) = \sum_j g_j^T S_W g_j,$$

where g_j represents the jth column of G. Since $x_i^T S_W x_i = 0$ and $x_i^T S_B x_i = 0$, adding the column x_i to G does not contribute to either maximization or minimization of the trace. So we do not those columns in G. When

$$x_i \in \text{null}(S_W) - \text{null}(S_B),$$

the generalized singular value α_i/β_i is infinite. The leftmost columns of X will correspond to these. Including these columns in G increases trace($G^T S_B G$), while leaving trace($G^T S_W G$) unchanged.

5.1.3 Equivalence of J_1 = trace($S_2^{-1}S_1$) Criteria for various S_2 and S_1

Case 2:

$$(S_1, S_2) = (S_M, S_W)$$

according to our previous analysis we would have to include rank(S_M) columns of X in G, which is not less than or equal to $k - 1$. However,

$$S_M x_i = \lambda_i S_W x_i$$

can be written as

$$S_B x_i = (\lambda_i - 1) S_W x_i, \text{ where } \lambda_i \geq 1 \text{ for } 1 \leq i \leq m$$

In this case, the eigenvector matrix is the same as for the case of $(S_1, S_2) = (S_B, S_W)$, but the eigenvalue matrix is $\Lambda - I$. Same permutation will put the $\Lambda - I$ in nonincreasing order as was used for Λ, and x_i corresponds to the ith largest eigenvalue of $S_W^{-1}S_B$, therefore, for nonsingular S_W, the solution is same as for $(S_1, S_2) = (S_B, S_W)$. When S_W is non-singular, in the m-dimensional space,

$$\text{trace}(S_W^{-1}S_M) = \text{trace}(S_W^{-1}(S_W + S_B))$$

$$= m + \text{trace}(S_W^{-1}S_B)$$

and, in l-dimensional space,

$$\text{trace}((S_W^Y)^{-1}S_M^Y) = \text{trace}((S_W^Y)^{-1}(S_W^Y + S_B^Y))$$
$$= l + \text{trace}((S_W^Y)^{-1}S_B^Y)$$

Subtracting the above two equations and as trace($S_W^{-1}S_B$) = trace($(S_W^Y)^{-1}(S_B^Y)$, we get

$$\text{trace}((S_W^Y)^-1S_M^Y) + (m - l) = \text{trace}(S_W^{-1}S_M)$$

These equations will help us validating our experimental results.

5.2 Alternative Approaches

5.2.1 Orthogonal Centroid

Introducing simpler criteria to preserve the cluster structure, we focus on utilizing only one of the scatter matrices, either minimizing the trace of $G^T S_W G$ or maximizing the trace of $G^T S_B G$. Notably, minimizing the trace of $G^T S_W G$ becomes impractical, rendering it meaningless, as the optimum consistently reduces the dimension to one. Even when subject to the constraint of G having orthonormal columns, this criterion remains problematic. Conversely, when maximizing the trace of $G^T S_B G$ under the same constraint, the solution aligns with the orthogonal centroid method, offering an equivalent outcome.
Let $J_2(G) = \text{trace}\left(G^T S_B G\right)$ and $G \in \mathbb{R}^{m\times l}$ has orthonormal columns, then there exists $\hat{G} \in \mathbb{R}^{m\times(m-l)}$ such that $\left[G, \hat{G}\right]$ is an orthogonal matrix.
Since S_B is positive semidefinite,

$$\text{trace}\left(G^T S_B G\right) \leq \text{trace}\left(G^T S_B G\right) + \text{trace}\left(\hat{G}^T S_B \hat{G}\right) = \text{trace}\left(S_B\right).$$

If SVD of H_B is given by $H_B = U\Sigma V^T$, then $S_B U = U\Sigma\Sigma^T$. Columns of U form an orthonormal set of eigenvectors of S_B corresponding to the nonincreasing eigenvalues λ_i on the diagonal of $\Lambda = \Sigma\Sigma^T$. For $q = \text{rank}\left(S_B\right)$, if we let U_q denote the first q columns of U and $\Lambda_q = \text{diag}\left(\lambda_1, \ldots, \lambda_q\right)$, we have

$$J_2\left(U_q\right) = \text{trace}\left(U_q^T S_B U_q\right) = \text{trace}\left(U_q^T U_q \Sigma_q\right) = \lambda_1 + \cdots + \lambda_q = \text{trace}\left(S_B\right)$$

We can see that trace(S_B) is preserved when we take U_q as G. We define a centroid matrix $C = (c^{(1)}, c^{(2)}, \ldots, c^{(k)})$. C has reduced QR decomposition $C = Q_k R$, where $Q_k \in \mathbb{R}^{m\times k}$ has orthonormal columns and $R \in \mathbb{R}^{k\times k}$. Let x be an eigenvector corresponding to nonzero eigenvalue λ, then

$$S_B x = \sum_{i=1}^{k} n_i (c^{(i)} - c)(c^{(i)} - c)^T x = \lambda x$$

which means $x \in \text{span}\{c^{(i)} | 1 \leq i \leq k\}$ Hence, we have range$(U_q) \subseteq$ range$(C) \subseteq$ range(Q_k), which implies that $U_q = Q_k W$ for some matrix $W \in \mathbb{R}^{k\times q}$ with orthonormal columns. We get,

$$J_2(U_q) = \text{trace}(W^T Q_k^T S_B Q_k W) \leq \text{trace}(Q_k^T S_B Q_k) = J_2(Q_k)$$

Hence, $J_2(Q_k) = \text{trace}(S_B)$ and therefore by computing reduced QR of the centroid matrix, we obtain a solution that maximizes the trace$(G^T S_B G)$ over all G with orthonormal columns.

5.2.2 Two-Stage Approach

This is an another approach for dealing with the singularity of S_W when $m > n$. As the name suggests, this approach works in two stages.

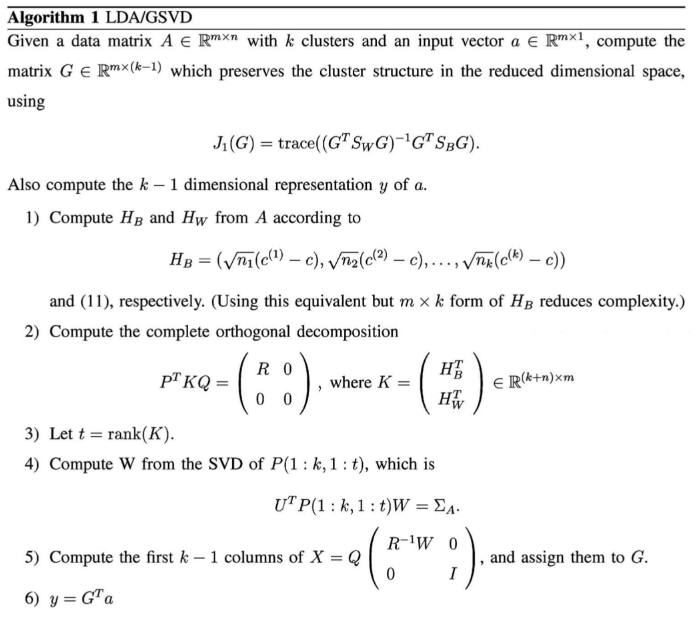
1. Using LSI/SVD, reduce the dimension of the data enough so that the new S_W is nonsingular.
2. Perform classical LDA.

Truncated SVD is used to find rank-l approximation of A. If $l \leq \text{rank}(A)$, then

$$A \approx U_l \Sigma_l V_l^T$$

LSI/SVD uses $\Sigma_l V_l^T$ as the reduced dimensional representation of A or equivalently computes the l-dimensional representation of $a \in \mathbb{R}^{m\times 1}$ as $y = U_l^T a$. We won't be implementing this method in our experiments.

6 Algorithms



Algorithm 1 LDA/GSVD

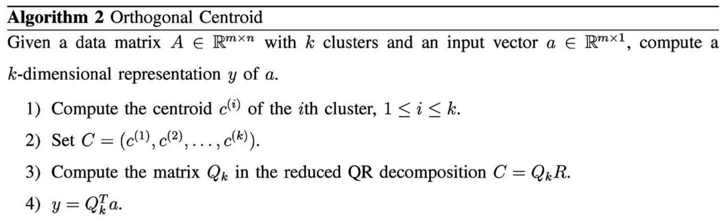
Given a data matrix $A \in \mathbb{R}^{m \times n}$ with k clusters and an input vector $a \in \mathbb{R}^{m \times 1}$, compute the matrix $G \in \mathbb{R}^{m \times (k-1)}$ which preserves the cluster structure in the reduced dimensional space, using

$$J_1(G) = \text{trace}((G^T S_W G)^{-1} G^T S_B G).$$

Also compute the $k-1$ dimensional representation y of a.

```
1) Compute $H_B$ and $H_W$ from $A$ according to
       $H_B = (\sqrt{n_1}(c^{(1)} - c), \sqrt{n_2}(c^{(2)} - c), \ldots, \sqrt{n_k}(c^{(k)} - c))$
   and (11), respectively. (Using this equivalent but $m \times k$ form of $H_B$ reduces complexity.)
2) Compute the complete orthogonal decomposition
       $P^T K Q = \begin{pmatrix} R & 0 \\ 0 & 0 \end{pmatrix}$, where $K = \begin{pmatrix} H_B^T \\ H_W^T \end{pmatrix} \in \mathbb{R}^{(k+n) \times m}$
3) Let $t = \text{rank}(K)$.
4) Compute W from the SVD of $P(1:k, 1:t)$, which is
       $U^T P(1:k, 1:t) W = \Sigma_A$.
5) Compute the first $k-1$ columns of $X = Q \begin{pmatrix} R^{-1}W & 0 \\ 0 & I \end{pmatrix}$, and assign them to $G$.
6) $y = G^T a$
```

Figure 1: LDA/GSVD



Algorithm 2 Orthogonal Centroid

Given a data matrix $A \in \mathbb{R}^{m \times n}$ with k clusters and an input vector $a \in \mathbb{R}^{m \times 1}$, compute a k-dimensional representation y of a.

```
1) Compute the centroid $c^{(i)}$ of the $i$th cluster, $1 \le i \le k$.
2) Set $C = (c^{(1)}, c^{(2)}, \ldots, c^{(k)})$.
3) Compute the matrix $Q_k$ in the reduced QR decomposition $C = Q_k R$.
4) $y = Q_k^T a$.
```

Figure 2: Orthogonal Centroid

7 Experimental Setting

Dataset Used: Department of Justice 2009-2018 Press Releases [1].

Used the TfIdf Vectorizer which converts a collection of raw documents to a matrix of TF-IDF features. We have 300 documents (samples) with 1000 features for the undersampled case and we have 900 documents (samples) with 100 features for the oversampled case.

Setup:

- We have 3 clusters in the sampled dataset.
- For the oversampled data, we implement GSVD and reduce the dimensions of the data to 2 (number of clusters - 1) and also calculate the traces of the scatter matrices to compare it with the full implentation.
- For the undersampled data, we calculate the traces of the scatter matrices using the GSVD, orthogonal centroid method and the full implementation. We also plot the GSVD implementation in 2D along with the PCA (2D) to observe the differences in the cluster structures.

8 Results

We present the results of the experiment that we performed on the chosen dataset.

Method	**Full**	**GSVD**
Dim	100×900	2×900
trace (S_W)	461.109	0.16
trace (S_B)	150.281	1.839
trace (S_M)	611.390	1.999
trace $(S_W^{-1} S_B)$	30.078	30.078
trace$(S_W^{-1} S_M)$	130.078	32.078

Table 1: Oversampled Data (Non singular S_W)

Method	**Full**	**Orthogonal Centroid**	**GSVD**
Dim	1000×300	3×300	2×300
trace (S_W)	179.33	5.46	7.108e-18
trace (S_B)	35.288	35.288	1.999
$\frac{\text{trace}(S_B)}{\text{trace}(S_W)}$	0.196	6.463	2.81e+17

Table 2: Undersampled Data (Singular S_W)

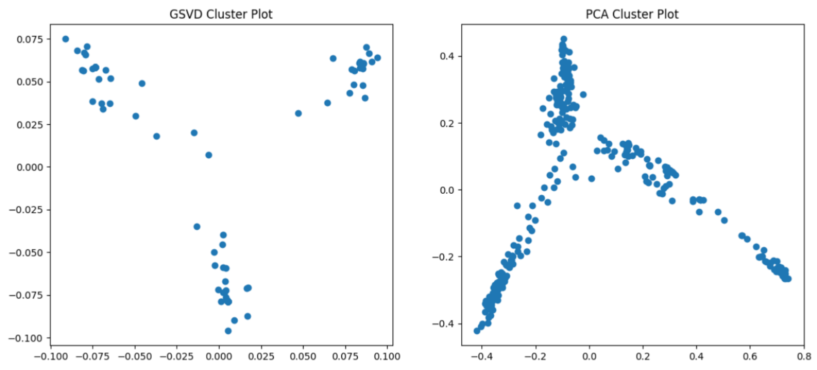


Figure 3: Comparison of GSVD clusters (left) with the PCA (right)

9 Conclusions

- LDA/GSVD extends the applicability to cases that classical discriminant analysis cannot handle.
- In addition, LDA/GSVD algorithm never explicitly forms the scatter matrices, which results in two advantages.
  - We avoid the numerical problems inherent in forming cross-product matrices.
  - We reduce the storage requirements considerably.
- Orthogonal decomposition is found to be cheaper than LDA/GSVD as we just need to compute Q_k instead of computing the eigenvectors.
- Compared to the two-stage approach of LSI followed by LDA, the one-stage LDA/GSVD avoids the potentially costly experimentation involved in determining the dimension for LSI.

References

[1] JOHN B. Department of justice 2009-2018 press releases, 2018.

[2] P. Howland and H. Park. Generalizing discriminant analysis using the generalized singular value decomposition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(8):995–1006, aug 2004.

[3] Christopher C Paige and Michael A Saunders. Towards a generalized singular value decomposition. *SIAM Journal on Numerical Analysis*, 18(3):398–405, 1981.

[4] Charles Van Loan. Computing the cs and the generalized singular value decompositions. *Numerische Mathematik*, 46(4):479–491, 1985.

Code

```python
# -*- coding: utf-8 -*-
"""NDA Project.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1Epol1hK69fDjcbagPl2bxA6Ihpj0Ox7K

#Data Cleaning (Used different clustering methods to get the clusters)
"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

from sklearn.cluster import MiniBatchKMeans, KMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.manifold import TSNE
from google.colab import drive

"""Original Dataset"""

data = pd.read_json('/content/drive/MyDrive/combined.json', lines=True)
data.head()

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.5,
    max_features = 150,
    stop_words = 'english'
)
tfidf.fit(data.contents)
text = tfidf.transform(data.contents)

def find_optimal_clusters(data, max_k):
    iters = range(2, max_k+1, 4)

    sse = []
    for k in iters:
        sse.append(KMeans(n_clusters=k, random_state=20).fit(data).inertia_)
        print('Fit {} clusters'.format(k))

    f, ax = plt.subplots(1, 1)
    ax.plot(iters, sse, marker='o')
    ax.set_xlabel('Cluster Centers')
    ax.set_xticks(iters)
    ax.set_xticklabels(iters)
    ax.set_ylabel('SSE')
    ax.set_title('SSE by Cluster Center Plot')

find_optimal_clusters(text, 40)
```

```python

# clusters = MiniBatchKMeans(n_clusters=3, init = 'random', init_size=115, batch_size=2048, random_state=20).fit_predict(text)
# from sklearn.cluster import AgglomerativeClustering
clusters = KMeans(n_clusters=20, random_state=20).fit_predict(text)
# clusters = kmeans(n_clusters=20).fit_predict(text.toarray())

def plot_tsne_pca(data, labels):
    max_label = max(labels)
    max_items = np.random.choice(range(data.shape[0]), size=3000, replace=False)

    pca = PCA(n_components=2).fit_transform(data[max_items,:].toarray())
    tsne = TSNE().fit_transform(PCA(n_components=50).fit_transform(data[max_items,:].toarray()))


    idx = np.random.choice(range(pca.shape[0]), size=300, replace=False)
    label_subset = labels[max_items]
    label_subset = [cm.hsv((i+1)/(max_label+1)) for i in label_subset[idx]]

    f, ax = plt.subplots(1, 2, figsize=(14, 6))

    ax[0].scatter(pca[idx, 0], pca[idx, 1], c=label_subset)
    ax[0].set_title('PCA Cluster Plot')

    ax[1].scatter(tsne[idx, 0], tsne[idx, 1], c=label_subset)
    ax[1].set_title('TSNE Cluster Plot')

# print(text)
# print(clusters)
plot_tsne_pca(text, clusters)

import sys
np.set_printoptions(threshold=sys.maxsize)
def get_top_keywords(data, clusters, labels, n_terms):
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()
    # print(df)
    # print(clusters.shape)
    for i,r in df.iterrows():
        # print(i)
        # print(r)
        print('\nCluster {}'.format(i))
        print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))
        # pass
get_top_keywords(text, clusters, tfidf.get_feature_names_out(), 10)

data.head()

data['clusterid']=clusters
data.head()

selected_clusters = [0, 1, 5]
filtered_df = data[data['clusterid'].isin(selected_clusters)]
print(filtered_df.shape)
# Sample 100 entries from each cluster
sampled_df = filtered_df.groupby('clusterid').apply(lambda x: x.sample(300))

# Reset the index to get a clean DataFrame
```

```python
sampled_df = sampled_df.reset_index(drop=True)

sampled_df.head()

print(sampled_df.shape)

print(filtered_df.shape)

csv_path = '/content/drive/MyDrive/oversampled_data.csv'

# Export the DataFrame to a CSV file
sampled_df.to_csv(csv_path, index=False)

# Print a message to confirm the export
print(f"Data exported to {csv_path}")

"""Undersampled data was generated similarly by just changing the number of samples and features in the above code."""



"""#Under SAMPLED DATA"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.manifold import TSNE
from google.colab import drive

data1 = pd.read_csv('/content/drive/MyDrive/filtered_data.csv')
data1.head()

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.5,
    max_features = 1000,
    stop_words = 'english'
)
tfidf.fit(data1.contents)
text1 = tfidf.transform(data1.contents)
print(text1)

array_0 = np.zeros(100)
array_1 = np.ones(100)
array_5 = np.full(100, 5)
clusters1 = np.concatenate([array_0, array_1, array_5])

print(clusters1)

# # clusters = MiniBatchKMeans(n_clusters=3, init = 'random', init_size=115, batch_size=2048, random_state=20).fit_predict(text)
# from sklearn.cluster import KMeans
# kmeans = KMeans(n_clusters=3, random_state=1, n_init="auto").fit(text1)
# clusters = kmeans.labels_
# print(clusters)

def get_top_keywords(data, clusters, labels, n_terms):
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()
    # print(df)
    # print(clusters.shape)
    for i,r in df.iterrows():
        # print(i)
        # print(r)
        print('\nCluster {}'.format(i))
        print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))
        # pass
get_top_keywords(text1, clusters1, tfidf.get_feature_names_out(), 10)

# print(clusters1.shape)

"""#GSVD Implementation"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.manifold import TSNE
from google.colab import drive

clustered_data = pd.read_csv('/content/drive/MyDrive/filtered_data.csv')
clustered_data.head()

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.5,
    max_features = 1000,
    stop_words = 'english'
)
tfidf.fit(clustered_data.contents)
text_ = tfidf.transform(clustered_data.contents)
print(text_.T.shape)
clusters = clustered_data['clusterid'].tolist()
print(clusters)

A = text_.T
print(A)

"""###3 Clusters A1, A2, A3"""

A1 = A[:, 0:80]
A1 = A1.toarray()
A2 = A[:, 100:180]
A2 = A2.toarray()
A3 = A[:, 200:280]
A3 = A3.toarray()
```

```

# print(A1)
a1_test = A[:, 80:100]
a1_test = a1_test.toarray()
a2_test = A[:, 180:200]
a2_test = a2_test.toarray()
a3_test = A[:, 280:300]
a3_test = a3_test.toarray()

Atrain = np.hstack([A1,A2,A3])
# print(A.shape)

size_cluster = 80
size_test_cluster = 20

# print(A1.toarray())
c1 = np.matmul(A1,np.ones((size_cluster,1)))/size_cluster
c2 = np.matmul(A2,np.ones((size_cluster,1)))/size_cluster
c3 = np.matmul(A3,np.ones((size_cluster,1)))/size_cluster
c = np.matmul(Atrain,np.ones((3*size_cluster,1)))/(3*size_cluster)
e1 = np.ones((size_cluster, 1))

S_m = np.matmul(Atrain - c, (Atrain-c).T)

S_w = np.matmul((A1 - c1),(A1-c1).T) + np.matmul((A2 - c2),(A2-c2).T) + np.matmul((A3 - c3),(A3-c3).T)
print("Trace Sw = ", np.trace(S_w))

S_b = size_cluster*(np.matmul(c1 - c, (c1 - c).T) + np.matmul(c2 - c, (c2 - c).T) + np.matmul(c3 - c, (c3 - c).T))
print("Trace Sb = ", np.trace(S_b))

Hb = np.hstack([np.sqrt(size_cluster)*(c1 - c),np.sqrt(size_cluster)*(c2 - c),np.sqrt(size_cluster)*(c3 - c)])
# print("Hb dim= ",Hb.shape)

Hw = np.hstack([A1 - (c1@e1.T),A2 - (c2@e1.T),A3 - (c3@e1.T)])
# print("Hw dim= ",Hw.shape)

K = np.vstack([Hb.T,Hw.T])
print(K.shape)
# S_M = S_w + S_b
# print(S_m[0])
# print(S_M[0])

import matplotlib.pyplot as plt
plt.figure(figsize=(15, 7))  # You can adjust the values (width, height) as needed
plt.imshow(A1, cmap='YlGnBu', aspect='auto')  # Use aspect='auto' to prevent distortion
plt.colorbar()
plt.title('Sparse Matrix Heatmap')
plt.show()



"""###Orthogonal Centroid"""

C = np.hstack([c1,c2,c3])
print(C.shape)

```

```python
Q_red, R_red = np.linalg.qr(C, mode='reduced')

print(R_red.shape)

y = Q_red.T@A
print(y.shape)

traceSbcenQR = np.trace(Q_red.T@S_b@Q_red)
print(traceSbcenQR)

traceSwcenQR = np.trace(Q_red.T@S_w@Q_red)
print(traceSwcenQR)

# from sklearn.neighbors import KNeighborsClassifier
# Knn = KNeighborsClassifier(n_neighbors=1).fit(y.T, clusters)
# a1test = Q_red.T@A
# clusters_cenqr = Knn.predict(y.T)
# print(clusters_cenqr)

"""###LDA/GSVD"""

K = np.vstack([Hb.T,Hw.T])
k = 3

U, S, Vh = np.linalg.svd(K, full_matrices=False)

Pt = U.T
Q = Vh.T
t = np.linalg.matrix_rank(K, tol=None, hermitian=False)
ptkq = Pt@K@Q
R = ptkq[0:t,0:t]
print(Q.shape)
R_inv = np.linalg.inv(R)

P = Pt.T
P = P[0:k,0:t]
U, S, Vh = np.linalg.svd(P, full_matrices=False)
W = Vh.T


RinvW = R_inv @ W
zeroes = np.zeros((4,3))
RinvWstacked = np.vstack([RinvW, zeroes])
X = Q@RinvWstacked
G = X[:,0:k-1]

traceSw = np.trace(G.T@S_w@G)
print("Trace Sw = ",traceSw)
traceSb = np.trace(G.T@S_b@G)
print("Trace Sb = ",traceSb)

# print(G.T@A)
Areduced = G.T@A

f, ax = plt.subplots(1, 2, figsize=(14, 6))

```

```python
ax[0].scatter(Areduced[0], Areduced[1])
ax[0].set_title('GSVD Cluster Plot')


pca = PCA(n_components=2).fit_transform(np.asarray(A.T.todense()))
idx = np.random.choice(range(pca.shape[0]), size=300, replace=False)

ax[1].scatter(pca[idx, 0], pca[idx, 1])
ax[1].set_title('PCA Cluster Plot')

# from sklearn.neighbors import KNeighborsClassifier
# knn = KNeighborsClassifier(n_neighbors=10).fit(Areduced.T, clusters)
# a1test = G.T@a3_test
# clusters_gsvd = knn.predict(Areduced.T)
# print(clusters_gsvd)
# print(clusters)



# def get_top_keywords(data, clusters, labels, n_terms):
#     df = pd.DataFrame(data.todense()).groupby(clusters).mean()
#     # print(df)
#     # print(clusters.shape)
#     for i,r in df.iterrows():
#         # print(i)
#         # print(r)
#         print('\nCluster {}'.format(i))
#         print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))
#         # pass
# get_top_keywords(text1, clusters, tfidf.get_feature_names_out(), 10)

"""### Eigenvectors of £S_w^{-1}S_b£"""

# import scipy as sp
# sw_inv_sb = np.matmul(np.linalg.inv(S_w),S_b)
# eigvalues, eigvectors = np.linalg.eig(sw_inv_sb)

# # eigvalues = sp.linalg.eigh(S_b, S_w)

# B = np.array([[1, 2, 3],
#               [2, 5, 6],
#               [3, 6, 9]])

# try:
#     # Attempting Cholesky decomposition
#     L = np.linalg.cholesky(S_w)
#     print("Cholesky decomposition successful.")
# except np.linalg.LinAlgError as e:
#     print(f"LinAlgError: {e}")

# import sys
# np.set_printoptions(threshold=sys.maxsize)
# print(np.real(eigvalues[0]), np.real(eigvalues[1]), np.real(eigvectors[0]), np.real(eigvectors[1]),)



```

```python



"""#Over_Sampled Data"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.manifold import TSNE
from google.colab import drive

data1 = pd.read_csv('/content/drive/MyDrive/oversampled_data.csv')
data1.head()

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.5,
    max_features = 50,
    stop_words = 'english'
)
tfidf.fit(data1.contents)
text1 = tfidf.transform(data1.contents)
# print(text1)

array_0 = np.zeros(300)
array_1 = np.ones(300)
array_5 = np.full(300, 5)
clusters1 = np.concatenate([array_0, array_1, array_5])

# print(clusters1)

def get_top_keywords(data, clusters, labels, n_terms):
    df = pd.DataFrame(data.todense()).groupby(clusters).mean()
    # print(df)
    # print(clusters.shape)
    for i,r in df.iterrows():
        # print(i)
        # print(r)
        print('\nCluster {}'.format(i))
        print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))
        # pass
get_top_keywords(text1, clusters1, tfidf.get_feature_names_out(), 10)

"""##GSVD Implementation"""

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import matplotlib.cm as cm

```

```python
from sklearn.cluster import MiniBatchKMeans
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.decomposition import PCA, TruncatedSVD
from sklearn.manifold import TSNE
from google.colab import drive

clustered_data = pd.read_csv('/content/drive/MyDrive/oversampled_data.csv')
clustered_data.head()

tfidf = TfidfVectorizer(
    min_df = 5,
    max_df = 0.5,
    max_features = 100,
    stop_words = 'english'
)
tfidf.fit(clustered_data.contents)
text_ = tfidf.transform(clustered_data.contents)
print(text_.T.shape)
clusters = clustered_data['clusterid'].tolist()
print(clusters)

"""######CLusters Ai and other calculations"""

A = text_.T
A1 = A[:, 0:260]
A1 = A1.toarray()
A2 = A[:, 300:560]
A2 = A2.toarray()
A3 = A[:, 600:860]
A3 = A3.toarray()

# print(A1)
a1_test = A[:, 260:300]
a1_test = a1_test.toarray()
a2_test = A[:, 560:600]
a2_test = a2_test.toarray()
a3_test = A[:, 860:900]
a3_test = a3_test.toarray()

Atrain = np.hstack([A1,A2,A3])
# print(A.shape)

size_cluster = 260
size_test_cluster = 40

# print(A1.toarray())
c1 = np.matmul(A1,np.ones((size_cluster,1)))/size_cluster
c2 = np.matmul(A2,np.ones((size_cluster,1)))/size_cluster
c3 = np.matmul(A3,np.ones((size_cluster,1)))/size_cluster
c = np.matmul(Atrain,np.ones((3*size_cluster,1)))/(3*size_cluster)
e1 = np.ones((size_cluster, 1))

S_m = np.matmul(Atrain - c, (Atrain-c).T)
print("Trace Sm = ", np.trace(S_m))

S_w = np.matmul((A1 - c1),(A1-c1).T) + np.matmul((A2 - c2),(A2-c2).T) + np.matmul((A3 - c3),(A3-c3).T)
```

```python
print("Trace Sw = ", np.trace(S_w))

S_b = size_cluster*(np.matmul(c1 - c, (c1 - c).T) + np.matmul(c2 - c, (c2 - c).T) + np.matmul(c3 - c, (c3 - c).T))
print("Trace Sb = ", np.trace(S_b))

SwInv = np.linalg.inv(S_w)
print("Trace S_wInv_Sb = ", np.trace(SwInv@S_b))
print("Trace S_wInv_Sm = ", np.trace(SwInv@S_m))

Hb = np.hstack([np.sqrt(size_cluster)*(c1 - c),np.sqrt(size_cluster)*(c2 - c),np.sqrt(size_cluster)*(c3 - c)])
# print("Hb dim= ",Hb.shape)

Hw = np.hstack([A1 - (c1@e1.T),A2 - (c2@e1.T),A3 - (c3@e1.T)])
# print("Hw dim= ",Hw.shape)

K = np.vstack([Hb.T,Hw.T])
print(K.shape)
# S_M = S_w + S_b
# print(S_m[0])
# print(S_M[0])

import matplotlib.pyplot as plt
plt.figure(figsize=(15, 7))  # You can adjust the values (width, height) as needed
plt.imshow(A1, cmap='YlGnBu', aspect='auto')  # Use aspect='auto' to prevent distortion
plt.colorbar()
plt.title('Sparse Matrix Heatmap')
plt.show()

"""###Orthogonal Centroid"""

# C = np.hstack([c1,c2,c3])
# print(C.shape)

# Q_red, R_red = np.linalg.qr(C, mode='reduced')

# print(R_red.shape)

# y = Q_red.T@A
# print(y.shape)

# traceSbcenQR = np.trace(Q_red.T@S_b@Q_red)
# print(traceSbcenQR)

# traceSwcenQR = np.trace(Q_red.T@S_w@Q_red)
# print(traceSwcenQR)

# from sklearn.neighbors import KNeighborsClassifier
# Knn = KNeighborsClassifier(n_neighbors=1).fit(y.T, clusters)
# a1test = Q_red.T@A
# clusters_cenqr = Knn.predict(y.T)
# print(clusters_cenqr)



"""###LDA/GSVD"""

```

```
K = np.vstack([Hb.T,Hw.T])
k = 3

U, S, Vh = np.linalg.svd(K, full_matrices=False)

Pt = U.T
Q = Vh.T
t = np.linalg.matrix_rank(K, tol=None, hermitian=False)
ptkq = Pt@K@Q
R = ptkq[0:t,0:t]
print(Q.shape)
R_inv = np.linalg.inv(R)

P = Pt.T
P = P[0:k,0:t]
U, S, Vh = np.linalg.svd(P, full_matrices=False)
W = Vh.T

RinvW = R_inv @ W
print(RinvW.shape)
# zeroes = np.zeros((4,3))
# RinvWstacked = np.vstack([RinvW, zeroes])
RinvWstacked = RinvW
X = Q@RinvWstacked
G = X[:,0:k-1]

SwGsvd = G.T@S_w@G
SbGsvd = G.T@S_b@G
SmGsvd = G.T@S_m@G
traceSw = np.trace(SwGsvd)
print("Trace Sw = ",traceSw)
traceSb = np.trace(SbGsvd)
print("Trace Sb = ",traceSb)
traceSm = np.trace(SmGsvd)
print("Trace Sm = ",traceSm)
SwInvGsvd = np.linalg.inv(SwGsvd)

traceSwInvSb = np.trace(SwInvGsvd@SbGsvd)
print("Trace SwInvSb = ",traceSwInvSb)
traceSwInvSm = np.trace(SwInvGsvd@SmGsvd)
print("Trace SwInvSm = ",traceSwInvSm)


# print(G.T@A)
Areduced = G.T@A

f, ax = plt.subplots(1, 2, figsize=(14, 6))

ax[0].scatter(Areduced[0], Areduced[1])
ax[0].set_title('GSVD Cluster Plot')
print(A.shape)

pca = PCA(n_components=2).fit_transform(np.asarray(A.T.todense()))
idx = np.random.choice(range(pca.shape[0]), size=900, replace=False)

ax[1].scatter(pca[idx, 0], pca[idx, 1])
```

```
ax[1].set_title('PCA Cluster Plot')

from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=10).fit(Areduced.T, clusters)
a1test = G.T@a3_test
clusters_gsvd = knn.predict(Areduced.T)
print(clusters_gsvd)
print(clusters)

# def get_top_keywords(data, clusters, labels, n_terms):
#     df = pd.DataFrame(data.todense()).groupby(clusters).mean()
#     # print(df)
#     # print(clusters.shape)
#     for i,r in df.iterrows():
#         # print(i)
#         # print(r)
#         print('\nCluster {}'.format(i))
#         print(','.join([labels[t] for t in np.argsort(r)[-n_terms:]]))
#         # pass
# get_top_keywords(text1, clusters, tfidf.get_feature_names_out(), 10)
```