Optimization of Discriminant Ratio (J)

Theoretical Analysis, Implementation, and Proof of Equivalence

Deep Metric Learning Lab

December 4, 2025

Abstract

This document analyzes the *Discriminant Ratio* (J) metric, used to evaluate the quality of class (cluster) separation in latent spaces. We will compare the classical (Naive) approach, which is computationally expensive ($\mathcal{O}(d^2)$), with an optimized approach based on Trace properties ($\mathcal{O}(d)$). Finally, we provide a **detailed mathematical proof** demonstrating the exact equivalence of the two methods.

1 The Problem: Computational Cost

The Discriminant Ratio is defined as the ratio between the *between-class* scatter (S_B) and the *within-class* scatter (S_W):

$$J = \frac{\text{Tr}(S_B)}{\text{Tr}(S_W)} \tag{1}$$

In Deep Learning contexts with high-dimensional embeddings (e.g., $d = 4096$), the explicit calculation of covariance matrices $S_B, S_W \in \mathbb{R}^{4096 \times 4096}$ entails:

1. **Memory Waste:** Allocating a float32 matrix requires approximately 64 MB. Multiplied by every batch or class, this saturates VRAM.
2. **Slowness:** The matrix product is slow.

The objective is to compute J **without ever constructing these matrices**.

2 Approach 1: Naive Implementation (Matrix-based)

The textbook approach follows the *Linear Discriminant Analysis* (LDA) formula.

2.1 Definitions

- μ: Global mean of all data.
- μ_c: Mean of data belonging to class c.
- N_c: Number of elements in class c.

The matrices are calculated by summing the outer products:

$$S_W = \sum_{c \in C} \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T \tag{2}$$

$$S_B = \sum_{c \in C} N_c (\mu_c - \mu)(\mu_c - \mu)^T \tag{3}$$

2.2 Python Code (Critical)

```
# Explicit construction of the matrix (d x d)
# BOTTLENECK: O(d^2) memory and time
S_W = torch.zeros((d, d))
diff = x - class_mean
S_W += torch.matmul(diff.T, diff) # Heavy MatMul
```

Listing 1: Naive Approach (Slow)

3 Approach 2: Optimized Implementation (Scalar-based)

The optimization relies on two fundamental mathematical pillars.

3.1 Pillar 1: The "Trace Trick"

The trace of an outer product (matrix) is equal to the inner product (scalar).

> **Cyclic Property:** $\text{Tr}(vv^T) = v^T v = \|v\|^2$

This means that instead of calculating a 4096×4096 matrix and summing the diagonal, we can simply calculate the squared length of the vector. We move from 16 million operations to 4096 operations.

3.2 Pillar 2: Variance Decomposition

Instead of calculating S_W (complex), we calculate the **Total Scatter Matrix** (S_T), which represents the global variance of the data ignoring classes, and subtract S_B.

$$S_W = S_T - S_B \quad \Rightarrow \quad \text{Tr}(S_W) = \text{Tr}(S_T) - \text{Tr}(S_B) \tag{4}$$

The final optimized formulas become simple sums of Euclidean distances:

$$\text{Tr}(S_T) = \sum_{i=1}^{N} \|x_i - \mu\|^2 \tag{5}$$

$$\text{Tr}(S_B) = \sum_{c \in C} N_c \, \|\mu_c - \mu\|^2 \tag{6}$$

3.3 Python Code (Optimized)

```python
# Direct calculation of scalars
# ADVANTAGE: O(d) linear memory
global_mean = embeddings.mean(dim=0)

# Tr(St) is the sum of squared distances from the global mean
tr_St = torch.sum((embeddings - global_mean) ** 2)

# Tr(Sb) is the weighted sum of centroid distances
tr_Sb = 0
for c in classes:
    # ... calculate mu_c ...
    tr_Sb += n_c * torch.sum((mu_c - global_mean)**2)

tr_Sw = tr_St - tr_Sb # Scalar subtraction
J = tr_Sb / tr_Sw
```

Listing 2: Optimized Approach (Fast)

4 Formal Proof of Equivalence

In this section, we demonstrate why it is legitimate to calculate S_W as the difference $S_T - S_B$. We must prove that:

$$S_T = S_W + S_B$$

Step 1: Definition of S_T The total scatter matrix is the sum of the squared deviations of each point from the global mean μ:

$$S_T = \sum_{c \in C} \sum_{i \in c} (x_i - \mu)(x_i - \mu)^T$$

Step 2: Algebraic Manipulation We add and subtract the class mean μ_c inside the parenthesis. This does not change the value but allows us to separate the intra-class and inter-class components:

$$(x_i - \mu) = \underbrace{(x_i - \mu_c)}_{\text{Internal deviation}} + \underbrace{(\mu_c - \mu)}_{\text{Between-class deviation}}$$

Step 3: Expansion of the Square Substitute and expand the product $(A+B)(A+B)^T = AA^T + BB^T + AB^T + BA^T$:

$$\begin{aligned}(x_i - \mu)(x_i - \mu)^T = &\underbrace{(x_i - \mu_c)(x_i - \mu_c)^T}_{\text{Term } S_W} \\ &+ \underbrace{(\mu_c - \mu)(\mu_c - \mu)^T}_{\text{Term } S_B} \\ &+ \underbrace{(x_i - \mu_c)(\mu_c - \mu)^T + (\mu_c - \mu)(x_i - \mu_c)^T}_{\text{Cross-terms}}\end{aligned}$$

Step 4: Summation Analysis Now we apply the summation $\sum_{i \in c}$ to all terms. The critical part is demonstrating that the sum of the **Cross-terms is zero**. Consider the first mixed term and factor out what does not depend on i (i.e., the means):

$$\sum_{i \in c} (x_i - \mu_c)(\mu_c - \mu)^T = \left[\sum_{i \in c} (x_i - \mu_c) \right] (\mu_c - \mu)^T$$

Analyze the term in square brackets (sum of deviations from the mean):

$$\sum_{i \in c} (x_i - \mu_c) = \sum_{i \in c} x_i - \sum_{i \in c} \mu_c$$

We know that $\sum_{i \in c} x_i = N_c \mu_c$ (by definition of mean) and that summing a constant N_c times yields $N_c \mu_c$. Therefore:

$$= (N_c \mu_c) - (N_c \mu_c) = \mathbf{0}$$

Since the sum of deviations is zero, **all cross-terms vanish**.

Conclusion Only the quadratic terms remain:

$$S_T = \underbrace{\sum_c \sum_{i \in c} (x_i - \mu_c)(x_i - \mu_c)^T}_{S_W} + \underbrace{\sum_c \sum_{i \in c} (\mu_c - \mu)(\mu_c - \mu)^T}_{S_B}$$

Note: in the second term, the argument is constant for all i, so the inner sum becomes a multiplication by N_c.

$$S_T = S_W + \sum_c N_c(\mu_c - \mu)(\mu_c - \mu)^T$$

$$S_T = S_W + S_B$$

Applying the Trace operator (which is linear), we obtain the scalar formula used in the optimized code:

$$\mathrm{Tr}(S_T) = \mathrm{Tr}(S_W) + \mathrm{Tr}(S_B) \quad \square$$