Gaussian Discriminant Analysis (GDA) - Complete Breakdown

---

1. In-Depth and Specific Intuitive Understanding

What is Gaussian Discriminant Analysis (GDA)?

Gaussian Discriminant Analysis (GDA) is a **probabilistic classification algorithm** based on **Bayes' theorem** and assumes that each class follows a **Gaussian (Normal) distribution** in feature space.

Instead of directly modeling $P(y|x)$ like Logistic Regression, GDA models the **joint probability** $P(x, y)$ and then applies Bayes' rule to obtain $P(y|x)$.

Key Idea

- **GDA assumes that each class has a Gaussian distribution:**

$$p(x|y = k) \sim \mathcal{N}(\mu_k, \Sigma)$$

- **Using Bayes' Theorem**, the probability of class $y = k$ given x is:

$$P(y = k|x) = \frac{P(x|y = k)P(y = k)}{P(x)}$$

- **We classify x into the class with the highest posterior probability.**

Intuition Behind GDA

- Imagine you have two groups of points (e.g., two species of flowers) and each group follows a **Gaussian (bell-shaped) distribution**.
- GDA finds the **probability density function** (PDF) of each group.
- When a new point arrives, GDA **compares probabilities** and assigns the class **with the highest likelihood**.

---

2. When GDA is Used and When It Should Be Avoided

✅ When to Use GDA

- When **features follow a roughly Gaussian distribution**.
- When **data is well-separated** (GDA models decision boundaries well).
- When the number of **samples is small** (parametric models perform better in small-data regimes).
- When **computational efficiency is needed** (GDA has a closed-form solution).

❌ When to Avoid GDA

- If the **Gaussian assumption does not hold**, GDA performs poorly.
- If the dataset is **high-dimensional**, estimating the covariance matrix Σ is computationally expensive.
- If the **classes have highly non-linear boundaries**, GDA's linear or quadratic decision boundary may not work well.

---

3. When It Fails to Converge and How to Avoid That

When GDA Fails

- **Singular Covariance Matrix Σ (Non-Invertibility)**:
  - Happens when there are **fewer data points than features**.
  - Happens when **some features are perfectly correlated** (multicollinearity).
- **Poor Performance if Data is Not Gaussian**:
  - If the data distribution is skewed or multimodal, GDA struggles.
- **Unstable Estimation in Small Datasets**:
  - Estimating the covariance matrix with very few samples leads to poor generalization.

How to Ensure Convergence

✅ **Use Regularized Covariance Estimation (Shrinkage Methods)** for stability.
✅ **Reduce feature dimensionality** (e.g., PCA) if $n \ll m$.
✅ **Check for Gaussianity** (use normality tests like Shapiro-Wilk).

When GDA Always Converges

- If there are **enough samples** relative to feature dimensions.
- If the covariance matrix is **well-conditioned** (not singular).
- If features are **reasonably Gaussian**.

4. Advantages and Disadvantages

Advantages

✅ **Simple and computationally efficient**.

✅ **Works well for small datasets** (unlike deep learning models).

✅ **Interpretable**—explicit probability distributions.

✅ **Handles missing values well** (if modeled probabilistically).

Disadvantages

❌ **Assumes Gaussian distributions**, which may not hold in real-world data.

❌ **Covariance estimation is expensive** for high-dimensional data.

❌ **Linear Decision Boundaries (if using shared Σ)** may not capture complex patterns.

5. Intuitive Algorithm / Pseudo Code

1. **Estimate class priors** $P(y)$.
2. **Estimate mean** μ_k **and covariance** Σ for each class.
3. **Compute the likelihood** $P(x|y)$ **using the Gaussian density function.**
4. **Apply Bayes' theorem to compute** $P(y|x)$.
5. **Assign** x **to the class with the highest posterior probability.**

plaintext Copy Edit

```
1. Compute class priors: P(y=k) = (# samples in class k) / (total samples)
2. Compute class means: μ_k = mean(X | y=k)
3. Compute shared covariance matrix Σ:
   Σ = (1/m) * Σ (x_i - μ_yi) (x_i - μ_yi)^T
4. Compute class-conditional probabilities using Gaussian PDF
5. Use Bayes' theorem to classify new data points
```

6. Mathematical and Logical Breakdown

Step 1: Estimate Class Priors

The probability of each class:

$$P(y = k) = \frac{\text{Number of points in class } k}{\text{Total number of points}}$$

Step 2: Estimate Mean and Covariance

For each class $y = k$, estimate the **mean vector**:

$$\mu_k = \frac{1}{N_k} \sum_{i:y_i=k} x_i$$

And the **shared covariance matrix**:

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_{y_i})(x_i - \mu_{y_i})^T$$

Step 3: Compute Class-Conditional Probability (Gaussian Density Function)

For a new data point x, the likelihood under class k is:

$$P(x|y = k) = \frac{1}{(2\pi)^{n/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(x - \mu_k)^T \Sigma^{-1} (x - \mu_k)\right)$$

Step 4: Compute Posterior Probability (Bayes' Rule)

Using Bayes' Theorem:

$$P(y = k|x) = \frac{P(x|y = k)P(y = k)}{P(x)}$$

7. Manual Implementation in Python

```python
import numpy as np

class GaussianDiscriminantAnalysis:
    def __init__(self):
        self.classes = None
        self.means = {}
        self.covariance = None
        self.priors = {}

    def fit(self, X, y):
        """Train GDA by estimating class priors, means, and covariance matrix."""
        self.classes = np.unique(y)
        m, n = X.shape
        self.covariance = np.zeros((n, n))

        for c in self.classes:
            X_c = X[y == c]
            self.means[c] = np.mean(X_c, axis=0)
            self.priors[c] = X_c.shape[0] / m
            self.covariance += np.cov(X_c, rowvar=False) * (X_c.shape[0] - 1)

        self.covariance /= m

    def predict(self, X):
        """Predict class labels using Bayes' theorem."""
        predictions = []
        inv_cov = np.linalg.inv(self.covariance)

        for x in X:
            posteriors = {}
            for c in self.classes:
                mean_diff = x - self.means[c]
                likelihood = -0.5 * mean_diff.T @ inv_cov @ mean_diff
                posteriors[c] = np.log(self.priors[c]) + likelihood

            predictions.append(max(posteriors, key=posteriors.get))

        return np.array(predictions)
```

8. Scikit-Learn Implementation (Fully Commented)

python

```python
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import train_test_split
from sklearn.datasets import make_classification

# Generate synthetic dataset
X, y = make_classification(n_samples=100, n_features=2, n_classes=2, random_state=4

# Split into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_sta

# Initialize GDA model
model = LinearDiscriminantAnalysis()
model.fit(X_train, y_train)

# Predict class labels
y_pred = model.predict(X_test)

# Print accuracy
print("Model Accuracy:", model.score(X_test, y_test))
```

Final Summary

- GDA is a probabilistic classifier that assumes Gaussian distributions per class.
- It estimates class priors, means, and a covariance matrix to classify points.
- Works well when data is Gaussian, but fails in non-Gaussian cases.
- Computationally efficient for small datasets.