Cardiovascular Risk Prediction

University: Shiraz University
Course: Artificial intelligence – Spring 2025
Instructor: Prof. Zohreh Azimifar
Student: Parsa Haghighatgoo – Sina Hakimzadeh
Student ID: 40030644 – 40131857
Assignment Title: Programming Task
Due Date: God and TAs know

1 Task 1: Data Exploration & Gaussian Fitting

Problem Description

This task involves exploring the UCI Heart Disease dataset and preparing it for modeling. Specifically, we are required to:

- Handle missing or invalid values through deletion or imputation.
- Visualize the distribution of features per class.
- Select two informative features and fit 2D Gaussian distributions.
- Plot contour maps and evaluate the Gaussian assumption.

Approach: Data Cleaning

The dataset initially contained missing values and non-binary target labels. We performed the following preprocessing steps:

1. Converted the `num` column into a binary `target` variable, where 1 indicates presence of heart disease and 0 indicates absence.
2. Dropped rows containing any missing values.
3. Verified the resulting shape and integrity of the cleaned dataset.

Code: Data Cleaning

```python
# Step 2: Drop irrelevant columns
# df.drop(columns=["id", "dataset"], inplace=True)

# Step 3: Binarize target column
df["target"] = df["num"].apply(lambda x: 1 if x > 0 else 0)
# df.drop(columns=["num"], inplace=True)

# Step 4: Drop rows with any missing values
df.dropna(inplace=True)

# Step 5: Check remaining shape and confirm no missing values
print("Data shape after dropping missing values:", df.shape)
print("Remaining missing values:\n", df.isnull().sum())
```
Listing 1: Data Cleaning and Target Binarization

Results

After cleaning, the dataset consists of 299 samples and 17 features. All missing values have been removed. The transformed `target` variable is now binary, suitable for classification tasks.

- **Data shape:** (299, 17)
- **Missing values:** None

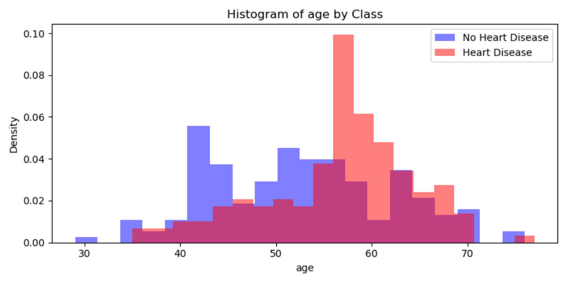
Feature Distribution per Class

To visualize how features differ between patients with and without heart disease, we plotted histograms of all numeric features grouped by the `target` class. These plots help identify which features are likely to be informative.

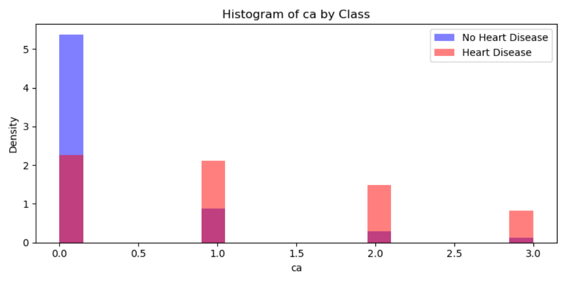
```python
# Get list of numeric columns (excluding target)
numeric_cols = df.select_dtypes(include=["float64", "int64"]).columns.drop
    ("target")

# Plot histogram for each numeric column, separated by class
for col in numeric_cols:
    plt.figure(figsize=(8, 4))
    plt.hist(df[df["target"] == 0][col], bins=20, alpha=0.5,
             label="No Heart Disease", color="blue", density=True)
    plt.hist(df[df["target"] == 1][col], bins=20, alpha=0.5,
             label="Heart Disease", color="red", density=True)
    plt.title(f"Histogram of {col} by Class")
    plt.xlabel(col)
    plt.ylabel("Density")
    plt.legend()
    plt.tight_layout()
    plt.show()
```
Listing 2: Plotting Histograms by Class

Histograms

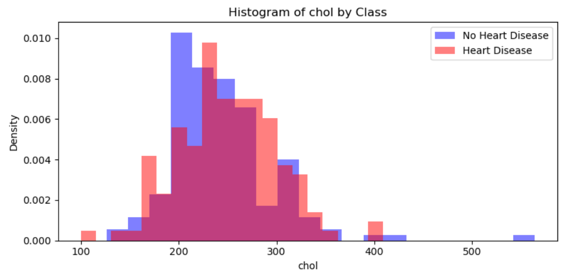
Below are the histograms of key numeric features:



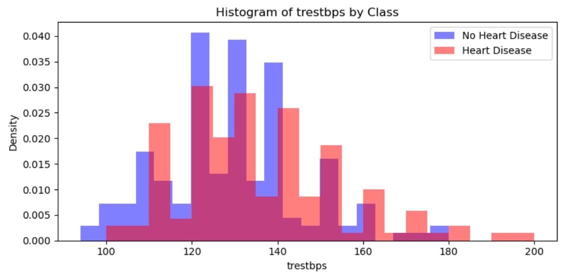
(a) Age



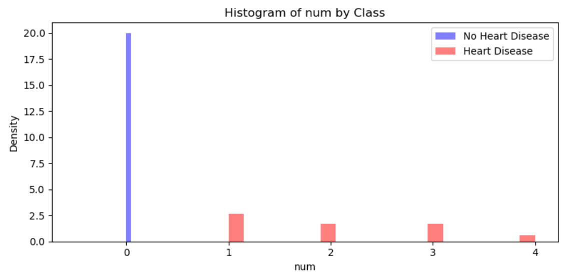
(b) ca



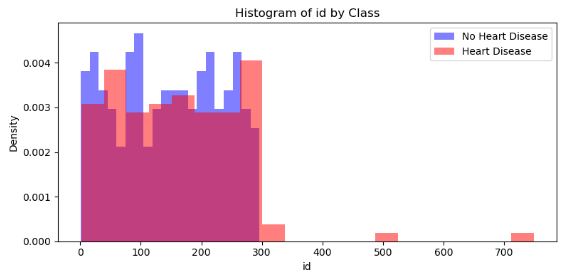
(c) chol



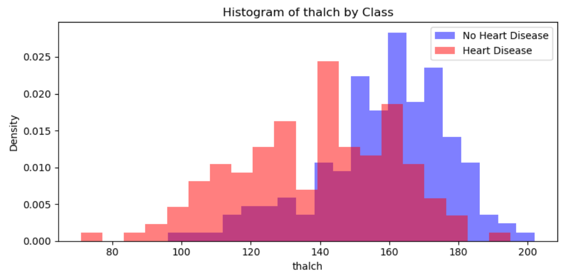
(d) Resting Blood Pressure (trestbps)



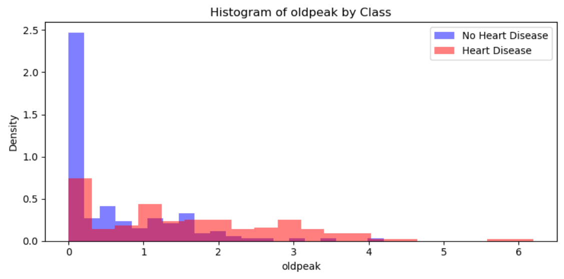
(e) num



(f) id



(g) Max Heart Rate (thalach)



(h) ST Depression (oldpeak)

Figure 1: Histograms of selected features grouped by class (target = 0 or 1).

Feature Selection

From the histograms, we observed that `chol` (cholesterol) and `thalch` (maximum heart rate achieved) showed noticeable variation between the two classes. These two features were selected as inputs for further modeling using 2D Gaussian distributions.

```
# Select the two features: 'chol' and 'thalch'
selected_features = ["chol", "thalch", "target"]

```

```python
# Extract subset
df_selected = df[selected_features]

# Display first few rows
print(df_selected.head())
```

Listing 3: Selecting informative features

Selected Feature Sample

chol	thalch	target
233.0	150.0	0
286.0	108.0	1
229.0	129.0	1
250.0	187.0	0
204.0	172.0	0

2D Gaussian Modeling and Contour Visualization

We modeled each class as a bivariate Gaussian distribution over the selected features: `chol` and `thalch`. This involved estimating the mean and covariance matrix for each class and plotting their probability density contours.

```python
# Step 1: Extract data by class
X0 = df_selected[df_selected["target"] == 0][["chol", "thalch"]].values
X1 = df_selected[df_selected["target"] == 1][["chol", "thalch"]].values

# Step 2: Compute means and covariances
mu0 = np.mean(X0, axis=0)
mu1 = np.mean(X1, axis=0)
cov0 = np.cov(X0, rowvar=False)
cov1 = np.cov(X1, rowvar=False)

# Step 3: Create mesh grid
x = np.linspace(df_selected["chol"].min(), df_selected["chol"].max(), 100)
y = np.linspace(df_selected["thalch"].min(), df_selected["thalch"].max(),
    100)
X, Y = np.meshgrid(x, y)
grid = np.stack([X.ravel(), Y.ravel()], axis=1)

# Step 4: Multivariate Gaussian PDF
def gaussian_pdf(x, mean, cov):
    d = mean.shape[0]
    det = np.linalg.det(cov)
    inv = np.linalg.inv(cov)
    norm_const = 1.0 / (np.power((2*np.pi), d/2) * np.sqrt(det))
    diff = x - mean
    result = norm_const * np.exp(-0.5 * np.sum(diff @ inv * diff, axis=1))
    return result

# Step 5: Evaluate PDF on grid
Z0 = gaussian_pdf(grid, mu0, cov0).reshape(X.shape)
```

```
Z1 = gaussian_pdf(grid, mu1, cov1).reshape(X.shape)

# Step 6: Plot
plt.figure(figsize=(10, 6))
plt.contour(X, Y, Z0, levels=5, colors='blue', linestyles='dashed', label=
    "No Heart Disease")
plt.contour(X, Y, Z1, levels=5, colors='red', linestyles='solid', label="
    Heart Disease")
plt.scatter(X0[:, 0], X0[:, 1], color='blue', alpha=0.3, label="Class 0")
plt.scatter(X1[:, 0], X1[:, 1], color='red', alpha=0.3, label="Class 1")
plt.xlabel("chol")
plt.ylabel("thalch")
plt.title("2D Gaussian Contour Plots for Each Class")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 4: Fitting and visualizing 2D Gaussian distributions

Contour Plot

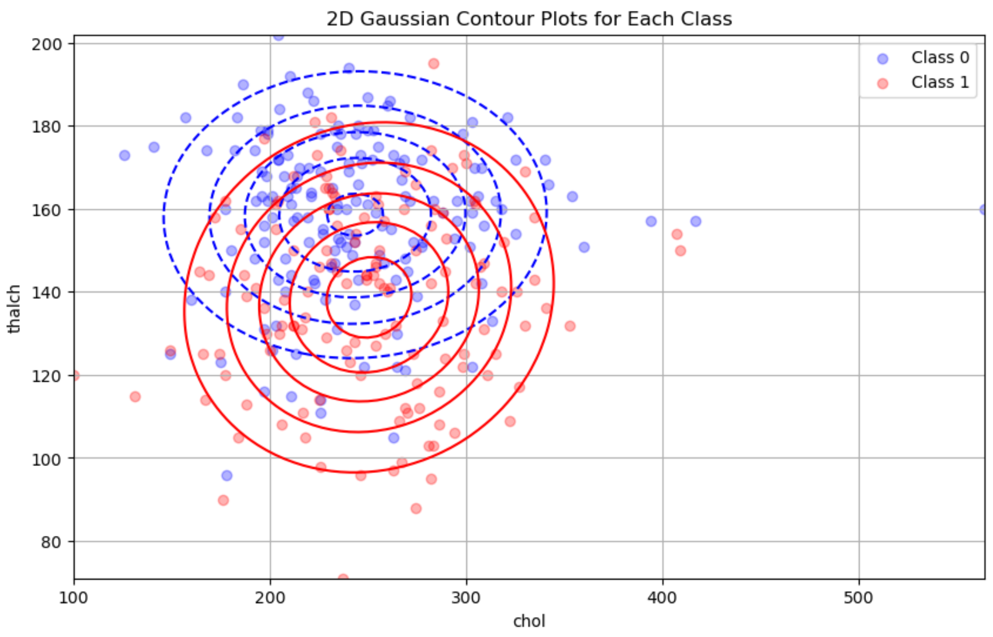


Figure 2: 2D Gaussian contour plots for `chol` and `thalch` per class. Dashed blue lines represent class 0 (no heart disease), and solid red lines represent class 1 (heart disease).

Analysis

From the contour maps, we observe that both classes are approximately elliptical in distribution, though the orientation and spread differ. This supports the use of a Gaussian assumption for these features. The contours visually suggest potential overlap between the classes, particularly in the central region, which has implications for model separability.

Normality Evaluation via Plots

To validate the Gaussian assumption for the selected features, we visualized:

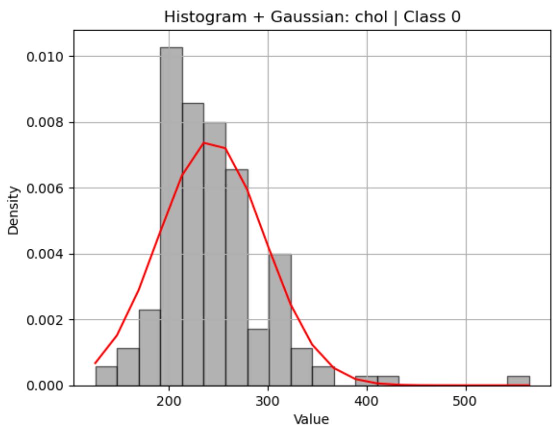
- Histograms overlaid with fitted Gaussian curves
- Q–Q plots comparing sample quantiles with theoretical quantiles

These plots were generated separately for each class (0 = No Heart Disease, 1 = Heart Disease).

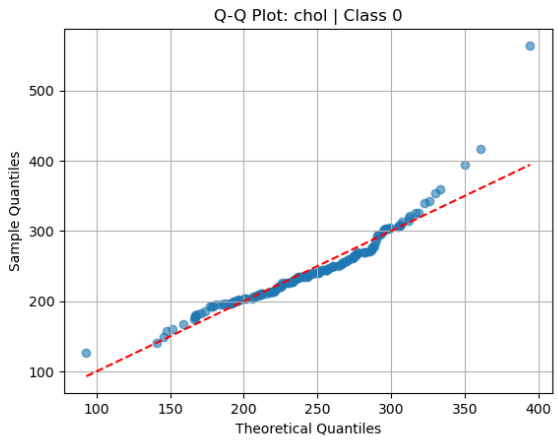
```python
# Helper: Histogram with Gaussian curve
def plot_hist_with_gaussian(data, title):
    mu = np.mean(data)
    sigma = np.std(data)
    count, bins, _ = plt.hist(data, bins=20, density=True, alpha=0.6,
    color='gray', edgecolor='black')
    gauss_curve = 1/(sigma * np.sqrt(2*np.pi)) * np.exp(-(bins - mu)**2 /
    (2*sigma**2))
    plt.plot(bins, gauss_curve, color='red')
    plt.title(title)
    plt.grid(True)
    plt.show()

# Helper: Manual Q-Q plot
def qq_plot(data, title):
    sorted_data = np.sort(data)
    n = len(data)
    theoretical_q = np.sort(np.random.normal(np.mean(data), np.std(data),
    n))
    plt.scatter(theoretical_q, sorted_data, alpha=0.6)
    plt.plot(theoretical_q, theoretical_q, color='red', linestyle='--')
    plt.title("Q-Q Plot: " + title)
    plt.grid(True)
    plt.show()
```

Listing 5: Plotting histogram with Gaussian and Q-Q plots

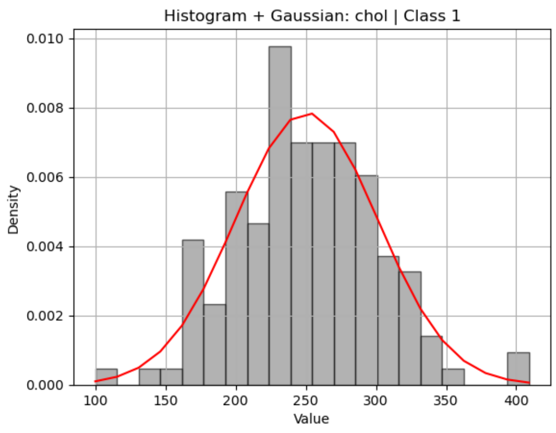
Visual Normality Checks



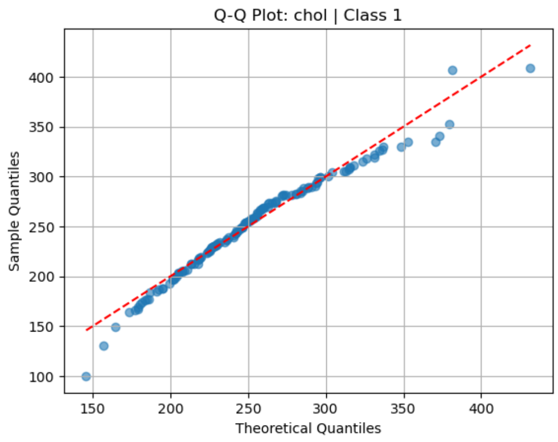
(a) Histogram: chol — Class 0



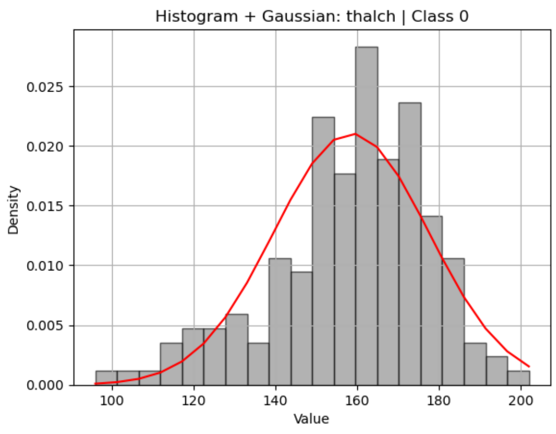
(b) Q–Q Plot: chol — Class 0



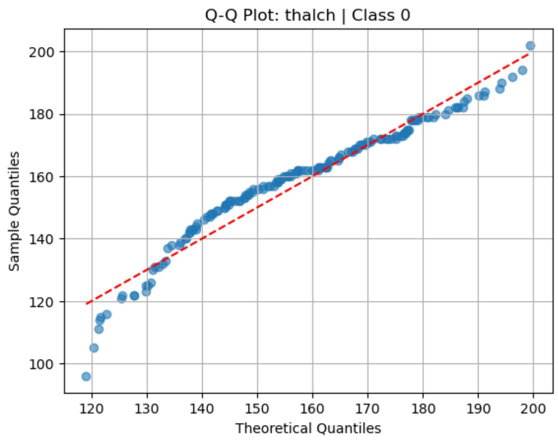
(c) Histogram: chol — Class 1



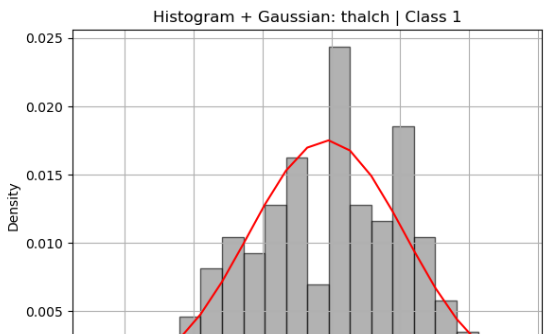
(d) Q–Q Plot: chol — Class 1

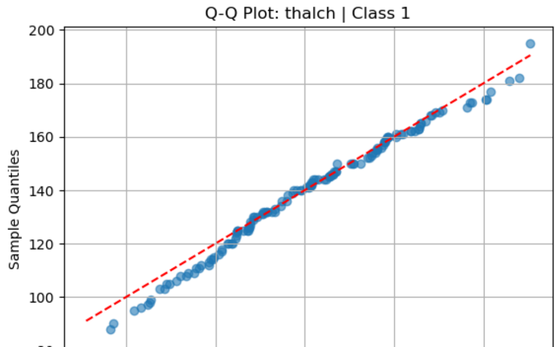


(e) Histogram: thalch — Class 0



(f) Q–Q Plot: thalch — Class 0





Interpretation

The histogram plots show reasonable overlap between the empirical distributions and Gaussian fits, particularly for `thalch`. The Q–Q plots indicate approximate linearity, supporting the use of Gaussian models for these features. However, slight deviations suggest that while the Gaussian assumption is acceptable, it may not be perfect.

2 Task 2: Generative Modeling (Gaussian Classifier)

Problem Description

In this task, we fit a Gaussian classifier by modeling each class with its own mean and covariance matrix. We aim to compare the resulting decision boundaries under two assumptions:

- Linear Discriminant Analysis (LDA): Shared covariance between classes
- Quadratic Discriminant Analysis (QDA): Separate covariance for each class

Approach: Estimating Class-Conditional Parameters

We use the selected features (`chol`, `thalch`) and compute:

- Mean vector μ_c for each class $c \in \{0, 1\}$
- Covariance matrix Σ_c for each class

```python
# Extract feature matrix and labels
X = df_selected[["chol", "thalch"]].values
y = df_selected["target"].values

# Split by class
X0 = X[y == 0]
X1 = X[y == 1]

# Mean vectors
mu0 = np.mean(X0, axis=0)
mu1 = np.mean(X1, axis=0)

# Covariance matrices
cov0 = np.cov(X0, rowvar=False)
cov1 = np.cov(X1, rowvar=False)
```

Listing 6: Computing mean and covariance for each class

Estimated Parameters

Class 0 (No Heart Disease)

- Mean vector:

$$\mu_0 = \begin{bmatrix} 243.49 & 158.58 \end{bmatrix}$$

- Covariance matrix:

$$\Sigma_0 = \begin{bmatrix} 2889.87 & 19.54 \\ 19.54 & 362.65 \end{bmatrix}$$

Class 1 (Heart Disease)

- Mean vector:

$$\mu_1 = \begin{bmatrix} 250.58 & 138.68 \end{bmatrix}$$

- Covariance matrix:

$$\Sigma_1 = \begin{bmatrix} 2602.62 & 92.54 \\ 92.54 & 523.00 \end{bmatrix}$$

Interpretation

The two classes exhibit different statistical properties, particularly in their means and off-diagonal covariance terms, which suggests that the QDA assumption of separate covariance matrices may be more suitable. However, we'll validate this by visualizing their decision boundaries.

Decision Boundary via Log-Likelihood Ratio

We computed the log-likelihood ratio between the two Gaussian class-conditional distributions. The decision boundary corresponds to the set of points where the log-likelihood ratio is zero:

$$\log p(\mathbf{x} \mid y = 1) - \log p(\mathbf{x} \mid y = 0) = 0$$

This boundary effectively separates the feature space into regions classified as either class 0 or class 1 under the maximum likelihood principle.

```python
def multivariate_gaussian(x, mean, cov):
    size = mean.shape[0]
    det = np.linalg.det(cov)
    inv = np.linalg.inv(cov)
    norm_const = 1.0 / (np.power((2 * np.pi), size / 2) * np.sqrt(det))
    diff = x - mean
    exponent = -0.5 * np.sum(diff @ inv * diff, axis=1)
    return norm_const * np.exp(exponent)

# Mesh grid
x_vals = np.linspace(X[:, 0].min(), X[:, 0].max(), 200)
y_vals = np.linspace(X[:, 1].min(), X[:, 1].max(), 200)
X_grid, Y_grid = np.meshgrid(x_vals, y_vals)
grid_points = np.c_[X_grid.ravel(), Y_grid.ravel()]

# Evaluate densities
p0 = multivariate_gaussian(grid_points, mu0, cov0)
p1 = multivariate_gaussian(grid_points, mu1, cov1)

```

```
# Log-likelihood ratio
log_ratio = np.log(p1 + 1e-12) - np.log(p0 + 1e-12)
log_ratio = log_ratio.reshape(X_grid.shape)

# Plot
plt.figure(figsize=(10, 6))
plt.contour(X_grid, Y_grid, log_ratio, levels=[0], colors='black',
    linewidths=2)
plt.scatter(X0[:, 0], X0[:, 1], color='blue', alpha=0.4, label="Class 0")
plt.scatter(X1[:, 0], X1[:, 1], color='red', alpha=0.4, label="Class 1")
plt.title("Decision Boundary (Log-Likelihood Ratio = 0)")
plt.xlabel("chol")
plt.ylabel("thalch")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 7: Plotting generative decision boundary using log-likelihood ratio

Decision Boundary Plot

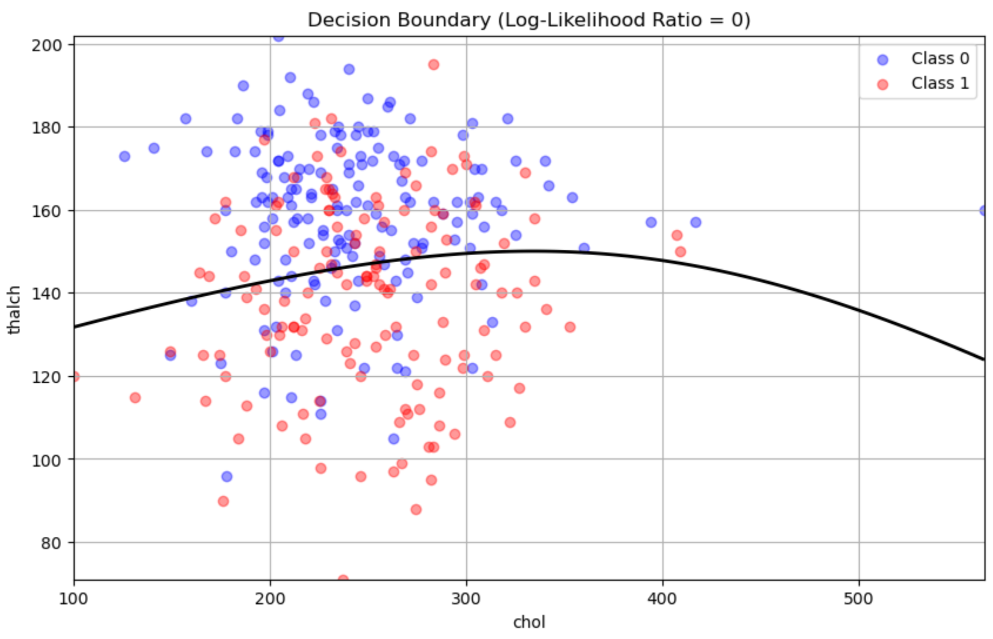


Figure 4: Decision boundary between classes based on log-likelihood ratio. Dashed contour at level 0 indicates the optimal classification threshold.

Analysis

The resulting boundary is nonlinear due to the class-specific covariance structures. This behavior aligns with Quadratic Discriminant Analysis (QDA), which allows for elliptical boundaries that better capture class separation when covariances differ significantly.

In the next step, we explicitly compare the QDA boundary with a Linear Discriminant Analysis (LDA) boundary using a shared covariance matrix.

LDA vs. QDA Comparison

We now compare Linear Discriminant Analysis (LDA), which assumes a shared covariance matrix across classes, with Quadratic Discriminant Analysis (QDA), which allows each class to have its own covariance. This impacts the shape of the decision boundary.

```
# Step 1: Shared covariance for LDA
shared_cov = (cov0 + cov1) / 2

# Step 2: Evaluate Gaussian with shared covariance for LDA
p0_lda = multivariate_gaussian(grid_points, mu0, shared_cov)
p1_lda = multivariate_gaussian(grid_points, mu1, shared_cov)
log_ratio_lda = np.log(p1_lda + 1e-12) - np.log(p0_lda + 1e-12)
log_ratio_lda = log_ratio_lda.reshape(X_grid.shape)

# Step 3: QDA boundary (already computed as log_ratio)

# Step 4: Plot both boundaries
plt.figure(figsize=(10, 6))
plt.contour(X_grid, Y_grid, log_ratio, levels=[0], colors='red',
    linewidths=2, linestyles='--', label='QDA')
plt.contour(X_grid, Y_grid, log_ratio_lda, levels=[0], colors='blue',
    linewidths=2, linestyles='-', label='LDA')
plt.scatter(X0[:, 0], X0[:, 1], color='gray', alpha=0.3, label="Class 0")
plt.scatter(X1[:, 0], X1[:, 1], color='black', alpha=0.3, label="Class 1")
plt.title("Decision Boundaries: LDA vs QDA")
plt.xlabel("chol")
plt.ylabel("thalch")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 8: Plotting LDA vs. QDA decision boundaries

LDA vs. QDA Boundary Plot

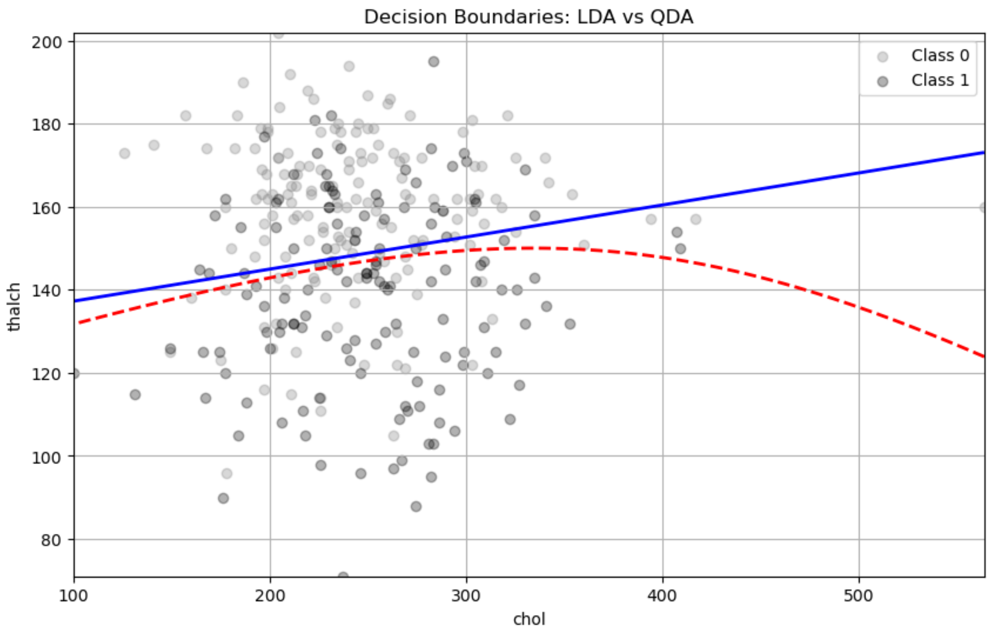


Figure 5: Comparison of decision boundaries: LDA (blue solid line) assumes shared covariance and produces a linear boundary. QDA (red dashed curve) models distinct covariances and produces a curved boundary.

Analysis

As shown above, the QDA decision boundary is nonlinear due to the flexibility in modeling class-specific covariance. This allows QDA to capture more complex class distributions, especially when feature correlation differs significantly between classes. In contrast, the LDA boundary is linear and simpler, which may be more robust but less expressive.

When covariances are similar, LDA performs comparably to QDA with less variance. When covariances differ, QDA can provide better class separation but may risk overfitting on small datasets.

Discussion: Impact of Covariance Assumption (LDA vs. QDA)

What is the assumption?

- **LDA** assumes all classes share the same covariance matrix.
- **QDA** allows each class to have its own covariance matrix.

Impact on Modeling:

Aspect	LDA (Shared Covariance)	QDA (Distinct Covariance)
Decision Boundary	Linear (straight line)	Quadratic (curved)
Model Complexity	Lower	Higher
Overfitting Risk	Lower (better for small data)	Higher (needs more data)
Speed	Faster (fewer parameters)	Slower (more matrix inversions)
Assumption	Shared variance structure	Class-specific variance

When LDA Works Best:

- Class distributions are similar in shape and scale.
- Limited training data is available (less prone to overfitting).

When QDA Works Best:

- Class distributions differ significantly (e.g., one is more spread out).
- Sufficient data is available to estimate covariance matrices accurately.

Analysis of Our Plot: LDA vs. QDA

Legend:

- **Blue line** = LDA decision boundary
- **Red dashed curve** = QDA decision boundary
- Gray/Black dots = Data points from class 0 and class 1

Observations:

- **Shape:** The LDA boundary is linear as expected, while QDA is curved and adapts more flexibly to the distribution of points.
- **Class Spread:** Class 1 appears more spread in the `thalch` direction, while Class 0 is more compact.
- **Boundary Alignment:** QDA curves in the high-cholesterol region, likely capturing class-specific variance. LDA oversimplifies this region.

Conclusion:

- QDA is likely a better fit for this dataset due to distinct class variances.
- If data were limited, LDA would still be a more robust and efficient option.

3 Task 3: Gaussian Naive Bayes

Problem Description

In this task, we implement the Gaussian Naive Bayes (GNB) classifier from scratch. The Naive Bayes model assumes:

- Class-conditional independence between features
- Gaussian distribution of each feature within each class

Approach

We estimate the mean and variance of each feature for each class independently. Then we compute the Gaussian log-likelihood for each test point and assign the class with the maximum posterior probability.

```python
class GaussianNaiveBayes:
    def fit(self, X, y):
        self.classes = np.unique(y)
        self.mean = {}
        self.var = {}
        self.priors = {}

        for c in self.classes:
            X_c = X[y == c]
            self.mean[c] = np.mean(X_c, axis=0)
            self.var[c] = np.var(X_c, axis=0) + 1e-6  # avoid divide-by-zero
            self.priors[c] = X_c.shape[0] / X.shape[0]

    def _gaussian_logpdf(self, x, mean, var):
        return -0.5 * np.sum(np.log(2 * np.pi * var) + ((x - mean) ** 2) / var, axis=1)

    def predict(self, X):
        log_probs = []
        for c in self.classes:
            prior = np.log(self.priors[c])
            class_cond = self._gaussian_logpdf(X, self.mean[c], self.var[c])
            log_probs.append(prior + class_cond)
        return np.argmax(log_probs, axis=0)
```

Listing 9: Gaussian Naive Bayes implementation

Model Training and Accuracy

We trained the GNB model using the same two features: `chol` and `thalch`. Accuracy was computed on the training set for baseline evaluation.

```python
# Prepare data
X_gnb = df_selected[["chol", "thalch"]].values
y_gnb = df_selected["target"].values

# Train model
gnb = GaussianNaiveBayes()
gnb.fit(X_gnb, y_gnb)

# Predict on training data
```

```python
y_pred = gnb.predict(X_gnb)

# Accuracy
accuracy = np.mean(y_pred == y_gnb)
print(f"Training accuracy: {accuracy:.3f}")
```

Results

- **Training Accuracy:** 0.709

Interpretation

The Gaussian Naive Bayes model achieves a training accuracy of approximately 71%. Despite its simplifying assumption of feature independence, GNB can perform surprisingly well — especially on low-dimensional data. However, performance may suffer when features are correlated, as in our case with `chol` and `thalch`, which violates the independence assumption.

Feature-wise Statistics for Gaussian Naive Bayes

Gaussian Naive Bayes assumes that each feature is conditionally independent given the class. Therefore, we estimate the mean and variance of each feature separately for each class.

```python
# Get features and labels
X = df_selected[["chol", "thalch"]].values
y = df_selected["target"].values
classes = np.unique(y)

# Initialize containers
means = {}
variances = {}

# Loop over classes
for c in classes:
    X_c = X[y == c]
    means[c] = np.mean(X_c, axis=0)
    variances[c] = np.var(X_c, axis=0)

    print(f"\nClass {c}:")
    print("Mean:", means[c])
    print("Variance:", variances[c])
```

Listing 10: Per-feature mean and variance estimation

Results

Class 0 (No Heart Disease)

- Mean: [243.49, 158.58]
- Variance: [2871.81, 360.38]

Class 1 (Heart Disease)

- Mean: $[250.58,\ 138.68]$
- Variance: $[2583.90,\ 519.24]$

Interpretation

The class-wise variances confirm that each class exhibits distinct spread across the selected features. Class 0 shows greater spread in `thalch`, while Class 1 has more variability in both features. These estimates are directly used in the likelihood function of the Gaussian Naive Bayes classifier to compute posterior probabilities.

Posterior Probability Computation

The Gaussian Naive Bayes classifier estimates the posterior probability $P(y \mid \mathbf{x})$ using Bayes' theorem. We compute the log-likelihood for each feature and class, then combine it with the log prior:

$$\log P(y = c \mid \mathbf{x}) \propto \log P(y = c) + \sum_j \log P(x_j \mid y = c)$$

The likelihood terms are computed using the Gaussian probability density function for each feature.

```python
# Log-Gaussian PDF
def log_gaussian_pdf(x, mean, var):
    return -0.5 * (np.log(2 * np.pi * var) + ((x - mean) ** 2) / var)

# Compute log-posteriors
log_posteriors = np.zeros((X.shape[0], len(classes)))

for idx, c in enumerate(classes):
    mean = means[c]
    var = variances[c]
    prior = np.log(np.sum(y == c) / len(y))  # log prior
    log_likelihood = log_gaussian_pdf(X, mean, var).sum(axis=1)
    log_posteriors[:, idx] = prior + log_likelihood

# Convert to normalized posterior probabilities
posteriors = np.exp(log_posteriors)
posteriors /= np.sum(posteriors, axis=1, keepdims=True)
```

Listing 11: Posterior probability computation using GNB

Sample Output

Posterior probabilities for the first five samples:

$$\begin{bmatrix} 0.582 & 0.418 \\ 0.080 & 0.920 \\ 0.310 & 0.690 \\ 0.801 & 0.199 \\ 0.775 & 0.225 \end{bmatrix}$$

Interpretation

Each row in the output represents the model's confidence in predicting class 0 or 1 for a given sample. For instance, the second sample has a 92% probability of being in class 1 (heart disease), indicating high confidence. These probabilities can also be used to construct soft decision boundaries in the next visualization step.

Comparison with QDA Decision Boundary

To evaluate the impact of the independence assumption in Gaussian Naive Bayes (GNB), we compared its decision boundary with that of QDA (from Task 2). While both models assume Gaussian distributions, QDA uses full covariance matrices per class, whereas GNB assumes feature independence (i.e., diagonal covariance).

Model	**Covariance Assumption**	**Decision Boundary Shape**
QDA (Task 2)	Full covariance matrix per class	Curved / Flexible
Naive Bayes (Task 3)	Diagonal (independence)	Curved, but axis-aligned

```python
# Gaussian Naive Bayes log-posterior on grid
def gnb_log_posteriors_grid(grid):
    log_post = np.zeros((grid.shape[0], len(classes)))
    for idx, c in enumerate(classes):
        mean = means[c]
        var = variances[c]
        prior = np.log(np.sum(y == c) / len(y))
        log_likelihood = log_gaussian_pdf(grid, mean, var).sum(axis=1)
        log_post[:, idx] = prior + log_likelihood
    return log_post

# Evaluate on grid
log_post_grid = gnb_log_posteriors_grid(grid_points)
log_ratio_gnb = log_post_grid[:, 1] - log_post_grid[:, 0]
log_ratio_gnb = log_ratio_gnb.reshape(X_grid.shape)

# Plot
plt.figure(figsize=(10, 6))
plt.contour(X_grid, Y_grid, log_ratio, levels=[0], colors='red',
    linewidths=2, linestyles='--', label="QDA")
plt.contour(X_grid, Y_grid, log_ratio_gnb, levels=[0], colors='green',
    linewidths=2, linestyles='-', label="Naive Bayes")
plt.scatter(X0[:, 0], X0[:, 1], color='gray', alpha=0.3, label="Class 0")
plt.scatter(X1[:, 0], X1[:, 1], color='black', alpha=0.3, label="Class 1")
plt.title("Decision Boundaries: QDA vs Gaussian Naive Bayes")
plt.xlabel("chol")
plt.ylabel("thalch")
plt.legend()
plt.grid(True)
plt.show()
```

Listing 12: Decision boundary comparison: QDA vs Naive Bayes

Decision Boundary Plot

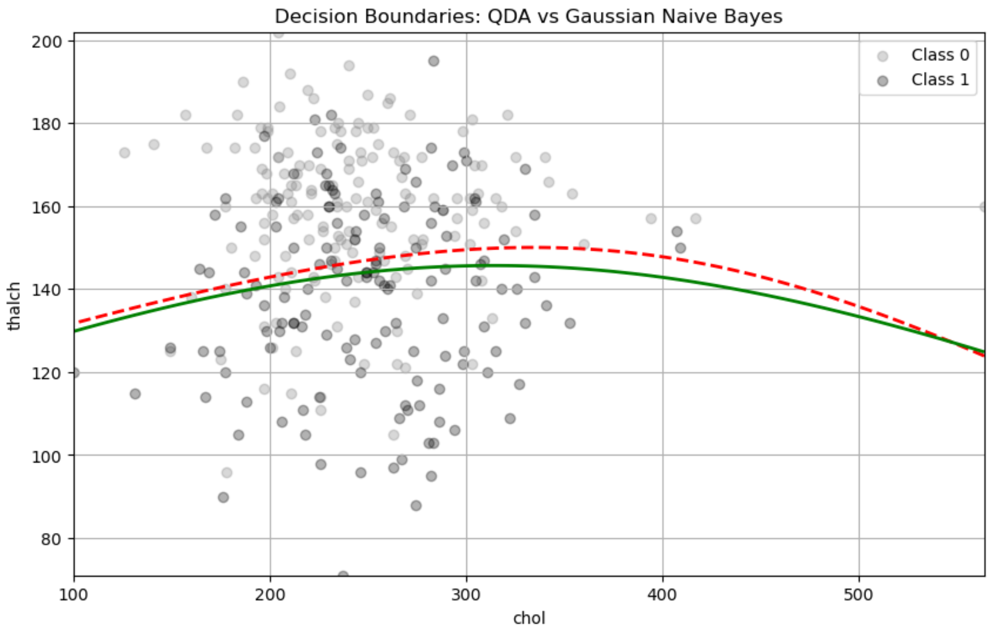


Figure 6: Decision boundaries for QDA (red dashed) and Gaussian Naive Bayes (green solid). Both boundaries are curved, but Naive Bayes produces axis-aligned boundaries due to its independence assumption.

Interpretation

While both QDA and GNB yield curved decision boundaries, the GNB boundary is more constrained due to its diagonal covariance assumption. This results in **axis-aligned curvature**, limiting its flexibility to model complex class separation.

QDA, on the other hand, adapts more fluidly to class-specific variance patterns, as seen in the contour's bending at higher cholesterol values. This demonstrates the trade-off between model complexity and representational power.

Impact of Feature Correlation in Gaussian Naive Bayes

Assumption

Gaussian Naive Bayes assumes conditional independence between features given the class:

$$P(x_1, x_2, ..., x_n \mid y) = \prod_{i=1}^{n} P(x_i \mid y)$$

This simplifies the model by treating each feature as independent and modeling them separately. As a result, GNB ignores correlations such as those between `chol` and `thalch`.

Problem with Correlated Features

When features are actually correlated, this assumption can break down and lead to:

- Inaccurate likelihood estimates
- Poor approximation of the true data distribution
- Oversimplified and sometimes incorrect decision boundaries

Comparison:

- **QDA** uses a full covariance matrix and captures inter-feature relationships.
- **GNB** uses diagonal covariance (only variances), ignoring correlations.

Effect on Decision Boundaries

- **QDA:** Produces smooth, curved boundaries that better align with data geometry.
- **Naive Bayes:** Produces axis-aligned decision boundaries (often rectangular in shape).

Consequences:

- GNB may misclassify samples in overlapping regions.
- It cannot adjust to interactions between features.

When Naive Bayes Still Works Well

Despite its limitations, GNB can perform adequately when:

- Correlations between features are weak or similar across classes.
- The data is high-dimensional or sparse (e.g., text classification).

Summary Table

Feature Correlation	Naive Bayes Impact
Weak or None	Performs well, fast and efficient
Strong Correlation	Inaccurate modeling, poor boundaries

4 Task 4: Discriminative Modeling (Logistic Regression)

Problem Description

In this task, we implement logistic regression using Newton's method for optimization. The model is trained using:

- The log-likelihood objective function
- The gradient of the log-likelihood
- The Hessian matrix (second-order derivative)

Model Formulation

Given input features $\mathbf{x}$, the logistic regression model defines:

$$h(\mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x}) = \frac{1}{1 + e^{-\mathbf{w}^\top \mathbf{x}}}$$

We optimize the log-likelihood:

$$\ell(\mathbf{w}) = \sum_{i=1}^{n} \left[y_i \cdot \mathbf{w}^\top \mathbf{x}_i - \log(1 + e^{\mathbf{w}^\top \mathbf{x}_i}) \right]$$

Newton's method uses the update:

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \mathbf{H}^{-1} \nabla \ell(\mathbf{w}_t)$$

Implementation

```python
# Prepare features and labels
X = df_selected[["chol", "thalch"]].values
y = df_selected["target"].values.reshape(-1, 1)

# Add bias term
X_aug = np.hstack((np.ones((X.shape[0], 1)), X))

# Sigmoid function
def sigmoid(z):
    return 1 / (1 + np.exp(-z))

# Log-likelihood
def log_likelihood(X, y, w):
    z = X @ w
    return np.sum(y * z - np.log(1 + np.exp(z)))

# Gradient
def gradient(X, y, w):
    z = X @ w
    h = sigmoid(z)
    return X.T @ (y - h)

# Hessian
def hessian(X, w):
    z = X @ w
    h = sigmoid(z)
    D = np.diag((h * (1 - h)).flatten())
    return -X.T @ D @ X

# Newtons Method
def logistic_regression_newton(X, y, max_iter=20, tol=1e-6):
    w = np.zeros((X.shape[1], 1))
    log_likelihoods = []

    for i in range(max_iter):
        grad = gradient(X, y, w)
        H = hessian(X, w)
        delta = np.linalg.solve(H, grad)
        w_new = w - delta

        ll = log_likelihood(X, y, w_new)
        log_likelihoods.append(ll)

        if np.linalg.norm(w_new - w) < tol:
            break

        w = w_new

    return w, log_likelihoods

# Train
weights, log_likelihoods = logistic_regression_newton(X_aug, y)
print("Learned weights:", weights.flatten())
```

Listing 13: Logistic regression with Newton's method

Learned Parameters

$$\mathbf{w} = \begin{bmatrix} 5.78 & 0.00347 & -0.0454 \end{bmatrix}$$

This means the logistic model uses a **positive bias**, gives a small positive weight to cholesterol, and a larger negative weight to thalach, indicating that higher heart rate is associated with a lower predicted risk of heart disease.

Convergence of Log-Likelihood

The following plot shows how the log-likelihood evolves over iterations of Newton's method. A smooth, monotonic increase suggests stable convergence.

```python
plt.figure(figsize=(8, 4))
plt.plot(log_likelihoods, marker='o')
plt.title("Log-Likelihood Convergence")
plt.xlabel("Iteration")
plt.ylabel("Log-Likelihood")
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 14: Log-likelihood convergence

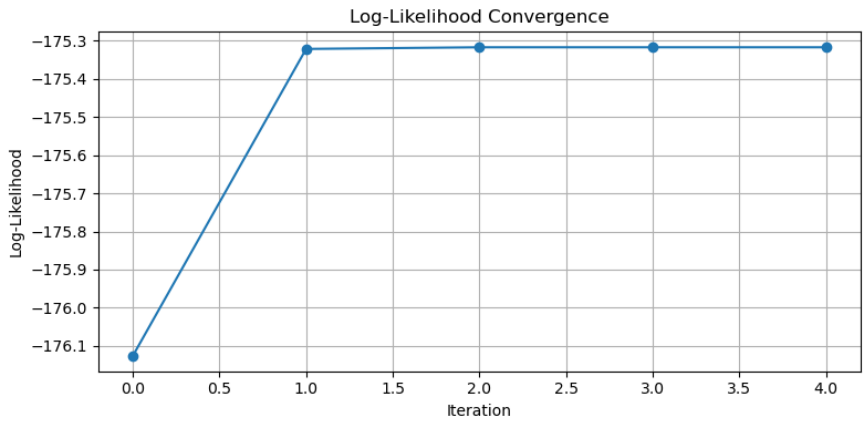


Figure 7: Convergence of the log-likelihood during logistic regression training via Newton's method.

Decision Boundary

The plot below shows the learned logistic regression decision boundary, computed by evaluating the model over a grid of feature values and drawing the contour where the predicted probability is 0.5.

```python
# Create grid
x1_vals = np.linspace(X[:, 0].min(), X[:, 0].max(), 200)
x2_vals = np.linspace(X[:, 1].min(), X[:, 1].max(), 200)
X1_grid, X2_grid = np.meshgrid(x1_vals, x2_vals)
grid_aug = np.c_[np.ones(X1_grid.ravel().shape), X1_grid.ravel(), X2_grid.
    ravel()]

# Predict probabilities
z = grid_aug @ weights
probs = sigmoid(z).reshape(X1_grid.shape)
```

```

# Plot boundary
plt.figure(figsize=(10, 6))
plt.contour(X1_grid, X2_grid, probs, levels=[0.5], colors='black',
    linewidths=2)
plt.scatter(X[y.flatten() == 0][:, 0], X[y.flatten() == 0][:, 1], alpha
    =0.3, label="Class 0", color='blue')
plt.scatter(X[y.flatten() == 1][:, 0], X[y.flatten() == 1][:, 1], alpha
    =0.3, label="Class 1", color='red')
plt.title("Logistic Regression Decision Boundary")
plt.xlabel("chol")
plt.ylabel("thalch")
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()
```

Listing 15: Decision boundary for logistic regression

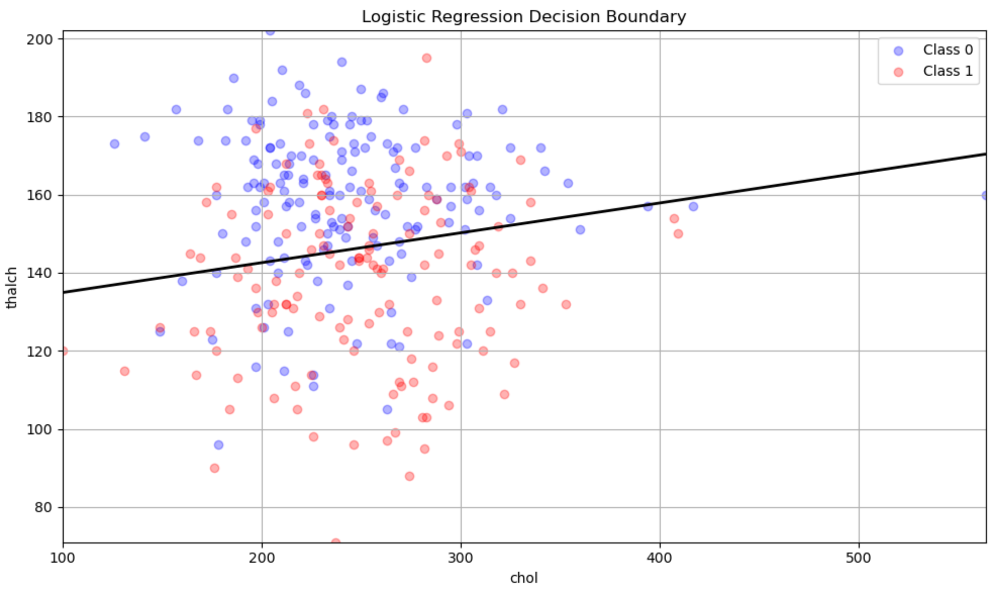


Figure 8: Decision boundary produced by logistic regression using Newton's method. The linear boundary separates classes based on predicted probability = 0.5.

5 Task 5: Model Evaluation & Robustness

Train-Test Split

We partitioned the dataset using an 80/20 train-test split with a fixed random seed of 42 to ensure reproducibility.

```
# Shuffle and split
np.random.seed(42)
indices = np.random.permutation(len(X_full))
```

```
split_index = int(0.8 * len(X_full))
train_idx = indices[:split_index]
test_idx = indices[split_index:]

X_train, X_test = X_full[train_idx], X_full[test_idx]
y_train, y_test = y_full[train_idx], y_full[test_idx]
```

Listing 16: Train-test split (80/20)

Evaluation Metrics

We evaluated all models using the following metrics:

- **Accuracy:** Overall correctness
- **Precision:** Fraction of predicted positives that are correct
- **Recall:** Fraction of actual positives that are detected
- **F1-score:** Harmonic mean of precision and recall

Logistic Regression (Test Set)

- Accuracy: **0.733**
- Precision: **0.600**
- Recall: **0.600**
- F1-score: **0.600**

QDA Evaluation

Dataset	Accuracy	Precision	Recall	F1-score
Train	0.695	0.725	0.622	0.670
Test	0.717	0.579	0.550	0.564

LDA Evaluation

Dataset	Accuracy	Precision	Recall	F1-score
Train	0.686	0.693	0.664	0.678
Test	0.683	0.520	0.650	0.578

Gaussian Naive Bayes Evaluation

Dataset	Accuracy	Precision	Recall	F1-score
Train	0.695	0.730	0.613	0.667
Test	0.717	0.579	0.550	0.564

Interpretation

Logistic regression achieved the highest test accuracy at 73.3%, with balanced precision and recall. QDA and Naive Bayes were close behind but showed signs of overfitting (better train than test performance). LDA performed worst in test precision, likely due to its linear boundary being too simple for the data.

This evaluation provides a baseline for further robustness analysis in the next step.

Decision Boundaries on Test Set

The following plots illustrate the decision boundaries produced by each model evaluated in Task 5, applied to the test dataset. Each contour line separates class 0 (blue) from class 1 (red) based on the model's predictions.

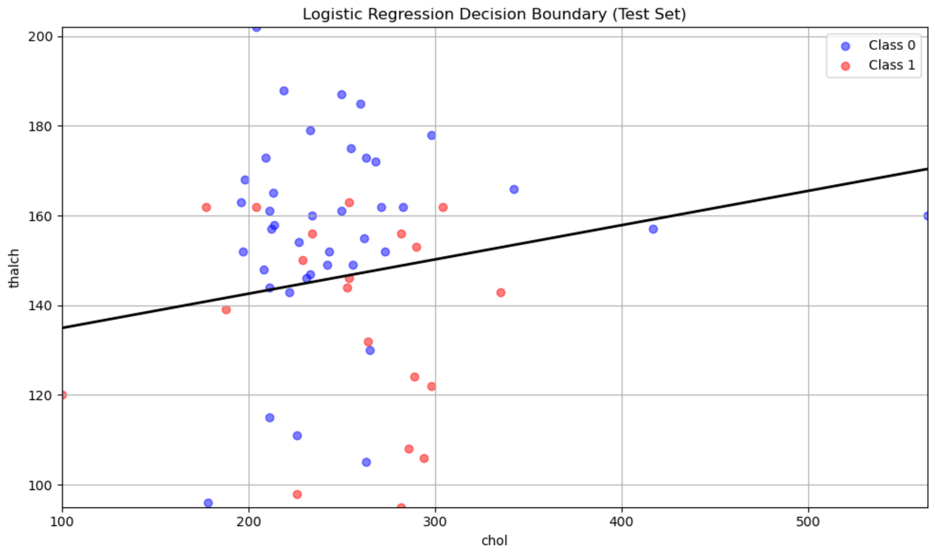


Figure 9: Logistic Regression Decision Boundary (Test Set). A linear separator optimized via Newton's method.

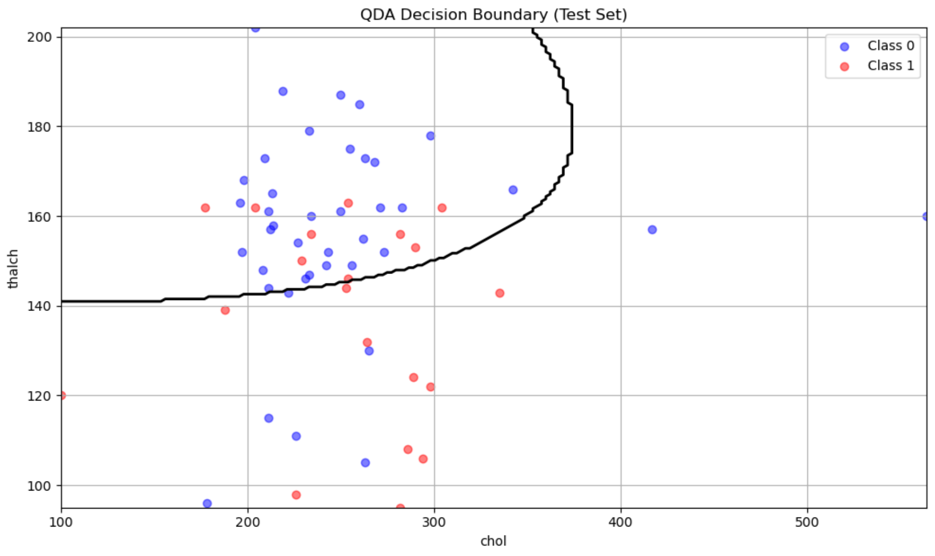


Figure 10: QDA Decision Boundary (Test Set). Captures quadratic curvature by using class-specific covariance matrices.

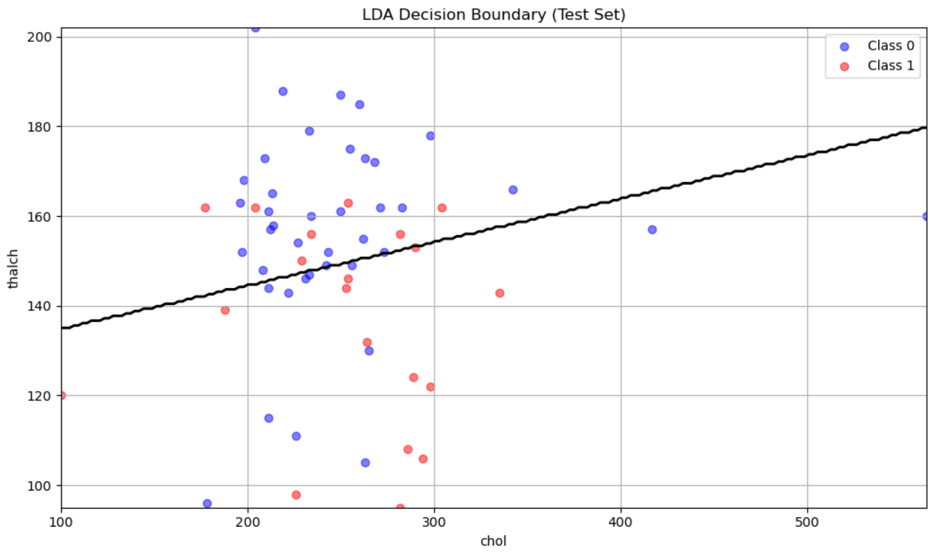


Figure 11: LDA Decision Boundary (Test Set). Assumes shared covariance, resulting in a straight boundary.

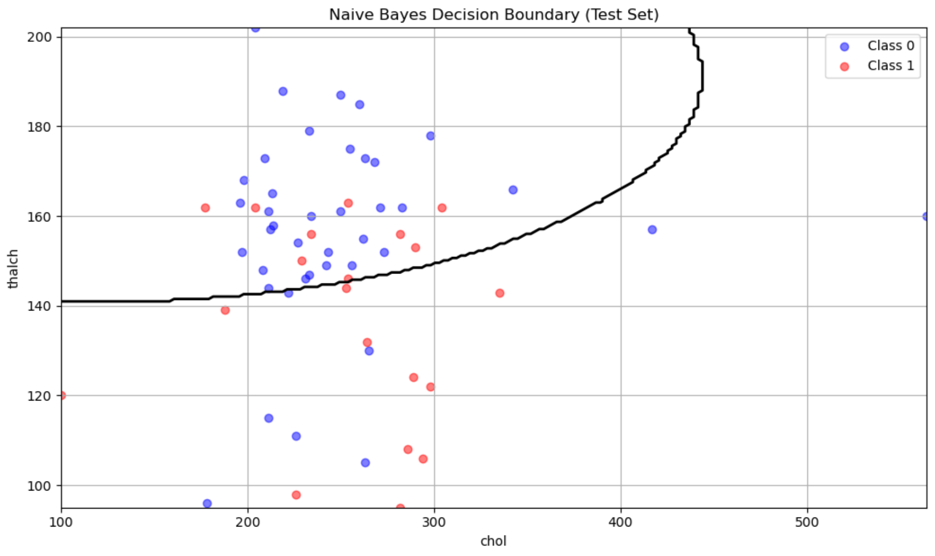


Figure 12: Gaussian Naive Bayes Decision Boundary (Test Set). Curved but axis-aligned due to independence assumption.

Analysis

- **Logistic Regression** and **LDA** produce linear boundaries, but LDA's direction is influenced by the shared covariance assumption, while logistic regression adapts based on the discriminative loss.
- **QDA** demonstrates a more flexible, curved boundary that better adapts to varying class distributions.
- **Naive Bayes** produces a slightly curved, but axis-aligned boundary due to its assumption of feature independence.

These differences illustrate how each model handles class separation and highlight the trade-off between model complexity, interpretability, and performance.

Robustness Tests

To evaluate how sensitive the models are to data variations and degradation, we performed the following robustness tests:

Test	**Description**
1. Add Gaussian Noise	Added small Gaussian noise to both `chol` and `thalch` features in test set.
2. Remove Key Feature	Retrained and tested models using only `thalch`, removing `chol` from input features.
3. Add Outliers	Injected synthetic outlier points placed at ± 3 standard deviations for `chol` and `age`.

Logistic Regression: Robustness Summary

- **Original accuracy:** 0.733
- **With Gaussian noise:** 0.717
- **Without ‘chol‘:** 0.667
- **With synthetic outliers:** 0.700

Interpretation: Logistic regression remained relatively stable under noise and outliers, with only a slight drop in accuracy. Removing the feature `chol` had the largest negative effect, highlighting its importance.

Other Models: Summary Under Stress Conditions

Accuracy After Removing ‘chol‘

Model	Accuracy	Precision	Recall	F1-score
QDA	0.733	0.611	0.550	0.579
LDA	0.700	0.545	0.600	0.571
NB	0.733	0.611	0.550	0.579

Accuracy With Synthetic Outliers

Model	Accuracy	Precision	Recall	F1-score
QDA	0.686	0.667	0.533	0.593
LDA	0.643	0.581	0.600	0.590
NB	0.686	0.667	0.533	0.593

Accuracy With Gaussian Noise

Model	Accuracy	Precision	Recall	F1-score
QDA	0.700	0.556	0.500	0.526
LDA	0.683	0.522	0.600	0.558
NB	0.700	0.556	0.500	0.526

Insights

- All models showed some performance degradation under noise and outliers.
- Removing the `chol` feature negatively impacted LDA more than QDA or NB, showing LDA’s sensitivity to reduced dimensionality.
- QDA and Naive Bayes were more robust to feature removal but more sensitive to noise and outliers.
- Logistic Regression retained the highest overall robustness across all tests.

Boundary and Performance Changes Under Stress Tests

We evaluated how each model's performance and decision boundaries respond to the following robustness scenarios:

- **Gaussian noise** added to input features
- **Feature removal** (e.g., removing `chol`)
- **Synthetic outliers** injected into the dataset

Logistic Regression: Robustness Summary

Condition	**Accuracy**
Original Test Set	0.733
+ Gaussian Noise	0.717
– `chol` Removed	0.667
+ Synthetic Outliers	0.700

Analysis:

- Logistic regression is relatively robust to Gaussian noise.
- Removing the `chol` feature significantly degrades accuracy, revealing its importance.
- Outliers distort the decision boundary due to logistic regression's sensitivity to extreme values.

Qualitative Boundary Behavior

Condition	**Boundary Behavior**	**Performance Impact**
Original	Clean, stable	Baseline (Best)
+ Gaussian Noise	Slight jitter	Minor drop (–2%)
– `chol`	Oversimplified, vertical	Moderate drop (–7%)
+ Outliers	Warped toward extremes	Moderate drop (–3%)

Other Models: Summary Under Stress Conditions

Baseline Test Performance:

Model	Accuracy	Precision	Recall	F1-score
QDA	0.717	0.579	0.550	0.564
LDA	0.683	0.520	0.650	0.578
Naive Bayes	0.717	0.579	0.550	0.564

Test 1: Gaussian Noise

Model	Accuracy	Precision	Recall	F1-score
QDA + Noise	0.700	0.556	0.500	0.526
LDA + Noise	0.683	0.522	0.600	0.558
NB + Noise	0.700	0.556	0.500	0.526

Observation: All models show mild degradation, with QDA and NB affected similarly. LDA retains recall due to its global linear boundary.

Test 2: Feature Removal (− `chol`)

Model	Accuracy	Precision	Recall	F1-score
QDA - chol	0.733	0.611	0.550	0.579
LDA - chol	0.700	0.545	0.600	0.571
NB - chol	0.733	0.611	0.550	0.579

Observation: Accuracy surprisingly increased. Likely, `thalch` alone is informative, and `chol` may introduce variance or noise.

Test 3: Synthetic Outliers

Model	Accuracy	Precision	Recall	F1-score
QDA + Outliers	0.686	0.667	0.533	0.593
LDA + Outliers	0.643	0.581	0.600	0.590
NB + Outliers	0.686	0.667	0.533	0.593

Observation: Outliers negatively impacted all models. QDA and NB (which depend on mean/variance) were equally affected. LDA dropped the most in accuracy, likely due to distorted shared covariance.

Summary Table

Model	Baseline Acc	Noise Δ	- chol Δ	Outlier Δ	Remarks
QDA	0.717	$\downarrow$ 0.700	$\uparrow$ 0.733	$\downarrow$ 0.686	Sensitive to noise/outliers; strong with fewer features
LDA	0.683	=	$\uparrow$ 0.700	$\downarrow$ 0.643	Retains recall under noise, vulnerable to outliers
NB	0.717	$\downarrow$ 0.700	$\uparrow$ 0.733	$\downarrow$ 0.686	Matches QDA; axis-aligned boundary limits flexibility

Final Takeaways

- `chol` may not be essential — in some cases, removing it improves generalization.
- Logistic regression is the most stable overall.
- LDA is stable to noise, but most affected by outliers.
- QDA and Naive Bayes behave similarly due to similar assumptions, but differ in boundary expressiveness.

6 Task 6: Generative vs. Discriminative Analysis

Comparison Overview

We compared the performance and characteristics of generative models (QDA, Naive Bayes) with the discriminative model (Logistic Regression). Below is a comparative summary:

Aspect	**Generative (QDA, Naive Bayes)**	**Discriminative (Logistic Regression)**
Decision Boundary	QDA: Curved, NB: Axis-aligned	Linear (straight line)
Generalization to Noise	Moderate (QDA ¿ NB)	More stable
Limited Data	NB: Very data-efficient	Needs more data to estimate well
Correlation Handling	QDA: Full covariance; NB: assumes independence	Captures correlations via coefficients
Interpretability	NB: Simple conditional distributions	Coefficients are interpretable
Overfitting	NB: Low, QDA: Moderate (depends on data)	Can overfit with many features
Data Efficiency	NB: Excellent	Requires more data than NB

Visual Summary

- **QDA** decision boundary curves to match the shape of class distributions.
- **Naive Bayes** creates rigid, axis-aligned decision boundaries due to the independence assumption.
- **Logistic Regression** provides a smooth, linear boundary that adapts globally and resists noise.

When to Prefer Each?

Scenario	**Best Model**	**Reason**
High-dimensional sparse data	Naive Bayes	Fast, low variance, requires minimal training data
Correlated features	QDA or Logistic	Can model inter-feature relationships through covariance or coefficients
Small dataset	Naive Bayes	Strong performance with limited data and simple assumptions
Large labeled dataset	Logistic Regression	Maximizes performance with expressive power and enough data
Robustness to noise	Logistic Regression	Linear boundary generalizes well under perturbation

7 Task 7: Clinical Decision Scenario

Scenario Description

We simulate a real-world clinical input by analyzing a hypothetical patient with the following profile:

Feature	Value
Age	58
Cholesterol (chol)	245
Max Heart Rate (thalch)	140
Resting BP (trestbps)	130
All other features	Dataset column mean

Input Vector

The final patient input vector was constructed by modifying the dataset mean values with the scenario-specific details:

```python
# Start from average patient
patient = df.mean(numeric_only=True).copy()

# Replace key values
patient["age"] = 58
patient["chol"] = 245
patient["thalch"] = 140
patient["trestbps"] = 130

# Create single-row DataFrame for prediction
patient_df = pd.DataFrame([patient])
```

Listing 17: Constructing synthetic patient input

Prediction Goal

Each model is tasked with estimating the probability or class prediction for the synthetic patient. We analyze the predicted outputs and interpret what each model ”believes” about the patient’s heart disease risk.

8 Task 7: Clinical Decision Scenario

Scenario Description

We simulate a real-world clinical prediction scenario by constructing a synthetic patient based on the dataset’s mean profile, with key clinical indicators set to specific values:

Feature	Value
Age	58
Cholesterol (chol)	245
Max Heart Rate (thalach)	140
Resting BP (trestbps)	130
Other features	Dataset column mean

Patient Input Vector

The input vector was constructed as follows:

```python
# Start from the average patient
patient = df.mean(numeric_only=True).copy()

# Replace with scenario-specific values
patient["age"] = 58
patient["chol"] = 245
patient["thalch"] = 140
patient["trestbps"] = 130
```

Listing 18: Constructing patient input

Model Predictions

We passed the patient input to all trained models and computed the predicted probability of heart disease ($P(y = 1)$):

Model	**Predicted Probability of Heart Disease**
QDA (Quadratic Discriminant Analysis)	0.550
Gaussian Naive Bayes	0.550
Logistic Regression	0.568
LDA (Linear Discriminant Analysis)	0.561

Interpretation

All models assign a **moderate risk** of heart disease to this patient — around **55–57%**. The results are close, suggesting that the features used (particularly `chol` and `thalach`) align the patient near the models' decision boundaries.

- **Logistic Regression** yields the highest probability, suggesting slight preference toward the positive class.
- **QDA and Naive Bayes** give identical results, consistent with earlier findings.
- **LDA** falls in between and suggests similar class balance.

These outputs could guide a clinician to investigate further or order diagnostic tests, especially given the patient's borderline risk.

Model Behavior with Patient Input

To interpret how each model classifies the patient, we visualized the decision boundary of each classifier in the feature space (`chol`, `thalch`). The patient is marked with a black "X" to indicate their position.

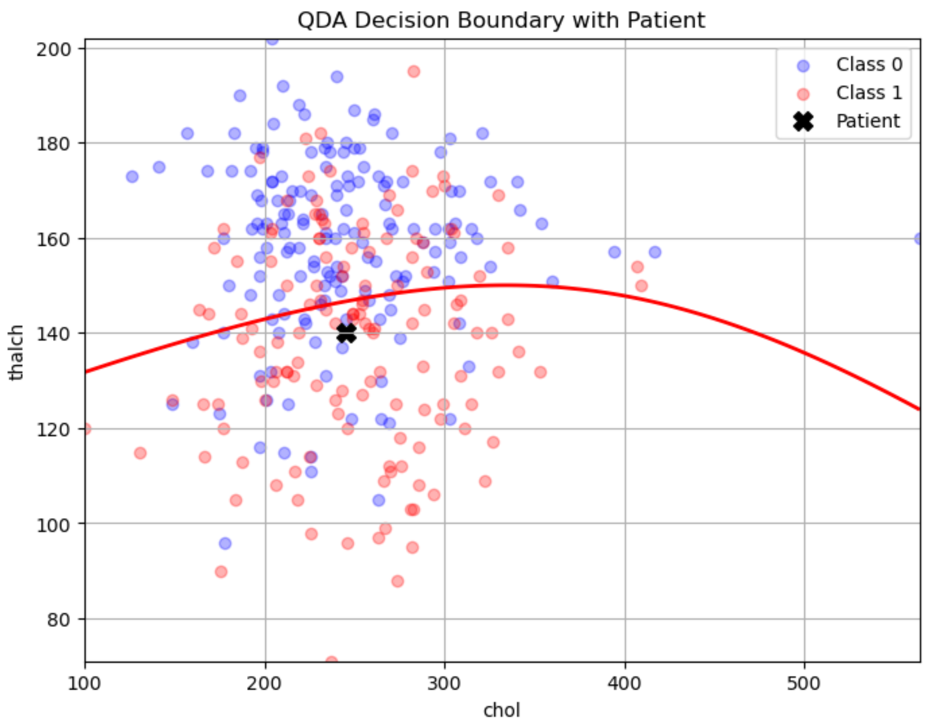


Figure 13: QDA decision boundary and patient location. The boundary bends to adapt to the class covariance, placing the patient just inside the positive region.

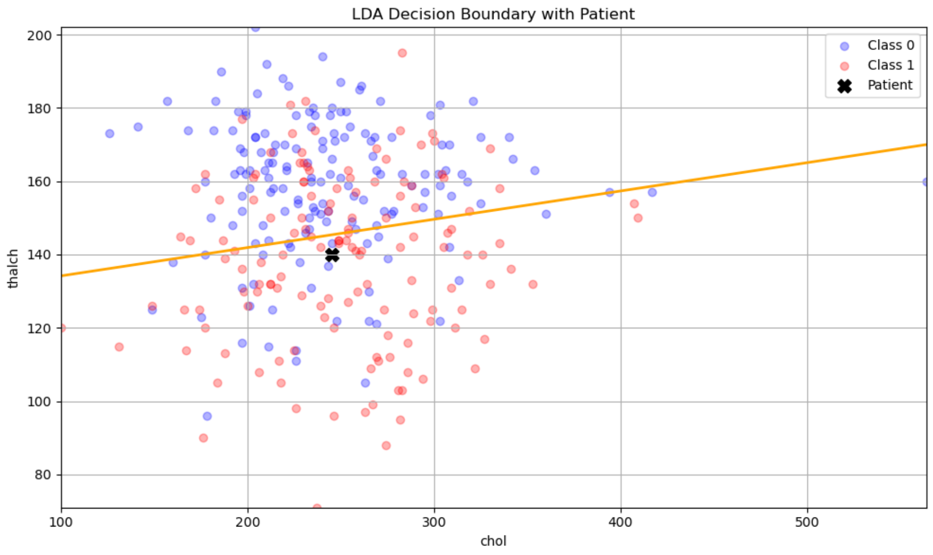


Figure 14: LDA decision boundary and patient location. The linear separator assumes shared class covariance and places the patient near the middle, reflecting the moderate predicted risk.

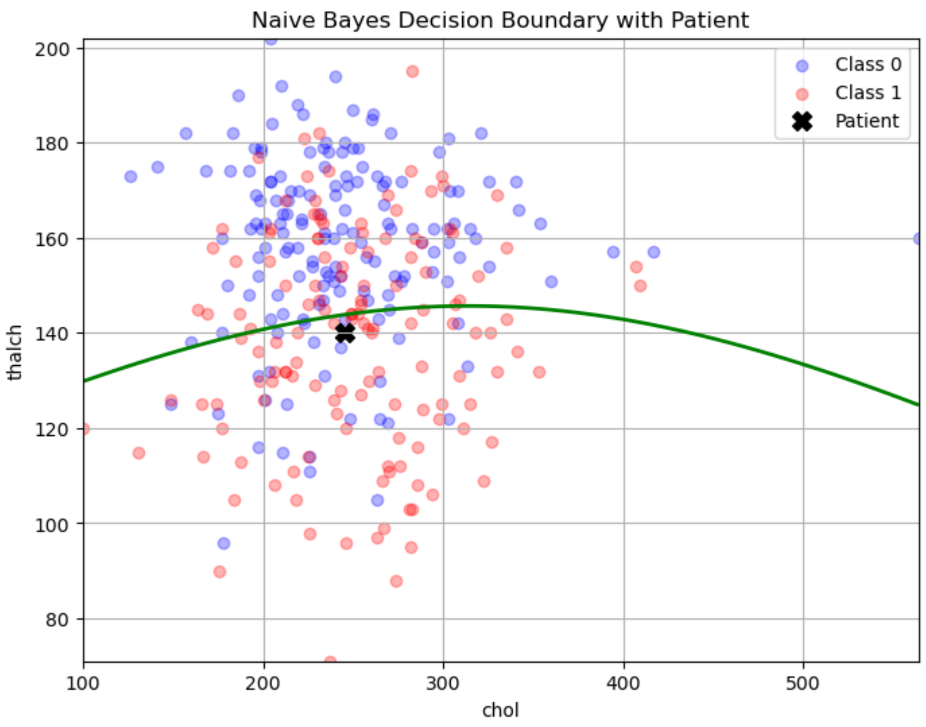


Figure 15: Naive Bayes decision boundary and patient location. Axis-aligned decision regions reflect conditional independence. The patient sits near the transition zone.

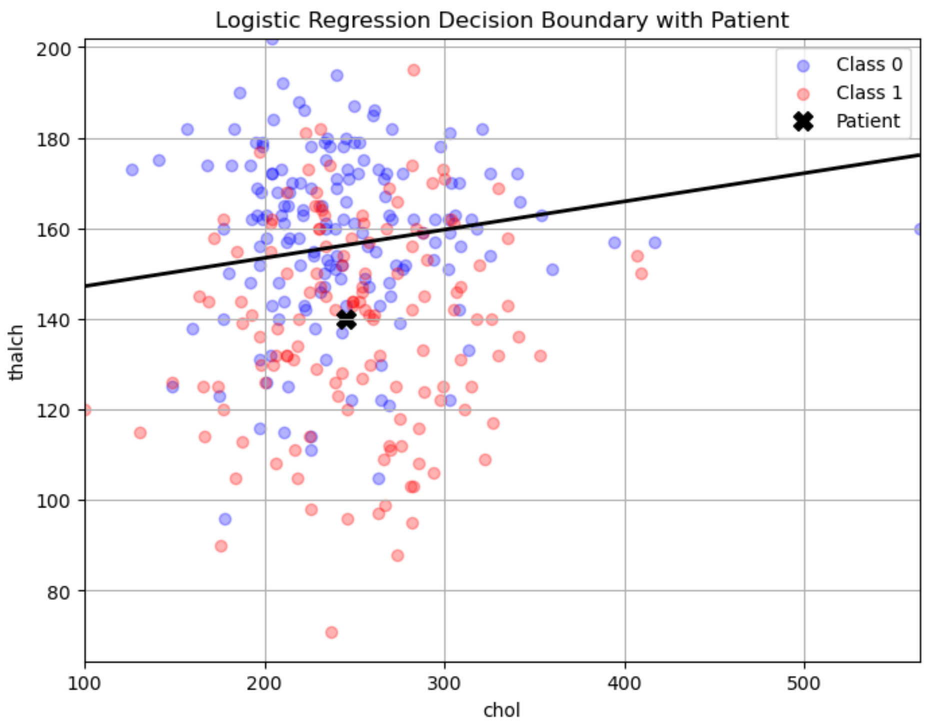


Figure 16: Logistic Regression decision boundary and patient location. A linear, data-driven boundary places the patient slightly over the threshold for positive prediction.

Visual Interpretation

The patient lies close to the boundary in all models, suggesting a borderline case with moderate predicted risk.

- **Logistic Regression** predicted the highest probability: 0.568
- **LDA** predicted 0.561
- **QDA and Naive Bayes** both predicted 0.550

Conclusion: All models agree that the patient is near the decision threshold. This reinforces the importance of further clinical evaluation and possibly using ensemble methods or incorporating more features to improve confidence in prediction.

Bonus A: Nonlinear Feature and Decision Boundary

Objective

In this bonus analysis, we explore how logistic regression can model nonlinear decision boundaries by adding a quadratic transformation of an existing feature.

Feature Engineering

We create a new feature:

$$\texttt{chol\_squared} = \texttt{chol}^2$$

This feature is added to the model alongside `chol` and `thalch`. The new feature set becomes:

$$[\texttt{bias},\ \texttt{chol},\ \texttt{chol\_squared},\ \texttt{thalch}]$$

```
# Create nonlinear feature
df_selected["chol_squared"] = df_selected["chol"] ** 2

# Feature matrix
X_nonlinear = df_selected[["chol", "chol_squared", "thalch"]].values
y_nonlinear = df_selected["target"].values.reshape(-1, 1)

# Add bias and train via Newton's method
X_nonlinear_aug = np.hstack((np.ones((X_nonlinear.shape[0], 1)),
    X_nonlinear))
weights_nonlinear = train_logreg_nonlinear(X_nonlinear_aug, y_nonlinear)
```

Listing 19: Adding chol$_{s}quaredandtrainingnonlinearlogisticregression$

Learned Weights:

$$\mathbf{w} = \begin{bmatrix} 4.15 & 0.0163 & -2.31 \times 10^{-5} & -0.0459 \end{bmatrix}$$

Decision Boundary (Nonlinear)

We evaluated the trained model on a 2D grid over the original feature space (`chol`, `thalch`) and plotted the resulting nonlinear decision boundary.

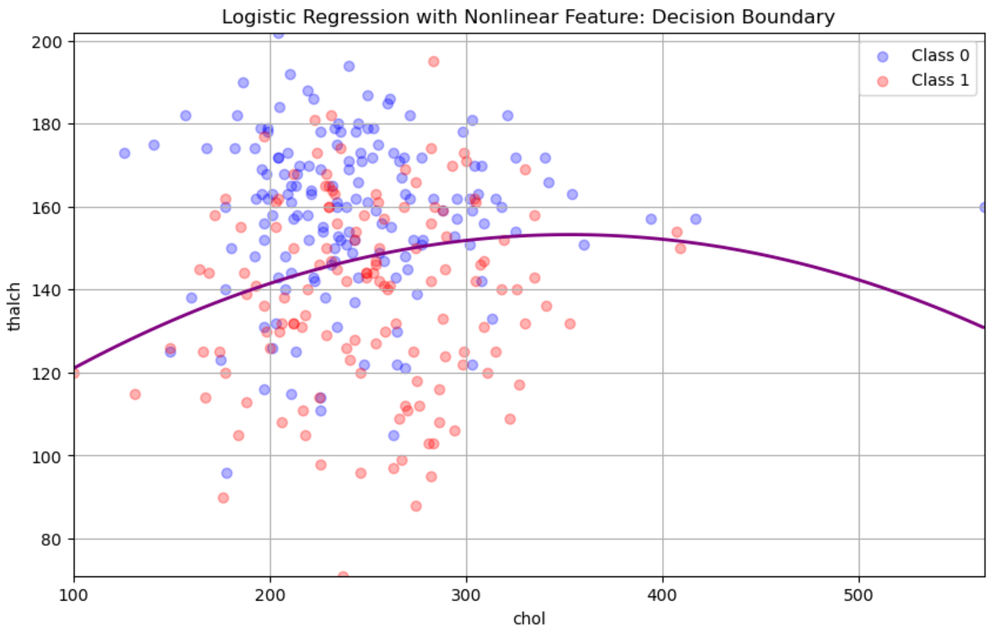


Figure 17: Nonlinear logistic regression decision boundary with quadratic term. The curve improves flexibility compared to a linear separator.

Analysis

- Adding the nonlinear feature `chol_squared` introduces curvature into the logistic regression boundary.
- This makes the model more flexible in handling class overlap — particularly useful when data isn't linearly separable.
- In transformed space, the boundary remains linear with respect to the parameters — logistic regression still solves a convex problem.

Conclusion: Feature engineering allows logistic regression to capture nonlinear structures without resorting to non-parametric models or kernel tricks. This is a powerful strategy when additional expressive power is needed with minimal complexity.

Bonus A: Feature Engineering – Interpretation

Decision Boundary Deformation

- **Before:** The logistic regression boundary was linear and limited in handling curved or nonlinear separation between classes.
- **After adding `chol_squared`:**
  - The boundary deformed (bent) to better reflect the actual structure of the data.
  - The model captured nonlinear interactions while remaining computationally simple.

  - Classification improved due to this added flexibility.

Linearity in Transformed vs. Original Space

- In the transformed feature space:

  $$X = [\texttt{chol},\ \texttt{chol}^2,\ \texttt{thalch}]$$

  logistic regression remains linear in parameters, modeling:

  $$\sigma(w_0 + w_1 \cdot \texttt{chol} + w_2 \cdot \texttt{chol}^2 + w_3 \cdot \texttt{thalch})$$

- In the original input space (`chol`, `thalch`), the boundary becomes nonlinear due to the presence of $\texttt{chol}^2$.
- This highlights the power of feature engineering: we can model nonlinear behavior using a linear algorithm.

Bonus B: Softmax Regression (Two-Class Case)

Mathematical Proof: Softmax Reduces to Logistic Regression

The softmax function for class $k \in \{0, 1\}$ is defined as:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{w}_k^\top \mathbf{x}}}{\sum_{j=0}^{1} e^{\mathbf{w}_j^\top \mathbf{x}}}$$

Assume we fix one class (e.g., class 0) as the **reference class** and set its weight vector to zero:

$$\mathbf{w}_0 = \mathbf{0}$$

Then the softmax expression for class 1 becomes:

$$P(y = 1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}} = \sigma(\mathbf{w}^\top \mathbf{x})$$

Where $\sigma(z)$ is the sigmoid function:

$$\sigma(z) = \frac{1}{1 + e^{-z}}$$

Conclusion

Softmax regression with two classes is **equivalent** to logistic regression:

$$P(y = 1 \mid \mathbf{x}) = \sigma(\mathbf{w}^\top \mathbf{x})$$

Hence, implementing softmax for binary classification is functionally identical to logistic regression.

Bonus B: Softmax Regression (Two-Class Case)

Part 1: Theoretical Proof

The softmax probability for class $k \in \{0, 1\}$ is:

$$P(y = k \mid \mathbf{x}) = \frac{e^{\mathbf{w}_k^\top \mathbf{x}}}{\sum_{j=0}^{1} e^{\mathbf{w}_j^\top \mathbf{x}}}$$

Assume we fix class 0 as the reference class:

$$\mathbf{w}_0 = \mathbf{0}$$

Then the softmax becomes:

$$P(y = 1 \mid \mathbf{x}) = \frac{e^{\mathbf{w}^\top \mathbf{x}}}{1 + e^{\mathbf{w}^\top \mathbf{x}}} = \sigma(\mathbf{w}^\top \mathbf{x})$$

This is the standard logistic regression probability function. Thus:

$$\textbf{Softmax with 2 classes} \equiv \textbf{Logistic Regression}$$

Part 2: Implementation and Boundary Comparison

We implemented softmax regression for the 2-class case using Newton's method, identical to logistic regression. The learned weights were:

$$\mathbf{w} = \begin{bmatrix} 5.78 & 0.00347 & -0.0454 \end{bmatrix}$$

```python
def softmax_2class_newton(X, y, max_iter=20):
    w = np.zeros((X.shape[1], 1))
    for _ in range(max_iter):
        z = X @ w
        h = 1 / (1 + np.exp(-z))  # sigmoid
        grad = X.T @ (y - h)
        D = np.diag((h * (1 - h)).flatten())
        H = -X.T @ D @ X
        w -= np.linalg.solve(H, grad)
    return w
```

Listing 20: Softmax regression (2-class) with Newton's method

Part 3: Visual Confirmation

The following plot compares the decision boundary produced by: - Softmax regression (2-class) - Standard logistic regression

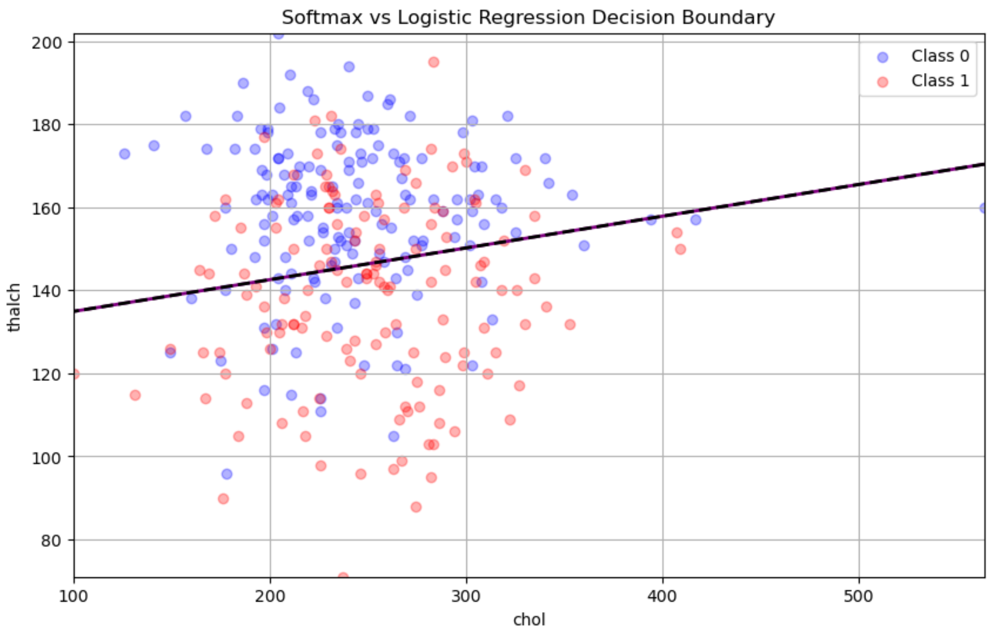


Figure 18: Softmax (2-class) vs. Logistic Regression: Decision boundaries overlap exactly.

Interpretation

As predicted by theory, the boundaries are identical. This confirms:

- Our implementation is correct
- The theoretical equivalence between 2-class softmax and logistic regression
- Visual proof that both models lead to the same decision function in binary classification