[Thesis Title Here]

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of
Master of Science (MSc)
in
[Department Name]

[Your Name]
[Student ID]
[University Name]

Month, Year

Declaration

I hereby declare that the work presented in this thesis is my own and has not been submitted elsewhere for the award of any degree.

[Your Name]
[Date]

Abstract

Acknowledgement

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

1.2 Problem Statement

1.3 Objectives

1.4 Significance of the Study

1.5 Thesis Organization

Chapter 2

THEORETICAL BACKGROUND

2.1 Overview

Antimicrobial Resistance (AMR) prediction from genomic data represents a critical intersection of machine learning and computational biology. This chapter provides a comprehensive theoretical foundation for the machine learning techniques, feature engineering methods, ensemble learning approaches, and model interpretability frameworks employed in this research. The theoretical concepts presented here underpin the methodology for predicting AMR from resistance gene profiles, offering both computational efficiency and biological interpretability. Understanding these fundamental concepts is essential for comprehending the rationale behind the proposed methodology and the interpretation of experimental results.

2.2 Machine Learning Fundamentals

2.2.1 Supervised Learning

Supervised learning is a machine learning paradigm where models learn from labeled training data to make predictions on unseen data. Given a dataset $D = \{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$, where x_i represents the feature vector and y_i represents the corresponding label, the objective is to learn a mapping function $f : X \to Y$ that accurately predicts the label y for a new input x.

In the context of binary classification, which is the focus of this research, the target variable $y \in \{0, 1\}$, where $y = 1$ typically represents the positive class (resistant) and $y = 0$ represents the negative class (susceptible).

2.2.2 Training and Testing

The supervised learning process involves partitioning the available data into training and testing sets. The training set is used to fit the model parameters, while the testing set provides an unbiased evaluation of the model's generalization capability on unseen data. A common practice is stratified splitting, which preserves the proportion of classes in both training and testing sets, particularly important when dealing with imbalanced datasets.

2.2.3 Cross-Validation

Cross-validation is a resampling technique used to assess model performance and tune hyperparameters while maximizing the use of available data. In k-fold cross-validation, the training data is divided into k equally sized folds. The model is trained k times, each time using $k-1$ folds for training and the remaining fold for validation. The final performance estimate is the average across all k iterations.

Stratified k-fold cross-validation extends this approach by ensuring that each fold maintains the same class distribution as the original dataset, which is particularly important for imbalanced classification problems.

2.3 Feature Engineering

2.3.1 Definition and Importance

Feature engineering is the process of creating new features or transforming existing features to improve model performance. Effective feature engineering can capture domain knowledge, reveal hidden patterns, and reduce the dimensionality of the problem space. In machine learning, the quality of features often has a greater impact on model performance than the choice of algorithm itself.

2.3.2 Feature Scaling and Normalization

Feature scaling transforms numerical features to a common scale without distorting differences in the ranges of values. Min-Max normalization is a scaling technique that transforms features to a specified range, typically $[0, 1]$:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)} \tag{2.1}$$

where x represents the original feature value, $\min(x)$ and $\max(x)$ are the minimum and maximum values of the feature across all samples, and x' is the normalized value.

This transformation is particularly important when features have different scales, as many machine learning algorithms are sensitive to feature magnitudes.

2.3.3 Derived Features

Derived features are constructed by combining or transforming existing features based on domain knowledge or statistical relationships. These engineered features can capture complex interactions or aggregate information that may not be explicitly represented in the raw data. Common approaches include:

- **Aggregation features**: Summarizing multiple features into a single value (e.g., sum, mean, count)
- **Ratio features**: Computing ratios between related features
- **Interaction features**: Capturing multiplicative or polynomial relationships between features
- **Binning features**: Discretizing continuous features into categorical bins

2.4 Feature Selection

2.4.1 Motivation for Feature Selection

Feature selection is the process of identifying and retaining the most relevant features while removing irrelevant or redundant ones. In high-dimensional datasets, feature selection addresses several critical challenges:

- **Curse of dimensionality**: As the number of features increases, the volume of the feature space grows exponentially, making data increasingly sparse and models prone to overfitting
- **Computational efficiency**: Fewer features reduce training time and memory requirements
- **Model interpretability**: Simpler models with fewer features are easier to understand and explain
- **Noise reduction**: Removing irrelevant features can improve model generalization by reducing noise

2.4.2 Filter Methods

Filter methods evaluate features independently of any specific machine learning algorithm, using statistical measures to score feature relevance. These methods are computationally efficient and model-agnostic.

ANOVA F-test

Analysis of Variance (ANOVA) F-test is a statistical technique used to assess whether the means of different groups are significantly different. For feature selection in classification, the F-statistic measures the ratio of between-class variance to within-class variance:

$$F = \frac{\text{Between-group variability}}{\text{Within-group variability}} \tag{2.2}$$

A higher F-statistic indicates that the feature exhibits greater separation between classes, suggesting higher predictive value. Features are typically ranked by their F-statistic, and those with p-values below a specified threshold are selected.

The advantages of ANOVA F-test include:

- Fast computation, suitable for high-dimensional data
- Model-independent, providing objective feature ranking
- Statistical foundation with interpretable significance levels

However, ANOVA assumes linear relationships and cannot capture complex non-linear interactions between features and the target variable.

2.4.3 Embedded Methods

Embedded methods perform feature selection during the model training process. The feature selection mechanism is integrated into the learning algorithm itself, allowing the model to determine which features contribute most to prediction accuracy.

Tree-Based Feature Importance

Decision tree-based algorithms naturally provide feature importance scores based on how much each feature contributes to reducing impurity (e.g., Gini impurity or entropy) across all splits in the tree. For ensemble methods like Random Forest and XGBoost, feature importance is aggregated across all trees in the ensemble.

Gain-Based Importance

In gradient boosting algorithms like XGBoost, gain-based importance measures the average improvement in the loss function contributed by each feature across all trees. Features that lead to larger reductions in loss receive higher importance scores:

$$\text{Importance(feature)} = \sum(\text{Gain from splits using this feature}) \tag{2.3}$$

Gain-based importance has several advantages:

- Captures non-linear relationships and complex interactions
- Considers feature contributions in the context of other features
- Naturally handles feature redundancy through the boosting process

2.4.4 Cumulative Importance Threshold

When using model-based feature importance, a cumulative importance threshold approach can be employed. Features are ranked by importance in descending order, and features are selected sequentially until their cumulative importance reaches a specified percentage (e.g., 85%) of the total importance:

$$\text{Selected Features} = \{\text{features where cumulative importance} \leq \text{threshold}\} \tag{2.4}$$

This approach balances model complexity and performance by retaining the most informative features while discarding those with minimal contribution.

2.4.5 Hybrid Feature Selection

Hybrid feature selection strategies combine multiple feature selection methods to leverage their complementary strengths. Common approaches include:

- **Intersection**: Selecting features identified by all methods (conservative approach)
- **Union**: Selecting features identified by any method (comprehensive approach)
- **Sequential**: Applying methods in stages, with each stage refining the feature set

The union-based approach is particularly effective when different methods capture distinct aspects of feature relevance. For example, combining statistical filter methods (which identify features with strong univariate relationships) with embedded methods (which capture multivariate interactions) can provide comprehensive feature coverage.

2.5 Class Imbalance Handling

2.5.1 The Class Imbalance Problem

Class imbalance occurs when the distribution of samples across classes is uneven, with one class (majority class) significantly outnumbering the other (minority class). This is a common challenge in real-world classification problems, including medical diagnosis, fraud detection, and anomaly detection.

Imbalanced datasets pose several challenges:

- **Bias toward majority class**: Models may achieve high overall accuracy by simply predicting the majority class
- **Poor minority class performance**: The minority class, often the class of interest, may be poorly predicted
- **Misleading evaluation metrics**: Accuracy alone can be misleading when classes are imbalanced

The imbalance ratio is defined as:

$$\text{Imbalance Ratio} = \frac{\text{Number of majority class samples}}{\text{Number of minority class samples}} \tag{2.5}$$

Ratios greater than 1.5:1 are generally considered imbalanced, with ratios above 3:1 representing severe imbalance.

2.5.2 Resampling Techniques

Resampling techniques address class imbalance by modifying the training data distribution. These methods can be categorized into oversampling (increasing minority class samples) and undersampling (decreasing majority class samples).

Random Oversampling

Random oversampling duplicates randomly selected minority class samples until class balance is achieved. While simple and effective, this approach can lead to overfitting

as the model may memorize the duplicated samples rather than learning generalizable patterns.

Random Undersampling

Random undersampling removes randomly selected majority class samples to balance the dataset. This approach can lead to information loss, as potentially informative majority class samples are discarded.

SMOTE (Synthetic Minority Over-sampling Technique)

SMOTE generates synthetic minority class samples by interpolating between existing minority class instances. For each minority class sample x, SMOTE:

1. Identifies k nearest minority class neighbors
2. Randomly selects one neighbor x_{nn}
3. Generates a synthetic sample along the line connecting x and x_{nn}:

$$x_{\text{synthetic}} = x + \lambda \times (x_{nn} - x) \tag{2.6}$$

where $\lambda \in [0, 1]$ is a random number.
SMOTE advantages:

- Creates diverse synthetic samples rather than duplicates
- Reduces overfitting risk compared to random oversampling
- Generalizes the decision boundary for the minority class

SMOTE limitations:

- May generate unrealistic samples in sparse, high-dimensional spaces
- Can create synthetic samples in majority class regions, increasing overlap
- Sensitive to noise and outliers in the minority class

ADASYN (Adaptive Synthetic Sampling)

ADASYN extends SMOTE by adaptively generating more synthetic samples for minority class instances that are harder to learn (i.e., those surrounded by majority class samples). This focuses the oversampling effort on the most challenging decision boundary regions.

Borderline-SMOTE

Borderline-SMOTE identifies minority class samples near the decision boundary (borderline samples) and generates synthetic samples only for these instances. This approach concentrates on the most critical region for classification while avoiding overgeneration of synthetic samples in well-separated regions.

2.5.3 Hybrid Resampling Methods

Hybrid methods combine oversampling and undersampling techniques to achieve both class balance and data quality improvement.

SMOTE-ENN (SMOTE with Edited Nearest Neighbors)

SMOTE-ENN first applies SMOTE to oversample the minority class, then uses Edited Nearest Neighbors (ENN) to remove samples whose class label differs from the majority of their k nearest neighbors. This cleaning step removes noisy and overlapping samples, creating clearer class boundaries.

SMOTE-Tomek (SMOTE with Tomek Links)

SMOTE-Tomek combines SMOTE oversampling with Tomek links removal. A Tomek link is a pair of samples (x_i, x_j) from different classes that are each other's nearest neighbors. Removing Tomek links eliminates ambiguous boundary samples and creates a clearer separation between classes.

The SMOTE-Tomek process:

1. Apply SMOTE to oversample the minority class
2. Identify all Tomek links in the resulting dataset
3. Remove samples involved in Tomek links

Advantages of SMOTE-Tomek:

- Balances classes while improving boundary clarity
- Removes potentially mislabeled or noisy samples
- Reduces overfitting risk compared to SMOTE alone
- Particularly effective for high-dimensional, sparse data

2.6 Classification Algorithms

2.6.1 Logistic Regression

Logistic Regression is a linear classification algorithm that models the probability of a binary outcome using the logistic (sigmoid) function. Despite its name, logistic regression is a classification algorithm, not a regression algorithm.

Model Formulation

The logistic regression model computes the probability that an instance belongs to the positive class:

$$P(y = 1 \mid x) = \frac{1}{1 + e^{-z}} \tag{2.7}$$

where $z = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_n x_n$

Here, β_0 is the intercept, and $\beta_1, \ldots, \beta_n$ are the feature coefficients learned during training. The sigmoid function maps the linear combination z to a probability value between 0 and 1.

Decision Boundary

The decision boundary is the set of points where $P(y = 1 \mid x) = 0.5$, corresponding to $z = 0$. Logistic regression creates a linear decision boundary in the feature space.

Advantages

- Computationally efficient and fast to train
- Produces well-calibrated probability estimates
- Interpretable coefficients indicating feature importance and direction
- Performs well when classes are approximately linearly separable
- Less prone to overfitting compared to complex models

Limitations

- Assumes linear relationship between features and log-odds
- Cannot capture complex non-linear patterns
- May underperform when decision boundaries are highly non-linear

2.6.2 Support Vector Machine (SVM)

Support Vector Machine is a powerful classification algorithm that finds the optimal hyperplane separating classes by maximizing the margin between them.

Linear SVM

For linearly separable data, SVM identifies the hyperplane that maximizes the distance (margin) between the closest points of each class (support vectors). The decision function is:

$$f(x) = \text{sign}(w \cdot x + b) \tag{2.8}$$

where w is the weight vector perpendicular to the hyperplane, and b is the bias term.

Kernel Trick

For non-linearly separable data, the kernel trick maps the input space to a higher-dimensional feature space where classes become linearly separable. Common kernels include:

- **Linear Kernel**: $K(x, x') = x \cdot x'$
- **Polynomial Kernel**: $K(x, x') = (x \cdot x' + c)^d$
- **Radial Basis Function (RBF) Kernel**: $K(x, x') = \exp(-\gamma \|x - x'\|^2)$

The RBF kernel is widely used due to its ability to handle non-linear relationships and its flexibility in controlling the decision boundary smoothness through the γ parameter.

Advantages

- Effective in high-dimensional spaces
- Memory efficient (only support vectors are stored)
- Versatile through different kernel functions
- Robust to overfitting, especially in high-dimensional spaces

Limitations

- Computationally expensive for large datasets
- Sensitive to kernel and hyperparameter selection
- Probability estimates require additional calibration
- Difficult to interpret compared to linear models

2.6.3 Decision Tree

Decision Trees are hierarchical models that make predictions by learning a series of decision rules inferred from the training data. The model structure resembles a tree, with internal nodes representing feature tests, branches representing decision outcomes, and leaf nodes representing class labels or predictions.

Tree Construction

Decision trees are constructed using a recursive partitioning process:

1. Select the best feature to split the data based on an impurity measure
2. Partition the data into subsets based on the selected feature
3. Recursively apply the process to each subset
4. Stop when a termination criterion is met (e.g., maximum depth, minimum samples per leaf, pure node)

Splitting Criteria

The quality of a split is evaluated using impurity measures:

Entropy (Information Gain)

Entropy measures the disorder or uncertainty in a dataset:

$$\text{Entropy}(S) = -\sum p_i \times \log_2(p_i) \tag{2.9}$$

where p_i is the proportion of samples belonging to class i. Information gain measures the reduction in entropy achieved by a split:

$$\text{Information Gain} = \text{Entropy(parent)} - \sum \frac{|S_{\text{child}}|}{|S_{\text{parent}}|} \times \text{Entropy}(S_{\text{child}}) \tag{2.10}$$

Gini Impurity

Gini impurity measures the probability of incorrectly classifying a randomly chosen sample:

$$\text{Gini}(S) = 1 - \sum p_i^2 \tag{2.11}$$

Lower Gini impurity indicates a purer node. The split that results in the lowest weighted average child node impurity is selected.

Advantages

- Highly interpretable with clear decision rules
- Requires minimal data preprocessing
- Handles both numerical and categorical features
- Captures non-linear relationships naturally
- Provides feature importance rankings

Limitations

- Prone to overfitting, especially with deep trees
- Unstable—small data changes can result in different trees
- Biased toward features with more levels
- May create overly complex trees that don't generalize well

2.6.4 Random Forest

Random Forest is an ensemble learning method that constructs multiple decision trees during training and outputs the class that is the mode of the individual tree predictions (for classification).

Algorithm

The Random Forest algorithm operates as follows:

1. **Bootstrap Sampling**: Create B bootstrap samples from the training data (sampling with replacement)

2. **Tree Construction**: For each bootstrap sample, grow a decision tree with the following modification:
    - At each node, randomly select m features from the total M features
    - Choose the best split among the m features (not all M features)
3. **Aggregation**: Combine predictions from all trees using majority voting (classification) or averaging (regression)

Key Parameters

- **Number of trees (n_estimators)**: More trees generally improve performance but increase computation time
- **Max features (m)**: Typical choices are $\sqrt{M}$ for classification, $M/3$ for regression
- **Max depth**: Controls tree complexity and overfitting
- **Min samples split/leaf**: Minimum samples required to split or form a leaf node

Feature Importance

Random Forest provides feature importance scores by measuring the average decrease in impurity contributed by each feature across all trees:

$$\text{Importance(feature)} = \frac{1}{B}\sum \text{Decrease in impurity from feature across all trees} \tag{2.12}$$

Advantages

- Reduces overfitting compared to individual decision trees
- Robust to noise and outliers
- Provides reliable feature importance estimates
- Handles high-dimensional data well
- Requires minimal hyperparameter tuning
- Parallelizable for efficient computation

Limitations

- Less interpretable than single decision trees
- Can be memory-intensive for large ensembles
- May not perform well on very small datasets
- Potentially biased toward features with more categories

2.6.5 XGBoost (eXtreme Gradient Boosting)

XGBoost is an optimized implementation of gradient boosting that has become one of the most successful machine learning algorithms for structured data. It builds an ensemble of decision trees sequentially, where each tree attempts to correct the errors of the previous trees.

Gradient Boosting Framework

Gradient boosting builds an additive model:

$$F(x) = \sum f_t(x) \tag{2.13}$$

where each f_t is a decision tree trained to minimize the loss function by fitting the negative gradient of the loss with respect to the previous model's predictions.

XGBoost Objective Function

XGBoost optimizes a regularized objective function:

$$\text{Obj} = \sum L(y_i, \hat{y}_i) + \sum \Omega(f_t) \tag{2.14}$$

where:

- $L(y_i, \hat{y}_i)$ is the loss function measuring prediction error
- $\Omega(f_t)$ is a regularization term penalizing model complexity

The regularization term is defined as:

$$\Omega(f_t) = \gamma T + \frac{\lambda}{2}\|w\|^2 \tag{2.15}$$

where T is the number of leaves, w are the leaf weights, γ controls leaf penalty, and λ controls weight penalty.

Key Features

1. **Regularization**: L1 (Lasso) and L2 (Ridge) regularization to prevent overfitting
2. **Tree Pruning**: Uses max_depth parameter and prunes trees backward using the γ parameter
3. **Handling Missing Values**: Built-in capability to learn the best direction for missing values
4. **Column Subsampling**: Randomly selects a subset of features for each tree, similar to Random Forest
5. **Row Subsampling**: Uses a fraction of training samples for each tree to reduce overfitting

Hyperparameters

Important XGBoost hyperparameters include:

- **n_estimators**: Number of boosting rounds (trees)
- **max_depth**: Maximum depth of each tree
- **learning_rate (eta)**: Shrinkage factor to prevent overfitting
- **subsample**: Fraction of samples used for training each tree
- **colsample_bytree**: Fraction of features used for training each tree
- **gamma**: Minimum loss reduction required to make a split
- **lambda**: L2 regularization term
- **alpha**: L1 regularization term

Advantages

- State-of-the-art performance on many datasets
- Built-in regularization reduces overfitting
- Handles missing values automatically
- Provides feature importance scores
- Supports custom loss functions and evaluation metrics
- Highly efficient and parallelizable implementation

Limitations

- Requires careful hyperparameter tuning
- Can overfit on small or noisy datasets
- Less interpretable than simpler models
- Sensitive to outliers in some configurations
- Sequential training limits parallelization compared to Random Forest

2.7 Ensemble Learning

2.7.1 Ensemble Learning Principles

Ensemble learning combines multiple base models (weak learners) to create a stronger, more robust predictor. The fundamental principle is that a group of models working together can outperform any individual model by:

- **Reducing variance**: Averaging predictions reduces overfitting
- **Reducing bias**: Combining diverse models can capture different patterns
- **Improving robustness**: Ensemble is less sensitive to noise and outliers in training data

Diversity Requirement

For an ensemble to be effective, the base models should make different errors—if all models make identical predictions, the ensemble provides no benefit. Diversity can be achieved through:

- Different algorithms (heterogeneous ensemble)
- Different training subsets (bagging, bootstrap sampling)
- Different feature subsets
- Different hyperparameter configurations

2.7.2 Voting Classifiers

Voting classifiers combine predictions from multiple models through a voting mechanism. There are two primary approaches:

Hard Voting

Hard voting predicts the class that receives the most votes from base models:

$$\hat{y} = \text{mode}\{C_1(x), C_2(x), \ldots, C_n(x)\} \tag{2.16}$$

where $C_i(x)$ is the class prediction from model i. Each model contributes equally, and the majority class is selected.

Soft Voting

Soft voting predicts the class with the highest average probability across all models:

$$\hat{y} = \arg\max \sum P_i(y = c \mid x) \tag{2.17}$$

where $P_i(y = c \mid x)$ is the probability estimate for class c from model i. Soft voting leverages probability estimates and often outperforms hard voting because it considers prediction confidence.

Weighted Voting

Weighted voting assigns different importance weights to different models:

$$\hat{y} = \arg\max \sum w_i \times P_i(y = c \mid x) \tag{2.18}$$

where w_i is the weight assigned to model i, typically based on model performance (e.g., validation accuracy, F1-score). The weights sum to 1.0:

$$\sum w_i = 1.0 \tag{2.19}$$

Weight selection can be performed through:

- Cross-validation performance
- Grid search optimization
- Inverse error weighting
- Stacking meta-learner

Advantages of Voting Ensembles

- Combines strengths of diverse algorithms

- Reduces variance and improves generalization
- More robust to individual model failures
- Simple to implement and interpret

Limitations

- Requires training multiple models (increased computation)
- May not improve performance if base models are highly correlated
- Weight selection can require extensive tuning

2.7.3 Bagging vs. Boosting

Bagging (Bootstrap Aggregating)

Bagging trains multiple instances of the same algorithm on different bootstrap samples (random sampling with replacement) and averages their predictions. Random Forest is a prime example of bagging applied to decision trees.

Characteristics:

- Parallel training (models are independent)
- Reduces variance
- Particularly effective for high-variance models (e.g., deep decision trees)

Boosting

Boosting trains models sequentially, with each new model focusing on correcting errors made by previous models. Examples include AdaBoost, Gradient Boosting, and XGBoost.

Characteristics:

- Sequential training (models are dependent)
- Reduces bias
- Can achieve higher accuracy than bagging
- More prone to overfitting if not regularized

2.8 Model Evaluation Metrics

2.8.1 Confusion Matrix

The confusion matrix is a fundamental tool for evaluating classification model performance. For binary classification, it is a 2×2 matrix with four components:

	Predicted Positive	**Predicted Negative**
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

- **True Positive (TP)**: Correctly predicted positive instances
- **True Negative (TN)**: Correctly predicted negative instances
- **False Positive (FP)**: Negative instances incorrectly predicted as positive (Type I error)
- **False Negative (FN)**: Positive instances incorrectly predicted as negative (Type II error)

2.8.2 Accuracy

Accuracy measures the proportion of correct predictions:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.20}$$

Advantages: Simple, intuitive, and easy to interpret

Limitations: Misleading for imbalanced datasets—a model predicting only the majority class can achieve high accuracy while failing to identify minority class instances

2.8.3 Precision

Precision (Positive Predictive Value) measures the proportion of positive predictions that are actually correct:

$$\text{Precision} = \frac{TP}{TP + FP} \tag{2.21}$$

Precision answers the question: "Of all instances predicted as positive, how many are truly positive?"

High Precision: Few false positives—important when false positives are costly

Use Cases: Spam detection (users tolerate missing spam but not blocking legitimate emails), medical screening (avoiding unnecessary treatments)

2.8.4 Recall (Sensitivity)

Recall (Sensitivity, True Positive Rate) measures the proportion of actual positive instances correctly identified:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{2.22}$$

Recall answers the question: "Of all actual positive instances, how many did we correctly identify?"

High Recall: Few false negatives—important when missing positive instances is costly **Use Cases**: Disease diagnosis (missing a disease is more harmful than a false alarm), fraud detection (catching all fraudulent transactions is critical)

2.8.5 F1-Score

The F1-score is the harmonic mean of precision and recall, providing a single metric that balances both:

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.23}$$

The harmonic mean penalizes extreme values more than the arithmetic mean, meaning high F1-score requires both high precision and high recall. **Advantages**:

- Single metric balancing precision and recall
- Robust to class imbalance
- More informative than accuracy for imbalanced datasets

F-beta Score Generalization

$$F_\beta = (1 + \beta^2) \times \frac{\text{Precision} \times \text{Recall}}{\beta^2 \times \text{Precision} + \text{Recall}} \tag{2.24}$$

- $\beta < 1$ $\beta < 1$: Emphasizes precision
- $\beta = 1$ $\beta = 1$: Balanced (F1-score)
- $\beta > 1$ $\beta > 1$: Emphasizes recall

2.8.6 ROC Curve and AUROC

ROC (Receiver Operating Characteristic) Curve

The ROC curve plots the True Positive Rate (Recall) against the False Positive Rate across all classification thresholds:

- **True Positive Rate (TPR)**: $\frac{TP}{TP+FN}$ = Recall
- **False Positive Rate (FPR)**: $\frac{FP}{FP+TN}$

Each point on the ROC curve represents a different classification threshold. A perfect classifier has a point at (0, 1)—zero false positives and 100

AUROC (Area Under ROC Curve)

AUROC measures the area under the ROC curve, providing a single scalar value representing overall model performance across all thresholds:

- AUROC = 1.0: Perfect classifier
- AUROC = 0.5: Random classifier (diagonal line)
- AUROC ¡ 0.5: Worse than random (predictions are inverted)

Interpretation: AUROC represents the probability that the model ranks a randomly chosen positive instance higher than a randomly chosen negative instance. **Advantages**:

- Threshold-independent evaluation
- Robust to class imbalance
- Comprehensive performance assessment across all thresholds

Limitations:

- May be overly optimistic for highly imbalanced datasets
- Does not reflect performance at a specific operating threshold

2.8.7 Precision-Recall Curve and AUPRC

Precision-Recall (PR) Curve

The PR curve plots Precision against Recall across different classification thresholds. Unlike the ROC curve, which uses False Positive Rate, the PR curve directly shows the trade-off between precision and recall.

AUPRC (Area Under Precision-Recall Curve)

AUPRC measures the area under the PR curve. It is particularly informative for imbalanced datasets where the positive class is rare.

Why AUPRC for Imbalanced Data?

For highly imbalanced datasets, AUROC can be misleadingly high because it gives equal weight to both classes. AUPRC focuses specifically on the positive (minority) class performance:

- High AUPRC: Model maintains high precision and recall for the positive class
- Low AUPRC: Model struggles to correctly identify the positive class

Comparison: AUROC vs. AUPRC

- AUROC: Balanced view, less sensitive to class imbalance
- AUPRC: Focuses on positive class, more sensitive to imbalance
- For imbalanced datasets, AUPRC is often more informative than AUROC

2.9 Model Interpretability and Explainability

2.9.1 Importance of Interpretability

Model interpretability refers to the degree to which a human can understand the cause of a model's predictions. Interpretability is crucial for:

- **Trust and adoption**: Clinicians and domain experts require understanding before deploying models
- **Debugging and validation**: Identifying when models rely on spurious correlations
- **Regulatory compliance**: Many domains require explainable predictions
- **Scientific insight**: Understanding which features drive predictions can reveal new knowledge

Intrinsic vs. Post-hoc Interpretability

- **Intrinsic interpretability**: Models that are inherently interpretable (e.g., linear regression, decision trees)
- **Post-hoc interpretability**: Explanation methods applied after model training (e.g., SHAP, LIME, permutation importance)

2.9.2 Permutation Feature Importance

Permutation importance measures the increase in model error when a feature's values are randomly shuffled, breaking the relationship between the feature and the target.

Algorithm

1. Train model on original data and compute baseline performance
2. For each feature:
    - Randomly permute the feature's values
    - Compute model performance on permuted data
    - Importance = Baseline performance - Permuted performance
3. Rank features by importance score

Characteristics

- Model-agnostic (works with any model)
- Considers feature's contribution in context of other features
- Captures both direct and indirect effects through feature interactions

Advantages

- Simple and intuitive
- Does not require model retraining
- Provides global feature importance

Limitations

- Can be computationally expensive for large datasets
- May be unreliable when features are highly correlated
- Does not explain individual predictions

2.9.3 SHAP (SHapley Additive exPlanations)

SHAP is a unified framework for interpreting predictions based on cooperative game theory. It assigns each feature an importance value (SHAP value) for a particular prediction, representing the feature's contribution to the difference between the actual prediction and the average prediction.

Shapley Values from Game Theory

In cooperative game theory, the Shapley value fairly distributes the total payout among players based on their contribution. SHAP adapts this concept to machine learning:

- **Players**: Features
- **Payout**: Model prediction
- **Contribution**: How much each feature contributes to the prediction

SHAP Value Definition

The SHAP value ϕ_j for feature j is computed as:

$$\phi_j = \sum_{S \subseteq F \setminus j} \frac{|S|!(M - |S| - 1)!}{M!} \times [f(S \cup j) - f(S)] \tag{2.25}$$

where:

- SS S is a subset of features (coalition)
- MM M is the total number of features
- f(S)f(S) f(S) is the model prediction using only features in SS S
- The sum is over all possible subsets SS S not containing feature jj j

SHAP Properties

SHAP values satisfy three desirable properties:

1. **Local Accuracy**: The sum of SHAP values equals the difference between the prediction and the expected value:

$$f(x) = \phi_0 + \sum \phi_j \tag{2.26}$$

2. **Consistency**: If a model changes so that a feature's contribution increases, its SHAP value should not decrease

3. **Additivity**: For ensemble models, SHAP values can be computed for individual models and aggregated

SHAP Explainer Types

Different SHAP explainers are optimized for specific model types: **TreeExplainer** Efficient exact computation for tree-based models (Decision Trees, Random Forest, XGBoost, LightGBM). Uses tree structure to compute SHAP values in polynomial time rather than exponential time. **LinearExplainer** Analytical computation for linear models (Linear Regression, Logistic Regression). SHAP values equal the feature coefficients multiplied by the feature values (after centering):

$$\phi_j = \beta_j \times (x_j - E[x_j]) \tag{2.27}$$

KernelExplainer Model-agnostic explainer that estimates SHAP values by sampling feature coalitions. Computationally expensive but works with any model.

2.9.4 Global vs. Local Explanations

Global Explanations

Global explanations describe overall model behavior across the entire dataset:

- **Mean Absolute SHAP Values**: Average $|\phi_j|$ across all instances indicates overall feature importance
- **SHAP Summary Plots**: Visualize feature importance and effect direction
- **Partial Dependence Plots**: Show the marginal effect of features on predictions

Local Explanations

Local explanations describe individual predictions:

- **Instance-level SHAP Values**: Show which features contributed to a specific prediction
- **Waterfall Plots**: Visualize how features incrementally shift prediction from base value
- **Force Plots**: Interactive visualization of feature contributions for individual predictions

2.9.5 SHAP Ensemble Aggregation

For ensemble models composed of multiple base learners, SHAP values can be aggregated to provide ensemble-level explanations. If an ensemble uses weighted voting with weights w_i for models $i = 1, \ldots, n$, the ensemble SHAP value for feature j is:

$$\phi_{j,\text{ensemble}} = \frac{\sum w_i \times \phi_{j,i}}{\sum w_i} \tag{2.28}$$

This weighted aggregation ensures that the ensemble SHAP values reflect the contribution weights of the constituent models, providing interpretability consistent with the ensemble prediction mechanism.

2.10 Hyperparameter Optimization

2.10.1 Hyperparameters vs. Parameters

- **Parameters**: Learned from data during training (e.g., neural network weights, tree split points)
- **Hyperparameters**: Set before training and control the learning process (e.g., learning rate, tree depth, regularization strength)

2.10.2 Grid Search

Grid search performs exhaustive search over a predefined hyperparameter grid. For each combination of hyperparameters:

1. Train the model using cross-validation

2. Compute average validation performance

3. Select the combination with the best performance

Advantages

- Guaranteed to find the best combination in the grid
- Parallelizable across different hyperparameter combinations
- Reproducible and systematic

Limitations

- Computationally expensive (exponential growth with number of hyperparameters)
- May miss optimal values between grid points
- Inefficient when some hyperparameters have little effect

2.10.3 Random Search

Random search samples hyperparameter combinations randomly from defined distributions. Often more efficient than grid search, especially when some hyperparameters are more important than others.

2.10.4 Cross-Validation for Hyperparameter Selection

Cross-validation is essential for unbiased hyperparameter selection. The process:

1. Split data into training and held-out test sets
2. Use cross-validation on training set to select hyperparameters
3. Train final model on full training set with selected hyperparameters
4. Evaluate on held-out test set

This ensures hyperparameters are not selected based on test set performance, which would lead to overfitting and overly optimistic performance estimates.

2.11 Data Leakage Prevention

2.11.1 Definition of Data Leakage

Data leakage occurs when information from outside the training dataset is used to create the model, leading to overly optimistic performance estimates that fail to generalize to real-world deployment.

2.11.2 Common Sources of Leakage

Leakage from Preprocessing

Applying preprocessing steps (normalization, feature selection, resampling) to the entire dataset before splitting leads to leakage because:

- Training set "sees" information from the test set
- Feature selection may select features specifically predictive of test set
- Normalization uses test set statistics

Correct Approach: Fit preprocessing on training set only, then apply to test set

Leakage from Resampling

Applying class imbalance resampling (SMOTE, oversampling) to both training and test sets causes:

- Synthetic samples in test set are generated using training set information
- Test set no longer represents real-world distribution
- Performance estimates are inflated

Correct Approach: Apply resampling only to training set after train-test split

2.11.3 Train-Test Split Protocol

The correct protocol to prevent leakage:

1. **Split Data**: Divide into train and test sets (stratified to preserve class distribution)
2. **Fit Preprocessing**: Compute preprocessing parameters (e.g., normalization, feature selection) using only training data

3. **Apply Preprocessing**: Apply fitted preprocessing to both train and test sets
4. **Apply Resampling**: Apply class imbalance handling only to training set
5. **Train Model**: Train on preprocessed, resampled training set
6. **Evaluate**: Evaluate on preprocessed (but not resampled) test set

This protocol ensures the test set remains completely unseen and representative of real-world deployment conditions.

2.12 Binary Classification Formulation

2.12.1 Problem Definition

Binary classification is the task of assigning instances to one of two mutually exclusive classes. Formally:

- **Input**: Feature vector $x \in \mathbb{R}^n$
- **Output**: Class label $y \in \{0, 1\}$
- **Objective**: Learn function $f : \mathbb{R}^n \to \{0, 1\}$ that minimizes prediction error

2.12.2 Decision Threshold

Most classifiers output a continuous score or probability $P(y = 1 \mid x)$. The predicted class is determined by comparing to a decision threshold τ:

$$\hat{y} = \begin{cases} 1 & \text{if } P(y = 1 \mid x) \geq \tau \\ 0 & \text{otherwise} \end{cases} \tag{2.29}$$

The default threshold is typically $\tau = 0.5$, but can be adjusted based on:

- Class imbalance
- Cost of false positives vs. false negatives
- Desired precision-recall trade-off

2.12.3 Probability Calibration

Well-calibrated classifiers produce probability estimates that reflect true likelihood. For example, if a classifier predicts P(y=1)=0.7P(y = 1) = 0.7 P(y=1)=0.7 for 100 instances, approximately 70 should be positive class.

Methods for calibration:

- **Platt Scaling**: Fits a logistic regression on classifier outputs
- **Isotonic Regression**: Fits a non-parametric monotonic function

Logistic Regression naturally produces well-calibrated probabilities, while tree-based models (especially deep trees) may require calibration.

2.13 Chapter Summary

This chapter provided a comprehensive theoretical foundation for the machine learning techniques employed in antimicrobial resistance prediction. Key concepts covered include:

1. **Supervised Learning**: Training models on labeled data for binary classification
2. **Feature Engineering**: Creating the R-Score to capture cumulative resistance burden
3. **Feature Selection**: Hybrid approach combining statistical (ANOVA) and model-based (XGBoost) methods
4. **Class Imbalance Handling**: Resampling techniques with focus on SMOTE-Tomek
5. **Classification Algorithms**: Logistic Regression, SVM, Decision Tree, Random Forest, and XGBoost
6. **Ensemble Learning**: Weighted soft voting to combine diverse classifiers
7. **Evaluation Metrics**: Comprehensive assessment using accuracy, precision, recall, F1-score, AUROC, and AUPRC
8. **Model Interpretability**: Permutation importance and SHAP for explainable predictions
9. **Hyperparameter Optimization**: Grid search with cross-validation

10. **Data Leakage Prevention**: Proper train-test protocol to ensure unbiased evaluation

Chapter 3

Literature Review

3.1 Antimicrobial Resistance Overview

3.2 Machine Learning in Genomics

3.3 Related Works

Table 3.1: Summary of the main characteristics of the study.

Authors and Country	Number of Microbial Strains	Antibiotics Studied	Critical and High-Priority Pathogens Studied	Machine Learning Model	Assessr Perforr
This Study (2025), Bangladesh	6,100	Meropenem, Doripenem, Imipenem, Ertapenem	*Escherichia coli*, *Klebsiella pneumoniae*, *Pseudomonas aeruginosa*	Logistic Regression, Decision Tree, Random Forest, XGBoost, SVM, R-Blend Ensemble	Accurac cision, F1-score Confusic trix
Sunuwar & Azad (2021), USA	724	Doripenem, Ertapenem, Imipenem, Meropenem	*Escherichia coli*, *Klebsiella pneumoniae*, *Pseudomonas aeruginosa*	LR, gNB, SVM, DT, RF, KNN, LDA, mNB, AdaBoost, GBDT, ETC, BG	Recall, cision, k-fold validatic

Chapter 4

Methodology

4.1 Overview

This chapter presents a comprehensive methodology for predicting Antimicrobial Resistance (AMR) using machine learning techniques applied to resistance gene profiles. The proposed approach demonstrates that resistance gene-based prediction can achieve performance comparable to, or exceeding, whole-genome sequence (WGS)-based methods while maintaining computational efficiency and biological interpretability.

The methodology encompasses six key stages:

1. Dataset Collection and Preprocessing
2. Feature Engineering
3. Feature Selection
4. Class Imbalance Handling
5. Model Training and Ensemble Construction
6. Model Interpretability and Explainability

The pipeline addresses specific challenges of AMR genomic data, such as high dimensionality, sparsity of binary gene matrices, and class imbalance. Figure **??** illustrates the complete workflow of the proposed methodology.

4.2 Dataset Collection and Preprocessing

4.2.1 Data Source

The study utilized antimicrobial susceptibility testing (AST) data and genomic profiles from the NCBI Pathogen Detection Database. Resistance gene annotations were obtained via the AMRFinderPlus pipeline. Twelve datasets were compiled:

- *Klebsiella pneumoniae*: Doripenem, Ertapenem, Imipenem, Meropenem
- *Escherichia coli* and *Shigella*: Doripenem, Ertapenem, Imipenem, Meropenem
- *Pseudomonas aeruginosa*: Doripenem
- *Salmonella enterica*: Streptomycin, Kanamycin
- *Campylobacter jejuni*: Clindamycin

4.2.2 Binary Gene Matrix Construction

Each isolate was represented as a binary gene presence vector $\mathbf{x}_i \in \{0,1\}^n$. Detection completeness metadata (e.g., COMPLETE, PARTIAL) was encoded as additional binary features. The label $y_i \in \{0,1\}$ denotes resistance status.

4.3 Feature Engineering

4.3.1 Resistance Gene Load Score (R-Score)

The R-Score is computed as:

$$\text{R-Score}_i = \sum_{j=1}^{n} x_{ij}$$

It is then normalized using min-max scaling:

$$\text{R-Score}'_i = \frac{\text{R-Score}_i - \min(\text{R-Score})}{\max(\text{R-Score}) - \min(\text{R-Score})}$$

4.3.2 Justification and Impact Analysis

Inclusion of R-Score improved model performance and provided biologically interpretable separation between resistant and susceptible classes.

4.4 Feature Selection

4.4.1 Stage 1: ANOVA F-test

Univariate statistical filtering was performed using ANOVA F-statistics. Features with $p \leq 0.30$ were retained.

4.4.2 Stage 2: XGBoost Importance

An XGBoost model (400 trees, max depth 6) was used to compute feature importances. Features accounting for 85% cumulative importance were selected.

4.4.3 Final Feature Set

Selected features were combined as:

$$S_{\text{final}} = S_{\text{ANOVA}} \cup S_{\text{XGB}}$$

4.5 Class Imbalance Handling

4.5.1 Resampling Strategy

The study compared several strategies (SMOTE, ADASYN, etc.) and selected SMOTE-Tomek based on classification performance and biological plausibility.

4.6 Model Training and Ensemble Construction

4.6.1 Base Models

Evaluated models included:

- Logistic Regression (LogR)
- Support Vector Machine (SVM)
- Decision Tree (DT)
- Random Forest (RF)
- XGBoost (XGB)

4.6.2 R-Blend Ensemble

R-Blend is a soft-voting ensemble combining DT, LogR, and XGB with weights [1.0, 1.5, 1.0]:

$$P(y = c|\mathbf{x}) = \frac{\sum_i w_i \cdot P_i(y = c|\mathbf{x})}{\sum_i w_i}$$

4.7 Evaluation Metrics

The following metrics were used:

- Accuracy
- Precision
- Recall
- F1-Score
- AUROC
- AUPRC

4.8 Model Interpretability and Explainability

4.8.1 Permutation Importance

Assesses drop in performance when shuffling feature values.

4.8.2 SHAP Values

Computed using TreeExplainer (for DT/XGB) and LinearExplainer (for LogR):

$$f(\mathbf{x}) = \phi_0 + \sum_j \phi_j(x_j)$$

4.8.3 Ensemble SHAP Aggregation

$$\phi_j^{\text{ensemble}}(\mathbf{x}) = \frac{\sum_i w_i \cdot \phi_j^{(i)}(\mathbf{x})}{\sum_i w_i}$$

4.9 Experimental Design and Validation

- Ablation studies on R-Score, resampling, and ensemble impact
- Validation across 12 datasets with mean & std metrics
- Comparison with published methods (Her & Wu, Yasir et al.)

4.10 Implementation

Code was implemented in Python 3 using scikit-learn, XGBoost, imbalanced-learn, SHAP, pandas, and matplotlib. Experiments were run in Google Colab with GPU acceleration.

Chapter 5

Results

5.1 Descriptive Statistics

5.2 Model Performance

5.3 Feature Importance

Chapter 6

Discussion

6.1 Interpretation of Findings

6.2 Comparison with Previous Studies

6.3 Limitations

Chapter 7

Conclusion and Future Work

7.1 Conclusion

7.2 Future Research Directions

Bibliography

[1] Y. Chang, D. Lee, and H. Kim. Alignment-free identification of clones in b-cell receptor repertoires using tf-idf weighted k-mers. *Frontiers in Immunology*, 12: 791377, 2021. doi: 10.3389/fimmu.2021.791377.

[2] A. Kumar, S. Banerjee, and P. Roy. Bioset2vec: Extraction of k-mer dictionaries from biological sequences via tf-idf. *BMC Bioinformatics*, 24:62, 2025. doi: 10.1186/s12859-025-06261-7.

[3] Pavel Májek, Fajfr Oldřich, Náplavová Jana, Milan Kolář, and Jansa Petr. Genome-wide mutation scoring for machine-learning-based antimicrobial resistance prediction. *International Journal of Molecular Sciences*, 22(23):13049, 2021. doi: 10.3390/ijms222313049.

[4] NCBI Pathogen Detection. Pathogen detection isolate browser. URL `https://www.ncbi.nlm.nih.gov/pathogens`. Accessed: 2025-12-01.

[5] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, et al. Scikit-learn: Machine learning in python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[6] E. Rannon, T. Smith, and J. Lee. Dramma: A multifaceted machine-learning approach for novel antimicrobial resistance gene detection. *Microbiome*, 13(1):67, 2025. doi: 10.1186/s40168-025-1567-9.

[7] J. Sunuwar and R. Azad. Identification of novel antimicrobial resistance genes using machine learning, homology modeling, and molecular docking. *Microorganisms*, 10 (11):2102, 2022. doi: 10.3390/microorganisms10112102.

[8] Jitendra Sunuwar and Rajeev Azad. A machine learning framework to predict antibiotic resistance traits and yet unknown genes. *Briefings in Bioinformatics*, 22 (6):bbab179, 2021. doi: 10.1093/bib/bbab179.

[9] R. Zhang and L. Wang. Tf-idf based alignment-free methods for dna sequence comparison. *Computational Biology and Chemistry*, 94:107583, 2021. doi: 10.1016/j.compbiolchem.2021.107583.

Appendix A

Appendix A

Appendix B

Appendix B