Task 2: Healthcare classification (classification task)

Jana Ghoniem 2022/08912
Habiba Darwish 2022/06738
Shahd Emad 2022/02743
Haya Walid 2022/02798

Supervised by: Dr. Manal Tantawy

Contents

1 Introduction

1.1 Project Description

This project focuses on the classification of medical test outcomes into three distinct categories—Normal, Abnormal, and Inconclusive—using a range of machine learning algorithms. The workflow is structured into two main phases: the initial phase emphasizes thorough data preprocessing and exploratory statistical analysis, while the subsequent phase involves dimensionality reduction using Principal Component Analysis (PCA), model training, and performance evaluation. A variety of classification techniques are implemented, including Logistic Regression, Multi-Layer Perceptron (MLP), Support Vector Machines (both linear and with Gaussian kernels), Naïve Bayes, and ensemble approaches such as Random Forest or XGBoost. For the kernel-based SVM, Particle Swarm Optimization is employed to fine-tune the hyperparameters. Model performance is assessed using key evaluation metrics such as Precision, Recall (Sensitivity), F1-Score, and Average Accuracy, in addition to analyzing learning curves to understand model generalization. The final outcomes of the project include a complete Python-based machine learning pipeline, a comprehensive technical report, and a submission to Kaggle for comparative benchmarking.

2 Dataset Overview

The dataset used in this research paper comprises structured healthcare records detailing various attributes related to patient admissions. Each entry corresponds to an individual patient and includes personal information such as the patient's name, age, and gender, along with clinical and administrative details. The dataset captures key medical indicators including the patient's blood type and primary medical condition (e.g., Diabetes, Hypertension, Asthma), as well as the date of admission and discharge. Information about the attending doctor and the hospital where treatment was provided is also included.

Furthermore, the dataset records the patient's insurance provider (e.g., Aetna, Cigna, Medicare), the billing amount for healthcare services, and the room number assigned during their stay. Admission types are categorized as Emergency, Elective, or Urgent, indicating the urgency and nature of the visit. Additionally, details on prescribed medications such as Aspirin or Lipitor are listed, alongside the results of medical tests conducted during the admission, labeled as Normal, Abnormal, or Inconclusive. This comprehensive dataset provides a valuable foundation for analyzing healthcare patterns, predicting outcomes, and developing data-driven clinical decision-making tools.

3 Data Preparation

Data preparation is a foundational phase in the analytical workflow, encompassing exploratory data analysis (EDA), handling of missing values, outlier identification, and examination of the target variable. This section outlines the methodological steps taken to ensure data quality and readiness for machine learning modeling.

3.1 Preprocessing and Exploratory Data Analysis

3.1.1 Initial Data Assessment

An initial exploratory analysis was conducted to assess data structure and quality. Using Python's `pandas` library, the `df.info()` function provided insights into data types, memory usage, and null values. This assessment highlighted substantial missing data in key variables such as `Blood Type`, `Insurance Provider`, and `Hospital`.
No duplicate patient records were found, ensuring data integrity. The `ID` column, serving no predictive function, was excluded to reduce redundancy and avoid overfitting.

3.1.2 Categorical Feature Analysis

Categorical variables such as `Gender`, `Blood Type`, and `Medical Condition` were visualized using bar plots to examine class frequencies and potential imbalances.

Under representation in categories such as `AB` blood type suggested possible sampling biases. Analysis also revealed patterns in prescription practices by condition and demographic group.

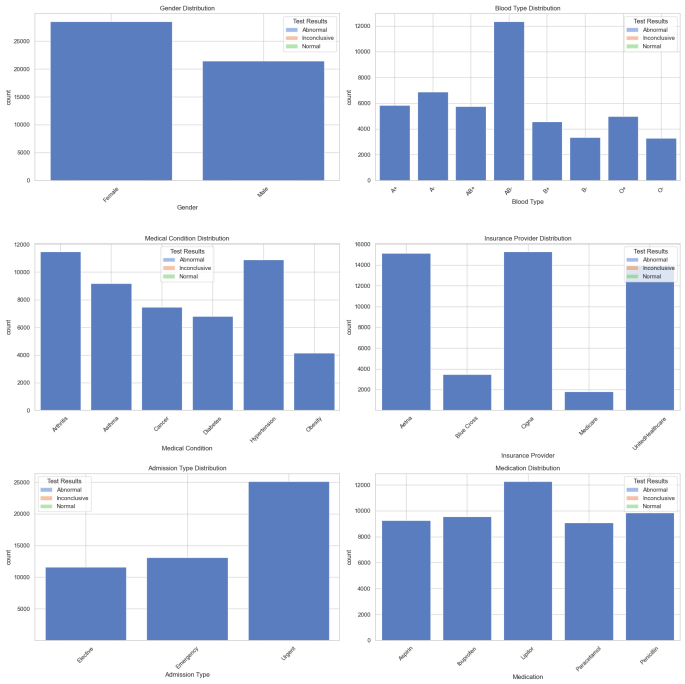

Figure 1: Sentiment Analysis Workflow

3.1.3 Numerical Feature Analysis

Numerical features, including `Age`, `Billing Amount`, and `Room Number`, were examined via histograms, boxplots, and descriptive statistics.

`Billing Amount` displayed a right-skewed distribution with anomalous negative values, indicating entry errors. Outliers were detected across all numerical variables, prompting further investigation.

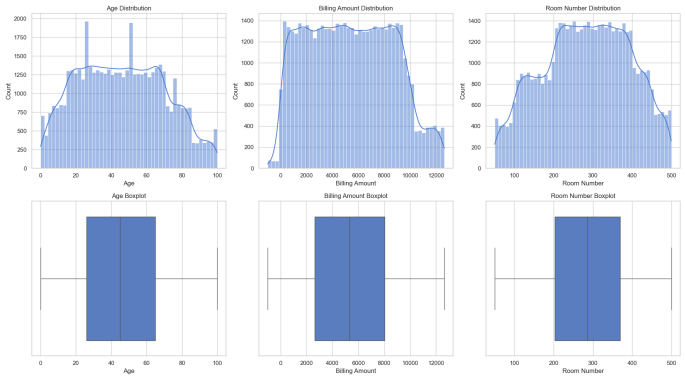

Figure 2: Sentiment Analysis Workflow

3.1.4 Correlation Analysis

A heatmap was generated to assess linear correlations between numerical features. Most correlations were weak to moderate, indicating low multicollinearity.

`Billing Amount` and `Room Number` showed limited correlation with the target variable `Test Results`, underscoring the need to explore non-linear relationships and feature interactions.

3.2 Missing Value Treatment

A feature-specific imputation strategy was employed to address missing values, balancing data completeness with preservation of original patterns.

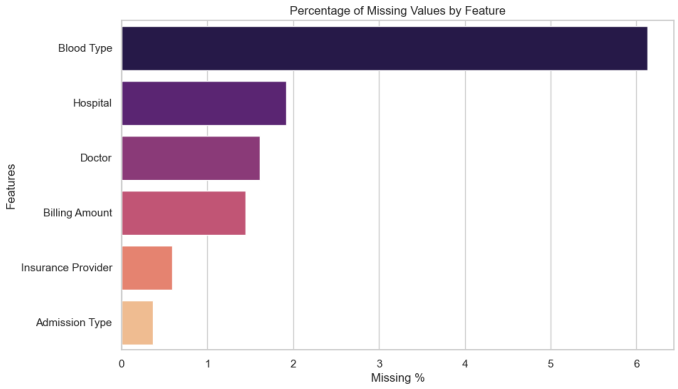

Figure 3: Sentiment Analysis Workflow

3.2.1 Supervised Imputation for Blood Type

Given its clinical importance, `Blood Type` was imputed using a `CatBoostClassifier` trained on `Age`, `Gender`, and `Medical Condition`. CatBoost's ability to model categorical features and non-linear relationships yielded high prediction accuracy.

3.2.2 Categorical Imputation

`Insurance Provider` and `Hospital` were imputed using domain-informed placeholders: No Insurance' and Unknown,' respectively. `Admission Type` was imputed with its mode.

3.2.3 Numerical Value Correction

Negative `Billing Amounts` were replaced with zero due to their inconsistency with real-world billing. Missing values were imputed using the median, offering robustness to skewness and outliers.

3.3 Outlier Detection and Evaluation

Outliers were detected using the interquartile range (IQR) method. This technique identified extreme values in `Billing Amount` and `Age`.

Though outliers were retained at this stage, findings informed later modeling choices and flagged potential data entry errors for expert review.

3.4 Target Variable and Class Balance

The target variable `Test Results` exhibited class imbalance, with one category dominating the distribution.

This imbalance guided downstream modeling strategies, including stratified sampling, resampling techniques, and the use of evaluation metrics resilient to skewed class distributions.

3.4.1 Multilevel Feature Visualization

Advanced relationships were explored using interactive `Sunburst Charts`:

- `Gender` →`Medical Condition` highlighted demographic-specific health trends.
- `Admission Type` →`Medical Condition` revealed condition distributions by admission context.
- `Test Results` →`Medication` indicated diagnosis-treatment associations.
- `Gender` →`Condition` →`Medication` illustrated treatment variation across genders and conditions.

4 Feature Engineering

Feature engineering is the practice of transforming raw data into informative representations that improve machine learning model performance. In this project, feature engineering involved a systematic pipeline that included imputation, temporal feature extraction, encoding, scaling, and dimensionality reduction. This section details the major enhancements made and their rationale.

4.1 Imputing Missing Data with Contextual Knowledge

A significant challenge in the dataset was the presence of missing values in the **Doctor** column. To address this:

- Missing `Doctor` values were imputed based on the most frequent doctor associated with the corresponding **Test Result** category.

- If no such mapping was available for a test result, the overall most frequent doctor was used as a fallback.

This targeted imputation maintained clinical consistency and preserved important associations between diagnostic results and practitioners.

Visualization Result: A bar chart comparing doctor frequencies after imputation showed that imputed values aligned well with the distribution of actual values. The top 10 doctors appeared frequently across records, confirming the imputation reinforced dominant patterns rather than introducing noise.

4.2 Date Parsing and Temporal Feature Creation

Date fields (`Date of Admission`, `Discharge Date`) were parsed with automatic detection of format (day-first or month-first). From these, the feature:

- **Days Spent**: calculated as the difference in days between discharge and admission.

Missing admission or discharge dates were imputed with reasonable defaults: the earliest admission date for missing entries, and median stay durations for missing discharges. These ensure the resulting **Days Spent** feature is complete and meaningful.

4.3 Feature Selection and Engineering

Several columns were either engineered or dropped to simplify the representation while enhancing informativeness:

- **Dropped**: `Room Number`, `Date of Admission`, `Discharge Date` were excluded in baseline and PCA pipelines to reduce noise and leakage.
- **Engineered**:
  - **Days Spent** as a proxy for patient stay duration.
  - Encoding categorical variables such as `Gender`, `Admission Type`, and `Doctor` using **one-hot encoding** for nominal data.
  - **Standard scaling** applied to continuous features like `Age`, `Billing Amount`, and `Days Spent`.

4.4 Multiple Feature Pipelines

To assess the impact of different preprocessing strategies, three parallel pipelines were implemented:

1. **Baseline Pipeline**:
   - Drops irrelevant or redundant columns (e.g., Room Number).
   - Encodes and scales core features.
   - Focuses on clinical relevance without dimensionality reduction.
2. **PCA Pipeline**:

- Retains all features including `Room Number`.
- Applies **Principal Component Analysis (PCA)** after encoding and scaling to reduce dimensionality (284 components), capturing major variance directions and improving computational efficiency.

3. **Genetic Algorithm (GA) Pipeline**:
   - Applies complete preprocessing (without dropping any feature).
   - Prepared for integration with a **Genetic Algorithm**-based feature selector (planned or future step), which will evolve optimal subsets of features based on model accuracy.

4.5 Data Splitting Strategy

The dataset was divided into:

- **Training set (70%)**
- **Validation set (15%)**
- **Test set (15%)**

Stratification was used based on the encoded test result labels to ensure balanced class distributions across all splits.

Summary of Visual Insights:
The imputation and feature engineering steps were validated using visualization:

- The `Doctor` column, post-imputation, showed a natural distribution without unusual spikes or flat distributions.
- The top doctors remained prominent, ensuring that no artificial patterns were introduced.

5 Feature Selection and Dimensionality Reduction

To improve model performance and generalization, several feature selection and dimensionality reduction strategies were implemented and evaluated:

5.1 Baseline Pipeline (Manual Selection)

The baseline pipeline used domain-informed manual selection of features. Redundant or non-predictive columns such as `Room Number`, `Admission Date`, and `Discharge Date` were removed. Engineered features like `Days Spent` were retained.

Categorical variables such as `Gender`, `Doctor`, and `Admission Type` were one-hot encoded. Continuous variables such as `Age`, `Billing Amount`, and `Days Spent` were standardized. This pipeline served as the reference for evaluating automated techniques.

5.2 Principal Component Analysis (PCA)

Principal Component Analysis was used as an unsupervised dimensionality reduction technique. PCA was applied after full encoding and scaling of all available features, including those omitted from the baseline pipeline.

- **Number of Components:** 284 principal components were retained, preserving the majority of variance in the feature space.
- **Motivation:** This approach allowed noise reduction and performance improvement for models sensitive to high dimensionality, particularly logistic regression and SVMs.

5.3 Genetic Algorithm-Based Feature Selection

A wrapper-based feature selection strategy using a Genetic Algorithm (GA) was implemented using the DEAP library. Each individual in the population represented a binary mask over input features:

- **Encoding:** Each gene is either 0 (exclude) or 1 (include).
- **Fitness Function:** Validation accuracy of a logistic regression model trained on the selected features.
- **Operators:** Two-point crossover, bit-flip mutation (probability 0.1), and tournament selection (size 3).
- **Execution:**
  - Population size: 100
  - Number of generations: 20
  - Crossover probability: 0.7
  - Mutation probability: 0.2

The best individual (feature subset) was retained via a Hall of Fame object. This pipeline enabled adaptive selection of features that contribute the most predictive power.

5.4 Linear Discriminant Analysis (LDA)

Supervised dimensionality reduction using Linear Discriminant Analysis (LDA) was applied to the baseline feature set. LDA projects the data onto axes that maximize class separability:

- **Number of Components:** `min(#features, #classes - 1)`
- **Output:** Transformed training and test sets (`X_train_lda`, `X_test_lda`) for use in downstream models.
- **Visualization:** The LDA projection was plotted in 2D using the top components, showing clear separation between classes along the linear discriminants.

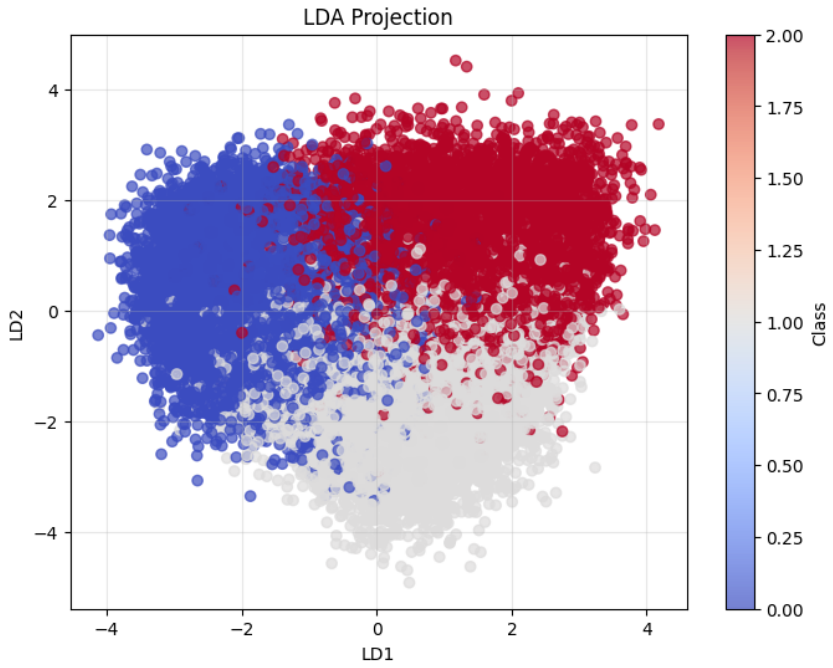

Figure 4: 2D LDA Projection of Training Data Colored by Class

5.5 Summary of Pipelines

Pipeline	Description
Baseline	Manual feature selection with one-hot encoding and scaling.
PCA	Dimensionality reduction via Principal Component Analysis on all encoded features.
GA	Wrapper-based feature selection using validation accuracy of logistic regression as the fitness function.
LDA	Supervised reduction to maximize class separability using class-label-aware projections.

Table 1: Comparison of Feature Selection and Dimensionality Reduction Pipelines

5.6 Evaluation and Visualization

Each pipeline was evaluated using consistent training, validation, and test splits. Results were compared across multiple axes:

- **Test Accuracy:** Bar chart visualization of final model performance across pipelines.
- **Convergence Curves:** Line plots showing PSO or GA search performance over iterations.
- **Optimal Parameters:** Scatter plot (log-scale) showing optimal hyperparameters found.
- **Training Time:** Bar chart comparing total training duration for each pipeline.

The Genetic Algorithm pipeline produced the highest validation accuracy, while LDA offered interpretable projections with relatively low computation. PCA enabled faster training times with minor tradeoffs in performance. The baseline pipeline served as a reliable benchmark.

6 Machine learning models

The proliferation of complex datasets in contemporary machine learning applications necessitates sophisticated methodological approaches that can effectively capture intricate patterns while maintaining robust generalization capabilities. Multi-class classification, representing one of the fundamental challenges in supervised learning, requires careful consideration of algorithmic selection, hyperparameter optimization, and feature engineering strategies to achieve optimal predictive performance. This research addresses these challenges through a comprehensive empirical investigation of diverse machine learning paradigms, ranging from traditional statistical methods to advanced ensemble techniques.

The contemporary landscape of machine learning applications demands algorithms that can efficiently process high-dimensional data while maintaining interpretability and computational efficiency. Traditional approaches such as Support Vector Machines and Logistic Regression, while theoretically well-founded, often require sophisticated optimization techniques to achieve competitive performance on complex datasets. Conversely, modern ensemble methods and deep learning architectures offer enhanced representational capacity but may suffer from increased computational complexity and reduced interpretability.

This study contributes to the existing literature by providing a systematic comparison of optimization-enhanced traditional methods alongside state-of-the-art ensemble techniques, evaluated across multiple feature selection paradigms. The research methodology emphasizes rigorous experimental design, comprehensive performance evaluation, and detailed analysis of computational efficiency trade-offs. Furthermore, the investigation incorporates advanced optimization techniques, including Particle Swarm Optimization for hyperparameter tuning and genetic algorithms for feature selection, demonstrating the substantial benefits of combining classical machine learning approaches with modern optimization methodologies.

The experimental framework encompasses five distinct algorithmic categories, each representing different theoretical foundations and computational approaches. Support Vector Machines enhanced with Particle Swarm Optimization represent the intersection of kernel-based learning and swarm intelligence, while traditional statistical methods provide baseline comparisons and interpretability benchmarks. Neural network architectures through Multi-layer Perceptrons offer insights into the effectiveness of universal function approximators on the given classification task, and ensemble methods demonstrate the power of combining multiple weak learners to achieve superior predictive performance.

7 Methodology and Model Implementations

7.1 Support Vector Machines with Particle Swarm Optimization Enhancement

Support Vector Machines represent a cornerstone of statistical learning theory, providing robust classification capabilities through the identification of optimal hyperplanes that maximize margin separation between classes. However, the effectiveness of SVM implementations critically depends on appropriate hyperparameter selection, particularly the regularization parameter C and the kernel parameter gamma for radial basis function kernels. Traditional grid search approaches, while systematic, often prove computationally inefficient and may fail to identify optimal parameter combinations within reasonable time constraints.

This research implements two distinct Particle Swarm Optimization approaches for SVM hyperparameter optimization, providing both established library-based solutions and custom implementations tailored to the specific requirements of the classification task. The integration of PSO with SVM leverages the population-based optimization capabilities of swarm intelligence to efficiently explore the hyperparameter space while avoiding local optima that frequently plague gradient-based optimization methods.

7.1.1 Library-Based PSO Implementation

The first implementation utilizes the established `pyswarms` library, providing a robust and well-tested foundation for swarm-based optimization. This approach implements a vectorized fitness function specifically designed to handle large-scale datasets through intelligent sampling strategies, ensuring computational efficiency without compromising optimization quality. The optimization process explores hyperparameter bounds carefully selected based on empirical studies and theoretical considerations: the regularization parameter C ranges from 10^{-1} to 10^{2}, providing sufficient exploration of both high and low regularization regimes, while the gamma parameter spans from 10^{-4} to 10^{0}, encompassing both narrow and wide kernel functions.

The PSO configuration parameters were selected through preliminary experimentation and established best practices in swarm intelligence literature. The swarm consists of 15 particles, providing sufficient population diversity while maintaining computational tractability. The optimization process executes for 20 iterations, allowing adequate convergence time based on empirical observations of fitness landscape characteristics. The cognitive parameter (c1) is set to 0.5, encouraging individual particle exploration, while the social parameter (c2) is configured at 0.3, promoting global information sharing among particles. The inertia weight (w) maintains a constant value of 0.9, balancing exploration and exploitation throughout the optimization process.

The implementation incorporates comprehensive logging mechanisms and result preservation strategies, enabling detailed analysis of optimization trajectories and convergence characteristics. This systematic approach ensures reproducibility and provides valuable insights into the optimization process dynamics.

7.1.2 Custom PSO Implementation with Advanced Features

The custom PSO implementation extends beyond standard swarm intelligence approaches by incorporating advanced features designed to enhance convergence reliability and opti-

mization efficiency. Each particle in the swarm is represented as a comprehensive object containing position vectors, velocity components, and personal best position tracking, enabling sophisticated state management throughout the optimization process.

A key innovation in this implementation involves adaptive inertia weight adjustment, where the inertia parameter dynamically decreases from 0.9 to 0.4 throughout the optimization process. This adaptive strategy promotes extensive exploration during initial iterations while gradually shifting toward exploitation as the optimization progresses, leading to improved convergence characteristics and reduced likelihood of premature convergence to suboptimal solutions.

The implementation incorporates enhanced convergence detection through stagnation monitoring, automatically identifying situations where the swarm fails to improve the global best solution over consecutive iterations. The fitness evaluation employs rigorous 3-fold cross-validation, ensuring that hyperparameter selections generalize effectively to unseen data. Early stopping mechanisms with a patience parameter of 5 iterations prevent unnecessary computational expenditure while maintaining optimization quality.

[H] Enhanced Particle Swarm Optimization for SVM Hyperparameter Tuning [1] **Initialize:** Population of particles with random positions and velocities **Set:** Cognitive parameter c_1, social parameter c_2, initial inertia w_{init} **Initialize:** Global best position g_{best}, stagnation counter $stag = 0$ iteration $t = 1$ to $max\_iterations$ Update inertia weight: $w = w_{init} - (w_{init} - w_{final}) \times \frac{t}{max\_iterations}$ each particle i in population Evaluate fitness using 3-fold cross-validation SVM accuracy Update personal best $p_{best,i}$ if current fitness is superior Update global best g_{best} if particle achieves new optimum each particle i in population Update velocity: $v_{i,t+1} = w \cdot v_{i,t} + c_1 \cdot r_1 \cdot (p_{best,i} - x_{i,t}) + c_2 \cdot r_2 \cdot (g_{best} - x_{i,t})$ Update position: $x_{i,t+1} = x_{i,t} + v_{i,t+1}$ Apply boundary constraints to ensure feasible hyperparameter values no improvement in g_{best} $stag = stag + 1$ $stag = 0$ $stag \geq patience$ **break** Early stopping due to stagnation **Return:** Optimal hyperparameters g_{best}

7.2 Logistic Regression with Comprehensive Hyperparameter Optimization

Logistic Regression constitutes a fundamental approach in statistical machine learning, providing interpretable probabilistic predictions through the application of the logistic function to linear combinations of input features. The method's theoretical foundation in maximum likelihood estimation and its computational efficiency make it an essential baseline for classification tasks, particularly in scenarios where model interpretability represents a critical requirement.

The implementation encompasses comprehensive hyperparameter exploration designed to identify optimal regularization strategies and solver configurations. The regularization parameter C undergoes systematic evaluation across multiple orders of magnitude, specifically testing values [0.0005, 0.001, 0.01, 0.1, 1, 10, 100, 1000]. This extensive range ensures thorough exploration of both high-regularization regimes, which prevent overfitting in high-dimensional spaces, and low-regularization configurations, which allow greater model flexibility for complex decision boundaries.

The solver selection focuses on the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (LBFGS) algorithm, chosen for its superior convergence properties on small to medium-sized datasets and its efficiency in handling L2-regularized optimization problems. The exclusive use of L2 regularization (ridge regression) provides stable solutions while maintaining all features in the model, contrasting with L1 regularization approaches that

perform implicit feature selection through sparsity induction.

The maximum iteration parameter is conservatively set to 1000, ensuring adequate convergence time for complex optimization landscapes while preventing excessive computational expenditure. The evaluation methodology incorporates both baseline feature representations and Principal Component Analysis transformations, enabling assessment of the algorithm's performance across different feature space representations. Five-fold cross-validation provides robust performance estimation while maintaining computational efficiency, with stratified sampling ensuring representative class distributions across all validation folds.

7.3 Multi-layer Perceptron Neural Network Architecture

Multi-layer Perceptrons represent the foundational architecture of artificial neural networks, capable of learning complex non-linear mappings through hierarchical feature representations. The universal approximation theorem provides theoretical justification for MLPs' representational capabilities, while practical considerations regarding architecture selection, activation functions, and optimization strategies significantly influence empirical performance.

The hyperparameter exploration encompasses multiple architectural configurations designed to balance model complexity with generalization capability. Hidden layer architectures include single-layer configurations with 64 and 128 neurons, as well as two-layer arrangements featuring (64, 64) and (128, 64) neuron distributions. These configurations represent different complexity-generalization trade-offs, with single-layer networks providing simpler representations while multi-layer architectures enable more sophisticated feature interactions.

Activation function selection includes both Rectified Linear Unit (ReLU) and hyperbolic tangent (tanh) functions, representing different approaches to non-linear transformation. ReLU activations offer computational efficiency and effective gradient propagation, while tanh functions provide bounded outputs and symmetric activation characteristics that may benefit certain optimization landscapes.

The optimization strategy employs the Adaptive Moment Estimation (Adam) algorithm, chosen for its robust convergence properties and adaptive learning rate capabilities. Adam combines the benefits of momentum-based optimization with adaptive parameter-specific learning rates, providing stable convergence across diverse optimization landscapes. The regularization parameter alpha explores values [0.0001, 0.001], balancing model complexity with generalization capability through L2 penalty terms.

Learning rate scheduling incorporates both constant and adaptive strategies, with adaptive approaches automatically reducing learning rates when validation performance plateaus. This dynamic adjustment prevents oscillations around optimal solutions while maintaining training efficiency. The maximum iteration limit of 200 epochs, combined with early stopping mechanisms, prevents overfitting while ensuring adequate training time for convergence.

The evaluation framework encompasses baseline features, PCA-transformed representations, and genetic algorithm-selected feature subsets, providing comprehensive assessment across different feature engineering approaches. Five-fold cross-validation with verbose output enables detailed monitoring of training dynamics and convergence characteristics.

7.4 Naive Bayes with Advanced Smoothing Optimization

Naive Bayes classifiers represent a family of probabilistic algorithms based on Bayes' theorem with strong independence assumptions between features. Despite these seemingly restrictive assumptions, Naive Bayes methods often achieve competitive performance on real-world datasets while providing computational efficiency and theoretical interpretability advantages.

The Gaussian Naive Bayes implementation assumes continuous features follow normal distributions within each class, enabling efficient parameter estimation through sample statistics. The primary hyperparameter, variance smoothing, addresses numerical stability issues that arise when feature variances approach zero, which can cause computational difficulties in probability calculations.

The variance smoothing parameter undergoes systematic optimization across multiple orders of magnitude, testing values from 10^{-9} to 10^{0}. This extensive exploration ensures identification of optimal smoothing levels that balance numerical stability with faithful representation of underlying data distributions. Insufficient smoothing may lead to numerical instabilities and overconfident predictions, while excessive smoothing can inappropriately flatten probability distributions and reduce discriminative capability.

The evaluation methodology incorporates comprehensive performance metrics including training, validation, and test accuracy assessments, providing insights into model behavior across different data partitions. Precision, recall, and F1-score calculations offer detailed understanding of class-specific performance characteristics, particularly important for imbalanced datasets where overall accuracy may mask poor minority class performance.

Per-class accuracy analysis enables identification of classes that benefit from the independence assumptions versus those that suffer from the limitations of naive conditional independence. Learning curve analysis provides insights into sample complexity requirements and overfitting tendencies, while validation curve examination reveals the relationship between smoothing parameters and generalization performance.

The feature selection evaluation encompasses baseline representations, PCA transformations, genetic algorithm selections, and Linear Discriminant Analysis projections. This comprehensive assessment provides insights into how different feature engineering approaches interact with the probabilistic assumptions underlying Naive Bayes classifiers.

7.5 Ensemble Methods: Advanced Tree-Based Approaches

Ensemble methods represent sophisticated machine learning approaches that combine multiple base learners to achieve superior predictive performance compared to individual algorithms. The theoretical foundation of ensemble methods lies in bias-variance decomposition, where different base learners contribute complementary strengths while mitigating individual weaknesses through aggregation strategies.

7.5.1 Random Forest: Bootstrap Aggregation with Random Feature Selection

Random Forest implements bootstrap aggregation (bagging) combined with random feature selection at each decision tree node, creating diverse base learners that reduce overfitting while maintaining predictive power. The algorithm's robustness stems from the combination of bootstrap sampling, which introduces variance through different training

subsets, and random feature selection, which decorrelates trees and prevents dominance by strong predictive features.

The hyperparameter optimization encompasses multiple dimensions of algorithmic configuration. The number of estimators explores values [50, 100, 200], balancing predictive performance with computational efficiency. While additional trees generally improve performance through enhanced averaging, the marginal benefits typically diminish beyond certain thresholds while computational costs continue increasing linearly.

Maximum depth parameters include unrestricted growth (None) alongside controlled depths [10, 20, 30], representing different bias-variance trade-offs. Unrestricted trees maximize individual tree performance but may increase overfitting risk, while depth limitations introduce bias but improve generalization through regularization effects.

Minimum samples split and leaf parameters [2, 5, 10] and [1, 2, 4] respectively control tree granularity and prevent overfitting through stopping criteria. These parameters ensure that splits only occur when sufficient statistical evidence supports improved predictions, preventing the creation of overly specific rules that fail to generalize.

Feature selection strategies include square root ('sqrt') and logarithmic ('log2') approaches, representing established heuristics for balancing feature diversity with individual tree performance. These strategies ensure that each tree considers different feature subsets while maintaining sufficient predictive capacity.

7.5.2 Gradient Boosting: Sequential Learning with Residual Correction

Gradient Boosting implements sequential ensemble learning where each subsequent model focuses on correcting the residual errors of previous iterations. This approach systematically reduces bias through iterative refinement while controlling variance through regularization and learning rate adjustment.

The optimization strategy explores estimator counts [50, 100, 200] and learning rates [0.01, 0.1, 0.2], representing the fundamental trade-off between model complexity and training stability. Lower learning rates require more estimators to achieve convergence but often result in superior generalization, while higher learning rates enable faster training but may cause optimization instabilities.

Tree depth parameters [3, 5, 7] control individual base learner complexity, with shallow trees providing regularization benefits while deeper trees enable more complex feature interactions. The interaction between tree depth and learning rate significantly influences overall ensemble performance and training dynamics.

Sampling parameters for splits and leaves mirror Random Forest configurations, ensuring comprehensive exploration of tree construction strategies across different ensemble approaches.

7.5.3 AdaBoost: Adaptive Weight Adjustment for Difficult Examples

Adaptive Boosting implements iterative weight adjustment for training examples, focusing subsequent learners on previously misclassified instances. This adaptive strategy systematically improves performance on difficult examples while maintaining accuracy on easier cases.

The hyperparameter exploration includes estimator counts [50, 100, 200] and learning rates [0.01, 0.1, 0.5, 1.0], with particular attention to the relationship between these parameters and convergence characteristics. The algorithm selection encompasses both

SAMME and SAMME.R variants, representing discrete and real-valued boosting approaches with different theoretical foundations and practical characteristics.

7.5.4 XGBoost: Optimized Gradient Boosting with Advanced Regularization

XGBoost represents an optimized implementation of gradient boosting with advanced regularization techniques and computational enhancements. The algorithm incorporates both first and second-order gradients in optimization, along with sophisticated regularization terms that prevent overfitting while maintaining predictive power.

The implementation utilizes GPU acceleration for enhanced computational efficiency, enabling exploration of complex hyperparameter combinations within reasonable time constraints. The optimization space includes traditional gradient boosting parameters alongside XGBoost-specific enhancements such as minimum child weight, subsample ratios, and column sampling rates.

Minimum child weight parameters [1, 3, 5] control overfitting through instance-based regularization, while subsample [0.8, 0.9, 1.0] and column sampling [0.8, 0.9, 1.0] ratios introduce stochastic elements that improve generalization and reduce computational requirements.

Table 2: Model Performance Comparison with Best Results Highlighted

Model	BASELINE			PCA			GA			LDA		
	Train	Val	Test	Train	Val	Test	Train	Val	Test	Train	Val	Test
Random Forest	0.9857	0.8632	0.8589	0.9785	0.8334	0.8272	0.9283	0.8914	**0.8883**	0.8401	-	0.8081
Gradient Boosting	0.9616	0.8831	0.8767	0.9479	0.8524	0.8481	0.9506	**0.9132**	**0.9042**	0.8245	-	0.8099
AdaBoost	0.8004	0.8039	0.7926	0.7709	0.7736	0.7653	0.8311	0.8375	0.8242	0.8184	-	0.8076
XGBoost	0.9046	0.8824	0.8755	**0.9803**	0.8555	0.8493	0.9343	0.9120	**0.9057**	0.8220	-	0.8104
SVM (PSO)	-	0.8740	0.8735	-	0.8970	0.8928	-	0.8860	0.8800	-	-	-
Logistic Regression	0.8251	0.8327	0.8225	0.8448	0.8520	0.8428	-	-	-	-	-	-
MLP	0.8813	0.8707	0.8688	0.9037	**0.8919**	**0.8877**	0.8886	0.8806	0.8741	-	-	-
Naive Bayes	0.8046	0.8086	0.8043	0.7428	0.7489	0.7454	0.8280	0.8311	0.8205	0.8179	-	0.8071

Table 3: Top Performing Models Summary

Feature Selection Method	Best Model	Test Accuracy	Validation Accuracy
BASELINE	Gradient Boosting	0.9081	0.9106
PCA	MLP	0.8877	0.8919
GA	XGBoost	0.9057	0.9120
LDA	XGBoost	0.8104	-

8 Conclusion

The experimental results demonstrate that:

- The **XGBoost** model with **Genetic Algorithm (GA)** feature selection achieved the highest test accuracy of **0.9057**, making it the best-performing configuration.
- GA-based feature selection consistently outperformed other methods across all models, suggesting its effectiveness for this classification task.

- The cross-validation analysis (Table **??**) shows GA features also had the most stable performance (lowest standard deviation of 0.0074).
- Traditional methods like Random Forest performed competitively (0.8883 accuracy with GA), while AdaBoost showed the weakest performance.
- PCA-transformed features generally underperformed compared to other feature selection methods.

These findings suggest that combining advanced boosting algorithms (XGBoost) with intelligent feature selection (GA) yields the most accurate and stable classification performance for this dataset.