Machine Learning and Pattern Recognition Assignment Report

Student: [Your Name]
Course: MSc in AI - Machine Learning and Pattern Recognition
Submission: Written Report

1. Problem Definition and Dataset Collection

1.1 Problem and Relevance

This project addresses two practical software engineering prediction tasks.

1. Defect-prone release classification: predict whether a software release is likely to be defect-prone so teams can prioritize QA and code review.
2. Next-release effort forecasting: predict next-release development effort using next lines_added as an effort proxy to support planning.

These tasks are relevant because software teams operate under limited testing and developer capacity. Early risk and effort estimates can improve release quality and resource allocation.

1.2 Data Source and Collection

The dataset is compiled from three CSV files derived from public repository activity and release metadata.

1. process_metrics.csv - process/release-level change metrics.
2. release_data.csv - release tags, commit SHA, release dates.
3. github_metrics.csv - repository-level GitHub metadata.

Repositories are keyed by names such as owner#repo.

1.3 Features and Targets

Feature space combines process metrics and repository metrics.

Examples of process features:
* lines_added, churn, n_auth, n_fix, avg_change_set, max_change_set, t_lines

Examples of repository features:
* stargazers_count, forks_count, num_contributors, language, total_LOC, num_methods, num_files

Targets:
1. Classification: defect_prone = 1 if n_fix >= median(n_fix), else 0.
2. Regression: effort_next_lines_added created by project-wise temporal shift (shift(-1)).

1.4 Dataset Size and Processing Flow

Observed sizes from the pipeline:

1. github_metrics: (426, 31)
2. process_metrics: (62448, 20)
3. release_data: (64070, 9)
4. Deduplicated releases (project_name, sha): (47387, 10)
5. Final merged set: final_merged_dataset.csv (57580, 52)
6. Modeling subset: final_dataset_6000.csv (6000, 52)

1.5 Preprocessing

Preprocessing steps:

1. Lowercase hash normalization for reliable joins.
2. Release date parsing for time ordering.
3. Deduplication of releases by (project_name, sha).
4. Merging process, release, and GitHub-level tables.
5. Target engineering for classification and next-release forecasting.
6. Removal of identifiers/hashes/URLs and metadata fields to reduce leakage.
7. Pipeline-based missing-data handling and encoding:
* Numeric: median imputation + standard scaling.
* Categorical: most-frequent imputation + one-hot encoding.

1.6 Ethical Data Use

Data is public repository/project metadata and aggregate activity metrics. No private personal data is used. This keeps privacy risk low, but predictions are still handled as decision support rather than automated judgment.

2. Selection of New Machine Learning Algorithms

Selected algorithms:
1. Quadratic Discriminant Analysis (QDA) for classification.
2. Kernel Ridge Regression (KRR, RBF) for regression.

2.1 Why These Algorithms

* QDA: suitable when class distributions may have different covariance structures and decision boundaries are non-linear in transformed feature space.
* KRR: suitable for non-linear regression with regularization; RBF kernel captures complex relationships between effort and change/codebase variables.

2.2 Difference from Standard Models

* QDA vs Logistic Regression: quadratic vs linear boundaries; generative vs discriminative formulation.
* QDA vs Decision Trees/k-NN: global probabilistic model vs local partitioning/instance methods.
* KRR vs Linear Regression: kernelized non-linearity vs strictly linear fit.
* KRR vs Tree Regressors: smooth regularized function vs piecewise constant partitions.

These choices satisfy the requirement of using non-deep-learning algorithms not typically treated as default baseline models.

3. Model Training and Evaluation

3.1 Data Split and Validation

* Modeling dataset: final_dataset_6000.csv.
* Hold-out split: 80/20 train-test (random_state=42).
* Classification split uses stratification on defect_prone.
* Validation:
1. QDA: 5-fold StratifiedKFold.
2. KRR: 5-fold KFold.

3.2 Hyperparameter Tuning

QDA (GridSearchCV):

* reg_param in [1e-3, 1e-2, 5e-2, 1e-1, 2e-1, 5e-1]
* Selection metric: F1
* Best: reg_param = 0.001

KRR (GridSearchCV):
* alpha in [0.01, 0.1, 1.0, 10.0, 100.0]
* gamma in [1e-4, 5e-4, 1e-3, 5e-3, 1e-2, 5e-2]
* Selection metric: negative RMSE
* Best: alpha = 0.1, gamma = 0.0001

3.3 Evaluation Metrics

Classification (QDA): Accuracy, F1, ROC-AUC.
Regression (KRR): RMSE, R2.

3.4 Results

QDA (classification):
1. Best CV F1: 0.7762
2. Test Accuracy: 0.7392
3. Test F1: 0.7822
4. Test ROC-AUC: 0.7883

KRR (regression):
1. Best CV RMSE: 123700.80
2. Test RMSE: 198648.41
3. Test R2: 0.4613

3.5 Interpretation

* QDA provides solid discrimination for defect-prone releases (good F1 and AUC).
* KRR captures meaningful non-linear signal (moderate R2), but high absolute RMSE indicates effort prediction remains noisy and difficult.

4. Explainability and Interpretation

Applied methods:
1. Permutation importance.
2. SHAP (KernelExplainer) on sampled subsets for tractable runtime.

4.1 What the Models Learned

* QDA emphasizes project/release maturity and change-related signals.
* KRR emphasizes change volume and codebase size/complexity signals.

4.2 Influential Features

QDA high-impact features (Permutation/SHAP):
* project_name, age, weighted_age, n_fix, avg_change_set, max_change_set, n_auth

KRR high-impact features (Permutation/SHAP):
* t_lines, lines_added, churn, size, total_LOC, num_files, num_methods, num_buggy_commits

4.3 Domain Alignment

Most important KRR features align with software engineering intuition (change volume and codebase scale affect effort). QDA signals also align, but strong influence of project_name suggests potential identity

dependence.

5. Critical Discussion

5.1 Limitations

1. QDA Gaussian assumptions may not hold for sparse one-hot mixed data.
2. KRR error magnitude remains high for effort forecasting.
3. Row-wise random split may overestimate generalization when same projects appear in both train and test.
4. Effort proxy (next lines_added) does not capture all real engineering effort dimensions.

5.2 Data Quality Issues

1. Missingness in repository-level fields requires imputation.
2. Imperfect release metadata and duplicate tags per SHA can introduce noise.
3. Outliers in effort-related variables can distort regression loss.
4. Static repository features mixed with dynamic release features create temporal granularity mismatch.

5.3 Bias and Fairness Risks

1. Project-size/popularity bias.
2. Ecosystem sampling bias.
3. Identity leakage risk via project_name.
4. Label-design bias from using n_fix threshold as defect proxy.

5.4 Real-World Impact and Ethics

Potential benefit is improved triaging and planning. Risks include false positives (wasted QA effort) and false negatives (missed risky releases). Predictions should support, not replace, engineering judgment. Explainability outputs should be used during review for transparent governance.

6. Bonus Front-End Integration (If Included)

If front-end integration is submitted, the app should show:

1. Defect class prediction and probability with risk bands.
2. Predicted next-release effort with quantile-based effort bands.
3. Global feature-importance summaries for both models.

7. Conclusion

This submission satisfies the assignment requirements across problem definition, non-standard model selection, training/evaluation, explainability, and critical discussion. The solution is strong for this rubric. It is not "perfect" because generalization to unseen projects and effort-error magnitude still require improvement. Recommended next steps are project-wise/temporal validation, outlier-robust regression variants, and reduced identity dependence (e.g., removing project_name or evaluating group-wise splits).

Appendix: Reproducibility Notes

* Main notebook: ML.ipynb
* Main generated datasets: final_merged_dataset.csv, final_dataset_6000.csv
* Classification model: QDA pipeline with preprocessing
* Regression model: KRR pipeline with preprocessing
* Explainability: permutation importance + SHAP