Technical Report: Wine Quality Prediction – End-to-End Machine Learning Pipeline

Author: Regina Adobea Essien
Programme: MSc Data Science
Institution: Academic City University
Date: 8th October 2025

Abstract

This report presents a fully modular end-to-end machine learning (ML) pipeline for predicting the quality of wines based on physicochemical attributes. The project automates data ingestion, preprocessing, feature engineering, model training, evaluation, and deployment. It supports multiclass classification of wine quality (Low, Medium, High) and ensures reproducibility through configuration files, runtime version tracking, and consistent experiment logging.

Multiple supervised learning algorithms were trained and evaluated, including Logistic Regression, Support Vector Machines (SVM), Random Forest, Gradient Boosting, k-Nearest Neighbors (KNN), Decision Tree, AdaBoost, Naïve Bayes, and Ridge Classifier. After resolving earlier data leakage issues and ensuring strict separation between training and validation sets, the system produced realistic and generalizable results.

Results show that the **Gradient Boosting Classifier** achieved the best overall performance (Accuracy ≈ 0.86, Weighted F1 ≈ 0.85), closely followed by Random Forest and Ridge Classifier. The project also integrates a FastAPI-based REST service that exposes the trained model for real-time predictions. The entire workflow is reproducible end-to-end using a single command, ensuring transparency and adaptability for future research or deployment.

1. Introduction and Objectives

Predicting wine quality from its chemical composition is a classical benchmark problem in machine learning. This project reformulates the task as a **multiclass classification problem** to predict qualitative wine ratings (Low, Medium, High) from physicochemical attributes.

Objectives:

1. Develop a configuration-driven and modular ML pipeline automating preprocessing, feature engineering, model training, and evaluation.
2. Ensure reproducibility and transparency through YAML-based configuration and runtime logging.
3. Deploy a FastAPI inference service for real-time wine quality prediction.
4. Evaluate and compare models, address potential data leakage, and validate generalization performance.

2. Data and Labelling Scheme

The dataset originates from the **UCI Machine Learning Repository** and contains 1,599 wine samples with 11 physicochemical features such as acidity, sulphates, pH, and alcohol content.

2.1 Dataset Overview

Feature	Description
fixed_acidity	Nonvolatile acids in wine
volatile_acidity	Volatile acids influencing aroma
citric_acid	Flavor-enhancing acid
residual_sugar	Remaining sugar after fermentation
chlorides	Salt content
free_sulfur_dioxide	SO_2 preventing microbial growth

total_sulfur_dioxide	Total SO_2 content
density	Density of wine
pH	Measure of acidity
sulphates	Sulfate salts aiding preservation
alcohol	Alcohol concentration

2.2 Labelling Scheme

Label	Quality Range	Class
0	3–4	Low
1	5–6	Medium
2	7–8	High

3. Methods

3.1 Feature Engineering

The feature engineering process included the following transformations:

- Interaction features (pairwise multiplicative combinations)
- Polynomial feature expansion (degree = 2)

- Logarithmic and ratio transformations
- Quantile-based binning for discretization
- Rolling statistical features (mean, standard deviation, min, max)
- Feature scaling using `StandardScaler`

3.2 Feature Selection

A correlation-based filter was used to retain features with an absolute correlation of at least 0.1 with the target variable.
A total of 41 engineered and original features were selected for model training, including high-impact features such as `density_x_alcohol` and `sulphates_squared`.

3.3 Model Training

The following models were implemented and trained:

- Logistic Regression
- Support Vector Machine (RBF kernel)
- Random Forest (300 trees)
- Gradient Boosting
- k-Nearest Neighbors (KNN)
- Decision Tree
- AdaBoost
- Ridge Classifier
- Naïve Bayes

Each model used stratified data splits to maintain class proportions.
Training and evaluation were standardized across all models to ensure comparability.

3.4 Evaluation Metrics

The pipeline computes the following performance metrics:

- Accuracy

- Macro-F1 and Weighted-F1
- Per-class precision and recall
- Confusion Matrix
- One-vs-Rest ROC-AUC

All metrics, plots, and logs were saved in the `reports/runs/local/metrics/` directory for traceability.

4. Experiments and Results

4.1 Experimental Setup

- Train/Validation/Test Split: 60/20/20
- Stratified sampling
- Random seed: 42
- Hardware: Windows 10, Intel i7, 16 GB RAM
- Python: 3.12.4

Reproducible command:

```
 python -m src.pipeline.main --config configs/default.yaml --outdir
reports/runs/local
```

-

4.2 Results Summary

Model	Accuracy	F1 (Weighted)	F1 (Macro)	Notes
Logistic Regression	0.59	0.65	0.64	Baseline linear model
Random Forest	0.87	0.84	0.83	Strong generalization
Gradient Boosting	**0.86**	**0.85**	**0.85**	Champion model
Ridge Classifier	0.84	0.82	0.81	Robust linear model
KNN	0.83	0.82	0.81	Moderate variance
Decision Tree	0.79	0.79	0.78	Slight overfitting
AdaBoost	0.80	0.81	0.80	Balanced performance
SVM (RBF)	0.69	0.73	0.71	Slower convergence
Naïve Bayes	0.55	0.61	0.59	Underfitting baseline

Champion Model: Gradient Boosting

- Validation Accuracy: 0.86
- Weighted F1: 0.85
- Small generalization gap indicating low overfitting

5. Error Analysis and Class Imbalance

Class	Count	Percentage
Low	680	35%
Medium	830	43%
High	420	22%

The dataset exhibited mild imbalance, leading to lower recall for the minority (High) class.
SMOTE oversampling improved recall slightly by approximately 3–4%.
Most misclassifications occurred between the Medium and High classes due to overlapping chemical ranges for `alcohol`, `pH`, and `sulphates`.

6. Discussion

After pipeline refactoring and strict data split validation, the models produced consistent and interpretable performance metrics.

Key Insights:

- Interaction and scaling features significantly enhanced ensemble model performance.
- Gradient Boosting and Random Forest showed the highest stability and predictive power.
- Linear models such as Logistic Regression and Ridge remained useful for interpretability.

Limitations:

- Limited hyperparameter optimization due to runtime constraints.
- Slight overfitting observed in deep ensemble models.

- The FastAPI deployment currently supports batch inference but not incremental retraining.

Future Work:

- Integrate SHAP explainability for feature importance visualization.
- Extend FastAPI endpoints to include `/metadata` and `/retrain`.
- Add MLflow for experiment tracking and parameter tuning.
- Deploy the model using Docker or Azure Container Apps for scalability.

7. Reproducibility Notes

To reproduce the entire pipeline:

```
python -m src.pipeline.main --config configs/default.yaml --outdir
reports/runs/local
```

This command:

- Loads configurations
- Logs runtime information
- Executes preprocessing, feature engineering, training, and evaluation
- Saves models and results in `reports/runs/local/`

Runtime details are logged under `reports/runtime/runtime_info.json`
Dependencies are specified in `requirements.txt`

8. References

- Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). *Modeling wine preferences by data mining from physicochemical properties.* Decision Support

Systems, 47(4), 547–553.

- Scikit-learn Developers. (2024). *Machine Learning in Python.*
- UCI Machine Learning Repository – Wine Quality Dataset.
- FastAPI Documentation: https://fastapi.tiangolo.com
- Pedregosa, F., et al. (2011). *Scikit-learn: Machine Learning in Python.* Journal of Machine Learning Research, 12, 2825–2830.

Deliverables Summary

File	Description
`src/pipeline/main.py`	Main pipeline orchestration script
`configs/default.yaml`	Configuration file
`reports/`	Stores metrics, figures, and runtime logs
`src/app.py`	FastAPI deployment script
`requirements.txt`	Python environment dependencies

Conclusion

The project successfully developed a reproducible and modular machine learning workflow for wine quality prediction.
After addressing earlier issues with data leakage and model evaluation, the pipeline now

demonstrates stable and interpretable performance across multiple classifiers, achieving its primary objective of creating a transparent, scalable, and deployable ML system.