Wine Quality Classification: A Comparative Study of SVM, Kernel Methods, and Logistic Regression

Farabi Amina
Matricola: 59486A

February 28, 2026

Declaration: I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

Contents

1 Introduction

This report presents a comprehensive machine learning project focused on classifying wine quality using various classification algorithms. The dataset combines red and white wine samples with physicochemical properties as features and quality scores as target variables. The main objectives of this project are:

- Implement and compare linear models (SVM and Logistic Regression) from scratch
- Extend SVM to handle non-linear decision boundaries using kernel methods
- Evaluate model performance using appropriate metrics and validation techniques
- Analyze the trade-offs between model complexity, computational efficiency, and predictive performance

2 Data Exploration and Preprocessing

2.1 Dataset Overview

The dataset consists of 6,497 wine samples (both red and white wines) with 11 physicochemical features and a quality score ranging from 3 to 9. Table 1 presents the statistical summary of all features.

Table 1: Statistical Summary of Wine Quality Dataset

	fixed acidity	volatile acid	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6497	6497	6497	6497	6497	6497	6497	6497	6497	6497	6497	6497
mean	7.215	0.340	0.319	5.443	0.056	30.525	115.745	0.995	3.219	0.531	10.492	5.818
std	1.296	0.165	0.145	4.758	0.035	17.749	56.522	0.003	0.161	0.149	1.193	0.873
min	3.800	0.080	0.000	0.600	0.009	1.000	6.000	0.987	2.720	0.220	8.000	3.000
25%	6.400	0.230	0.250	1.800	0.038	17.000	77.000	0.992	3.110	0.430	9.500	5.000
50%	7.000	0.290	0.310	3.000	0.047	29.000	118.000	0.995	3.210	0.510	10.300	6.000
75%	7.700	0.400	0.390	8.100	0.065	41.000	156.000	0.997	3.320	0.600	11.300	6.000
max	15.900	1.580	1.660	65.800	0.611	289.000	440.000	1.039	4.010	2.000	14.900	9.000

2.2 Class Distribution Analysis

The target variable (quality) was binarized to create a two-class classification problem:

- Quality ≥ 6: Good quality wine (+1 class)
- Quality ≤ 5: Poor quality wine (-1 class)

The resulting class distribution shows significant imbalance:

- Positive class (+1): 4,113 samples (63.3%)
- Negative class (-1): 2,384 samples (36.7%)

This imbalance necessitates careful handling during model training to avoid bias toward the majority class. We addressed this through:

- Class weights in loss functions
- Using F1-score as the primary optimization metric during hyperparameter tuning
- Stratified cross-validation to maintain class proportions

2.3 Preprocessing Pipeline

To ensure robust model development without data leakage, we implemented the following preprocessing steps:

1. **Train-Test Split**: 80% training, 20% testing (random_state=4 for reproducibility)
2. **Feature Standardization**:

$$X_{scaled} = \frac{X - \mu_{train}}{\sigma_{train}} \tag{1}$$

   Where μ_{train} and σ_{train} are computed only from training data and applied to both training and test sets.
3. **Stratified K-Fold**: For cross-validation, we implemented stratified k-fold to preserve class distribution across folds.

3 Methodology

3.1 Linear Support Vector Machine

3.1.1 Mathematical Formulation

The linear soft-margin SVM solves the following optimization problem:

$$\min_{w,b} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \max(0, 1 - y_i(w^T x_i + b)) \tag{2}$$

where:

- w is the weight vector
- b is the bias term
- C is the regularization parameter
- The term $\max(0, 1 - y_i(w^T x_i + b))$ is the hinge loss

3.1.2 Training Algorithm: Stochastic Gradient Descent

Algorithm 1 presents the pseudocode for training the linear SVM.

Algorithm 1 Linear SVM Training with Stochastic Gradient Descent

Require: Training data $X \in \mathbb{R}^{n \times d}$, labels $y \in \{-1, 1\}^n$, learning rate η, regularization parameter λ, number of epochs T
Ensure: Weight vector w, bias b
Initialize $w \leftarrow \mathbf{0} \in \mathbb{R}^d$, $b \leftarrow 0$
for epoch $= 1$ to T **do**
  for each training example (x_i, y_i) **do**
    if $y_i(w^T x_i + b) \geq 1$ **then**
      $w \leftarrow w - \eta(2\lambda w)$ ▷ Correctly classified, outside margin
    else
      $w \leftarrow w - \eta(2\lambda w - y_i x_i)$ ▷ Misclassified or within margin
      $b \leftarrow b - \eta(-y_i)$
    end if
  end for
end for
return w, b

3.2 Kernelized SVM with RBF

3.2.1 Kernel Trick

The kernel trick allows SVM to learn non-linear decision boundaries by implicitly mapping data to a higher-dimensional feature space. The RBF (Radial Basis Function) kernel is defined as:

$$K(x_i, x_j) = \exp(-\gamma \|x_i - x_j\|^2) \tag{3}$$

The kernelized decision function becomes:

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i K(x_i, x) + b \tag{4}$$

3.2.2 Sequential Minimal Optimization (SMO)

For training the kernelized SVM, we implemented the SMO algorithm, which breaks the large QP problem into smaller subproblems. Algorithm 2 presents the pseudocode.

Algorithm 2 SMO Algorithm for Kernel SVM

Require: Training data X, labels y, kernel function K, regularization C, tolerance tol, max passes

Ensure: Lagrange multipliers α, bias b

Initialize $\alpha \leftarrow \mathbf{0}$, $b \leftarrow 0$, passes $\leftarrow 0$
while passes $<$ max_passes **do**
    num_changed $\leftarrow 0$
    for each i from 1 to n **do**
        Compute $E_i = \sum_j \alpha_j y_j K(x_j, x_i) + b - y_i$
        if ($y_i E_i < -tol$ and $\alpha_i < C$) or ($y_i E_i > tol$ and $\alpha_i > 0$) **then**
            Select $j \neq i$ using heuristic
            Compute E_j
            Store old α_i, α_j
            Compute bounds L and H based on $y_i \neq y_j$
            if $L == H$ **then**
            end if
            Compute $\eta = 2K(x_i, x_j) - K(x_i, x_i) - K(x_j, x_j)$
            if $\eta \geq 0$ **then**
            end if
            Update α_j
            Clip α_j to $[L, H]$
            if $|\alpha_j - \alpha_j^{old}| < 10^{-5}$ **then**
            end if
            Update α_i
            Update b using thresholds
            num_changed $\leftarrow$ num_changed + 1
        end if
    end for
    if num_changed $== 0$ **then**
        passes $\leftarrow$ passes + 1
    else
        passes $\leftarrow 0$
    end if
end while
return α, b

3.3 Logistic Regression

3.3.1 Mathematical Formulation

Logistic regression models the probability of class membership using the sigmoid function:

$$P(y = 1|x) = \sigma(w^T x + b) = \frac{1}{1 + e^{-(w^T x + b)}} \tag{5}$$

The loss function (binary cross-entropy) with L2 regularization:

$$L(w, b) = -\frac{1}{N} \sum_{i=1}^{N} [y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)] + \frac{\lambda}{2} \|w\|^2 \tag{6}$$

3.3.2 Training Algorithm

Algorithm 3 presents the pseudocode for training logistic regression with gradient descent.

Algorithm 3 Logistic Regression Training with Gradient Descent

Require: Training data X, labels $y \in \{0, 1\}$, learning rate η, regularization λ, epochs T, sample weights w_s

Ensure: Weights w, bias b

Initialize $w \leftarrow \mathcal{N}(0, 0.01)$, $b \leftarrow 0$

Compute sample weights: $w_{pos} = \frac{n}{2n_{pos}}$, $w_{neg} = \frac{n}{2n_{neg}}$

for epoch $= 1$ to T **do**

Compute logits: $z = Xw + b$

Compute predictions: $\hat{y} = \sigma(z)$

Compute gradients:

$$dw = \frac{1}{n} X^T(\hat{y} - y) + \lambda w$$
$$db = \frac{1}{n} \sum (\hat{y} - y)$$

Gradient clipping: if $\|dw\| > 1$, $dw \leftarrow \frac{dw}{\|dw\|}$

Update parameters:

$$w \leftarrow w - \eta \cdot dw$$
$$b \leftarrow b - \eta \cdot db$$

end for

return w, b

4 Experimental Setup

4.1 Hyperparameter Tuning

We employed random search with 5-fold cross-validation for hyperparameter optimization. The search spaces were:

- **Linear SVM and Logistic Regression**:
  - Learning rate: log-uniform [1e-4, 1e-1]
  - Lambda (regularization): log-uniform [1e-4, 1e-2]
  - Number of iterations: uniform [200, 1500]
- **RBF SVM**:
  - Gamma: log-uniform [1e-2, 1]
  - C: log-uniform [1, 100]

The primary optimization metric was F1-score to account for class imbalance.

4.2 Evaluation Metrics

We used the following metrics for comprehensive model evaluation:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{7}$$

$$\text{Precision} = \frac{TP}{TP + FP} \tag{8}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{9}$$

$$\text{F1-Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{10}$$

5 Results and Analysis

5.1 Model Performance Comparison

Table 2 presents the final test set performance for all three models after hyperparameter tuning.

Table 2: Model Performance Comparison on Test Set

Model	**Accuracy**	**Recall**	**Precision**	**F1-Score**
Linear SVM	0.7252	0.8498	0.7459	0.7945
RBF SVM	0.7560	0.7820	0.8194	0.8003
Logistic Regression	0.7175	0.8239	0.7492	0.7848

5.2 Analysis of Results

5.2.1 Linear SVM

The linear SVM achieved a balanced performance with an F1-score of 0.7945. Notably, it has the highest recall (0.8498), indicating it's best at identifying positive samples (good quality wines). This suggests the linear decision boundary captures the underlying patterns reasonably well.

5.2.2 RBF SVM

The RBF SVM achieved the highest overall accuracy (0.7550) and F1-score (0.8003). However, it's important to note:

- **Class Imbalance Issue**: Initial experiments without class balancing resulted in the model classifying everything as positive class. This was addressed by implementing sample weights in the SMO algorithm.
- **Computational Complexity**: The RBF SVM is significantly slower than linear models due to:
  - Computing the full kernel matrix ($O(n^2)$ memory)
  - SMO algorithm requiring multiple passes through data
  - Kernel computations for predictions
- **Precision-Recall Trade-off**: The RBF SVM shows higher precision but lower recall compared to linear SVM, indicating it's more conservative in positive classifications but more accurate when it does classify positively.

5.2.3 Logistic Regression

Logistic regression performed comparably to linear SVM but slightly lower across all metrics. This is expected as both are linear models, but SVM's hinge loss might be more suitable for this classification task.

5.3 Error Analysis

5.3.1 Misclassification Patterns

Analysis of misclassified examples revealed:

1. **Boundary Cases**: Wines with quality score exactly 5 or 6 are often misclassified, suggesting these are inherently difficult cases near the decision boundary.
2. **Feature Ambiguity**: Some wines with similar physicochemical properties but different quality ratings indicate either:
   - The need for additional features
   - Subjectivity in quality ratings
   - Non-linear interactions not captured by linear models

5.4 Underfitting and Overfitting Analysis

- **RBF SVM Initial Underfitting**: The initial implementation of RBF SVM exhibited severe underfitting, classifying all samples as the majority class (+1). This occurred because:
  - The model failed to learn the decision boundary due to class imbalance
  - Without class weighting, the optimization focused on minimizing error on the majority class at the expense of completely ignoring the minority class
  - The model essentially learned a trivial solution: always predict the dominant class

  This underfitting issue was successfully addressed through:
  - **Class balancing**: Implementing sample weights to give equal importance to both classes
- **RBF SVM Post-Tuning**: After addressing underfitting and proper hyperparameter tuning, the model achieved balanced performance with no signs of overfitting, as evidenced by:
  - Consistent performance across cross-validation folds
  - Reasonable gap between training and validation metrics
  - More balanced precision and recall scores
- **Linear Models**: Both linear SVM and logistic regression showed appropriate fit without underfitting or overfitting due to:
  - Strong regularization (L2 penalty) controlling model complexity
  - Simple linear architecture matching the problem's complexity
  - Consistent cross-validation performance across all folds
  - Class balancing preventing majority class bias

6 Discussion

6.1 Kernel Methods: Trade-offs and Consequences

The kernelization of SVM introduces several important considerations:

6.1.1 Advantages

- Ability to learn non-linear decision boundaries
- RBF kernel can approximate any function given appropriate parameters
- Improved accuracy on this dataset (+0.6% F1-score)

6.1.2 Disadvantages

- **Computational Cost**: Training time increases from $O(n)$ for linear SVM to $O(n^2)$ for kernel SVM
- **Memory Requirements**: Storing the full kernel matrix requires $O(n^2)$ memory (42M entries for 6,497 samples)
- **Hyperparameter Sensitivity**: Performance highly dependent on both C and gamma
- **Prediction Time**: Each prediction requires computing kernel with all support vectors

6.2 Recommendations

Based on our analysis, we recommend:

1. **For Production Use**: Linear SVM offers the best accuracy-speed trade-off. It's nearly as accurate as RBF SVM but orders of magnitude faster.
2. **For Research/Accuracy-Critical Applications**: RBF SVM when computational resources allow and when the marginal accuracy gain justifies the cost.
3. **For Interpretability**: Logistic regression, despite slightly lower performance, offers probabilistic outputs and easily interpretable coefficients.

7 Conclusion

This project successfully implemented and compared three classification algorithms for wine quality prediction:

- **Linear SVM** achieved strong performance (F1=0.7945) with efficient training and prediction, making it suitable for production deployment.
- **RBF SVM** provided the best accuracy (F1=0.8003) but at significant computational cost. Initial issues with class imbalance and overfitting were successfully addressed through proper regularization and class weighting.
- **Logistic Regression** offered comparable performance (F1=0.7848) with the added benefit of probabilistic outputs and interpretability.

The project demonstrated the importance of proper methodology: stratified cross-validation, handling class imbalance, careful hyperparameter tuning, and preventing data leakage. While kernel methods can improve performance, the marginal gain must be weighed against the substantial increase in computational complexity.

For this specific dataset, the linear models provide excellent performance-efficiency trade-offs, suggesting that the decision boundary might be approximately linear in the original feature space.

Code Availability

The complete implementation is available at:

`https://github.com/aminafarabi/wine-quality-classification`