Wine Quality Prediction using SVM and Logistic Regression

Nazar Liubas
University of Milan
Machine Learning Course A.Y. 2024/25

October 19, 2025

Contents

1 Introduction

This report presents the implementation and evaluation of machine learning algorithms for wine quality prediction. The task is binary classification: predicting whether a wine is "good" (quality ≥ 6) or "bad" (quality < 6) based on its chemical properties.

We implement two main classification algorithms from scratch:

- **Logistic Regression** using Online Gradient Descent
- **Support Vector Machine (SVM)** using the Pegasos algorithm

Additionally, we extend both algorithms using kernel methods to capture non-linear relationships in the data. We use two kernel functions: Gaussian and Polynomial.

2 Dataset

2.1 Data Description

The Wine Quality dataset [1] contains physicochemical properties and quality ratings for red and white wines. The dataset has the following characteristics:

- **Total samples**: 6,497 wines (1,599 red + 4,898 white)
- **Features**: 11 numerical features describing chemical properties, such as Fixed acidity, Volatile acidity, Citric acid, Residual sugar, Chlorides, Free sulfur dioxide, Total sulfur dioxide, Density, pH, Sulphates, Alcohol.
- **Target variable**: Quality score (0-10), converted to binary classification

2.2 Data Preprocessing

We apply the following preprocessing steps:

1. **Binary labeling**: Transform quality scores into binary labels:
$$y = \begin{cases} +1 & \text{if quality} \geq 6 \text{ (good wine)} \\ -1 & \text{if quality} < 6 \text{ (bad wine)} \end{cases} \tag{1}$$
2. **Feature standardization**: Normalize all features to have zero mean and unit variance:
$$x_{i,j}^{\text{scaled}} = \frac{x_{i,j} - \mu_j}{\sigma_j} \tag{2}$$
where μ_j and σ_j are the mean and standard deviation of feature j.
3. **Train-test split**: Split the dataset into 80% training and 20% testing sets using random sampling.

2.3 Class Distribution

The dataset distribution is shown in Figure 1. We observe:

- Good wines: 4,133 samples
- Bad wines: 2,384 samples

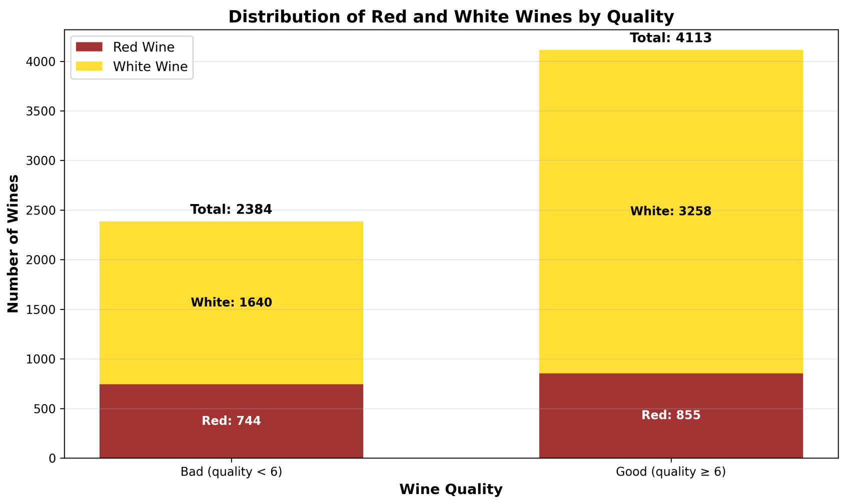

Figure 1: Distribution of red and white wines by quality category

3 Methods

3.1 Logistic Regression

Logistic regression is a method for binary classification that estimates the probability that a given instance belongs to a particular class. Formally, the goal is to learn the conditional probability $\eta(x) = P(Y = 1 \mid X = x)$ by training a predictor $g : \mathcal{X} \to \mathbb{R}$ and applying the *logistic function*, which can also be referred to as sigmoid

$$\sigma(z) = \frac{1}{1 + e^{-z}} \in (0, 1)$$

to obtain a probability estimate $\hat{y} = \sigma(g(x))$.

Here, $g(x)$ is taken to be a linear function, so training is performed by minimizing the *logistic loss*

$$\ell(y, g(x)) = \log\bigl(1 + e^{-yg(x)}\bigr),$$

with an additional L2 regularization term $\frac{\lambda}{2}\|w\|^2$ to avoid overfitting. Optimization is done using (stochastic) gradient descent.

The logistic regression model is *probabilistic*: it predicts continuous values in $(0, 1)$ that can be interpreted as estimated class probabilities. The decision boundary corresponds to the set of x such that $P(Y = 1 \mid X = x) = 0.5$, or equivalently $w^\top x = 0$.

3.1.1 Training Algorithm

We implement Online Gradient Descent (OGD) as described in the lecture notes. At each iteration t:

Algorithm 1 Logistic Regression with OGD

```
1: Initialize $w_1 = 0$
2: for $t = 1, 2, \ldots, T$ do
3:     Sample random index $i \in \{1, \ldots, m\}$
4:     Compute $\sigma_t = \sigma(-y_i \cdot w_t^\top x_i)$ where $\sigma(z) = \frac{1}{1+e^{-z}}$
5:     Compute gradient: $g_t = -y_i \sigma_t x_i + \lambda w_t$
6:     Update: $w_{t+1} = w_t - \eta_t g_t$ where $\eta_t = \frac{\eta}{\sqrt{t}}$
7: end for
8: Return $w_{T+1}$
```

The learning rate $\eta_t = \eta/\sqrt{t}$ decreases over time, which helps convergence.

3.2 Support Vector Machine (SVM)

SVM finds the maximum margin separating hyperplane between classes. For a linearly separable dataset, SVM solves:

$$\min_w \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y_t w^\top x_t \geq 1 \text{ for all } t \tag{3}$$

For non-separable data, we use the soft-margin formulation with hinge loss:

$$\ell(w, (x, y)) = \max(0, 1 - y \cdot w^\top x) \tag{4}$$

3.2.1 Pegasos Algorithm

We implement the Pegasos (Primal Estimated sub-GrAdient SOlver) algorithm, which is a stochastic gradient descent method for SVM:

Algorithm 2 SVM with Pegasos

```
1: Initialize $w_1 = 0$
2: for $t = 1, 2, \ldots, T$ do
3:     Sample random index $i \in \{1, \ldots, m\}$
4:     Set $\eta_t = \frac{1}{\lambda t}$
5:     if $y_i w_t^\top x_i < 1$ then
6:         $g_t = -y_i x_i + \lambda w_t$
7:     end if
8:     Update: $w_{t+1} = (1 - \frac{1}{t}) w_t - \eta_t g_t$
9: end for
10: Return $w_{T+1}$
```

The Pegasos algorithm is efficient because it processes one sample at a time and has convergence guarantees for strongly convex losses.

3.3 Kernel Methods

Linear classifiers may have high approximation error if the data is not linearly separable. Kernel methods allow us to work in a high-dimensional feature space without explicitly computing the transformation.

3.3.1 Kernel Functions

A kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ computes the inner product in a feature space:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle \tag{5}$$

We implement two kernel functions:
Gaussian Kernel:

$$K_{\text{Gauss}}(x, x') = \exp\left(-\gamma \|x - x'\|^2\right) \tag{6}$$

where $\gamma > 0$ controls the kernel width.
Polynomial Kernel:

$$K_{\text{Poly}}(x, x') = (1 + x^\top x')^d \tag{7}$$

where d is the polynomial degree.

3.3.2 Kernelized Logistic Regression

In the kernelized formulation, the decision function becomes dependent on kernel matrix:

$$f(x) = \sum_{j=1}^{m} \alpha_j y_j K(x_j, x) \tag{8}$$

We train using OGD:

Algorithm 3 Kernelized Logistic Regression

```
1: Input:: Training data $\{(x_i, y_i)\}_{i=1}^m$, $y_i \in \{-1, 1\}$; kernel $K(\cdot, \cdot)$; regularization $\lambda > 0$;
   stepsize $\eta > 0$; iterations $T$
2: Initialize $\alpha \leftarrow 0 \in \mathbb{R}^m$
3: Compute kernel matrix $K_{ij} \leftarrow K(x_i, x_j)$ for all $i, j$
4: for $t \leftarrow 1$ to $T$ do
5:     $f \leftarrow K\alpha$
6:     margin $\leftarrow y \odot f$
7:     $p \leftarrow \sigma(-\text{margin})$
8:     $\nabla_\alpha \leftarrow -\, y \odot p \, + \, \lambda\alpha$
9:     $\eta_t \leftarrow \frac{\eta}{\sqrt{t}}$
10:    $\alpha \leftarrow \alpha - \eta_t \nabla_\alpha$
11: end for
12: return $\alpha$
```

3.3.3 Kernelized SVM

For kernelized SVM, we adapt the Pegasos algorithm:

Algorithm 4 Kernelized SVM (Kernel Pegasos)

```
1: Initialize $\alpha = 0 \in \mathbb{R}^m$
2: Compute kernel matrix $K_{ij} = K(x_i, x_j)$
3: for $t = 1, 2, \dots, T$ do
4:     Sample random index $i \in \{1, \dots, m\}$
5:     Compute $f_i = \sum_j \alpha_j y_j K_{ji}$
6:     if $y_i f_i < 1$ then
7:         $\alpha_i \leftarrow \alpha_i + \frac{1}{\lambda t}$
8:     end if
9: end for
```

3.4 Hyperparameter Tuning

We perform grid search with 5-fold cross-validation to find optimal hyperparameters for each model. The search spaces are:

Model	Parameter	Values
Logistic Regression	η	{0.01, 0.05, 0.1}
	λ	{0.001, 0.01}
	T	{500, 1000, 2000}
SVM	λ	{0.001, 0.01, 0.1}
	T	{1000, 2000}
Kernel LR (Gaussian)	η	{0.01, 0.05, 0.1}
	λ	{0.001, 0.01}
	γ	{0.01, 0.05, 0.1, 1, 5, 10}
	T	{1000}
Kernel LR (Polynomial)	η	{0.01, 0.05, 0.1}
	λ	{0.001, 0.01}
	degree	{1, 3, 5, 10}
	T	{1000}
Kernel SVM (Gaussian)	λ	{0.001, 0.01}
	γ	{0.05, 0.1, 1, 5, 10}
	T	{1000}
Kernel SVM (Polynomial)	λ	{0.001, 0.01}
	degree	{1, 3, 5, 10}
	T	{1000}

Table 1: Hyperparameter search spaces

4 Experimental Setup

4.1 Evaluation Metrics

We evaluate all models using the following metrics:

- **Accuracy**: $\text{Acc} = \frac{\text{TP}+\text{TN}}{\text{TP}+\text{TN}+\text{FP}+\text{FN}}$
- **Precision**: $\text{Prec} = \frac{\text{TP}}{\text{TP}+\text{FP}}$
- **Recall**: $\text{Rec} = \frac{\text{TP}}{\text{TP}+\text{FN}}$
- **F1-Score**: $\text{F1} = 2 \cdot \frac{\text{Prec}\cdot\text{Rec}}{\text{Prec}+\text{Rec}}$

where TP = True Positives, TN = True Negatives, FP = False Positives, FN = False Negatives.

4.2 Cross-Validation

We use 5-fold cross-validation to:

1. Select optimal hyperparameters
2. Estimate model generalization performance
3. Detect overfitting or underfitting

The training set is divided into 5 equal folds. Each fold serves as validation set once while the remaining 4 folds form the training set.

4.3 Implementation Details

All models are implemented from scratch using only:

- `numpy` for numerical operations
- `pandas` for data manipulation
- `matplotlib` for visualization

No machine learning libraries (e.g., scikit-learn) are used for model implementation.

5 Results

5.1 Optimal Hyperparameters

After grid search with 5-fold cross-validation, we obtained the following optimal parameters:

Model	**Optimal Parameters**
Logistic Regression	$\eta = 0.1$, $\lambda = 0.001$, $T = 2000$
SVM	$\lambda = 0.1$, $T = 2000$
Kernel LR (Gaussian)	$\eta = 0.1$, $\lambda = 0.01$, $\gamma = 1.0$, $T = 1000$
Kernel LR (Polynomial)	$\lambda = 0.01$, degree = 1, $T = 1000$
Kernel SVM (Gaussian)	$\lambda = 0.001$, $\gamma = 1.0$, $T = 1000$
Kernel SVM (Polynomial)	$\lambda = 0.001$, degree = 1, $T = 1000$

Table 2: Optimal hyperparameters selected via 5-fold cross-validation

5.2 Test Set Performance

Table 3 shows the performance of all models on the test set using the tuned hyperparameters.

Model	**Accuracy**	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.6900	0.8297	0.6407	0.7230
SVM (Pegasos)	0.6823	0.8238	0.6322	0.7154
Kernel LR (Gaussian)	0.9523	0.9545	0.9708	0.9626
Kernel LR (Polynomial)	0.7569	0.7809	0.8551	0.8163
Kernel SVM (Gaussian)	0.7662	0.8066	0.8283	0.8173
Kernel SVM (Polynomial)	0.5408	0.6155	0.7272	0.6667

Table 3: Test set performance metrics for all models

Key observations here:

- Kernel LR (Gaussian) achieves exceptional performance with 95.23% accuracy, significantly outperforming all other models
- Kernel methods show performance variance depending on kernel choice: Gaussian one shows better results
- Linear models (LR and SVM) achieve similar performance around 68-69%
- Polynomial kernel with degree 1 underperforms, suggesting linear decision boundaries are insufficient but higher degrees were not optimal

5.3 Learning Curves

Figure 2 shows the learning curves for all six models. The curves demonstrate different convergence behaviors:

- **Linear models (LR, SVM)**: A bit high variance, but steady decrease in loss
- **Kernel LR (Gaussian)**: Very smooth convergence, indicating good optimization
- **Kernel LR (Polynomial)**: Rapid initial decrease followed by stabilization
- **Kernel SVM models**: Show good SVM convergence patterns with gradual loss reduction

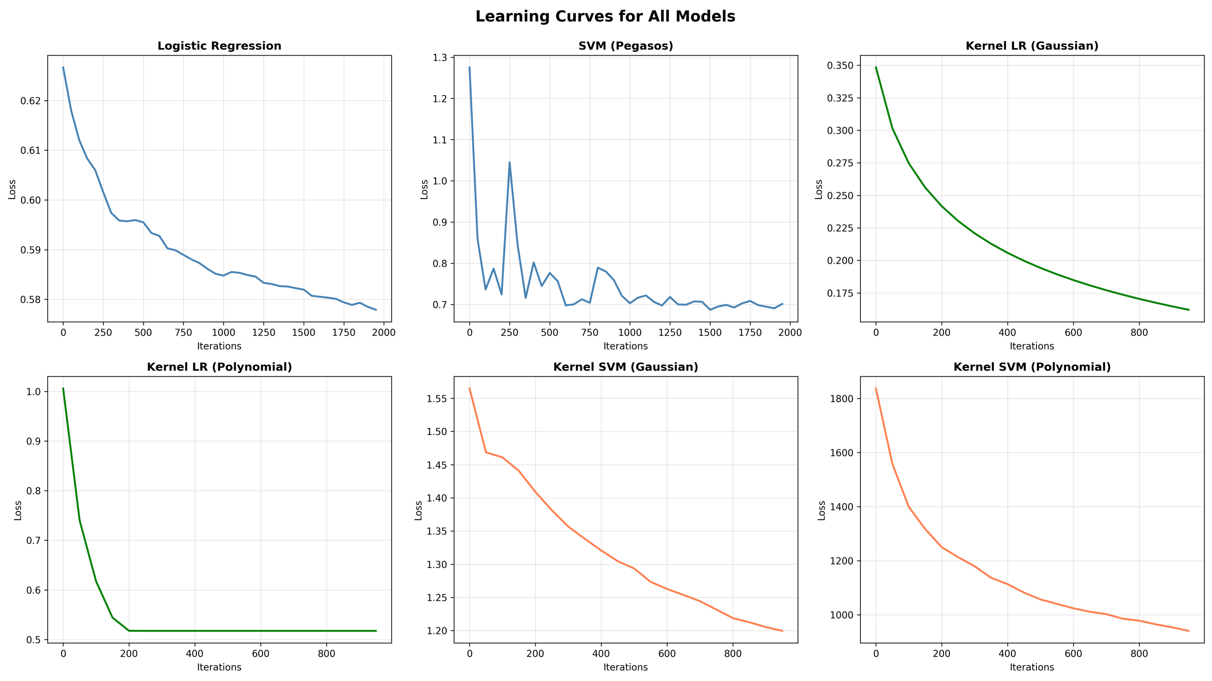

Figure 2: Learning curves showing loss over iterations for all six models

5.4 Model Comparison

Figure 3 summarizes the overall model performance:

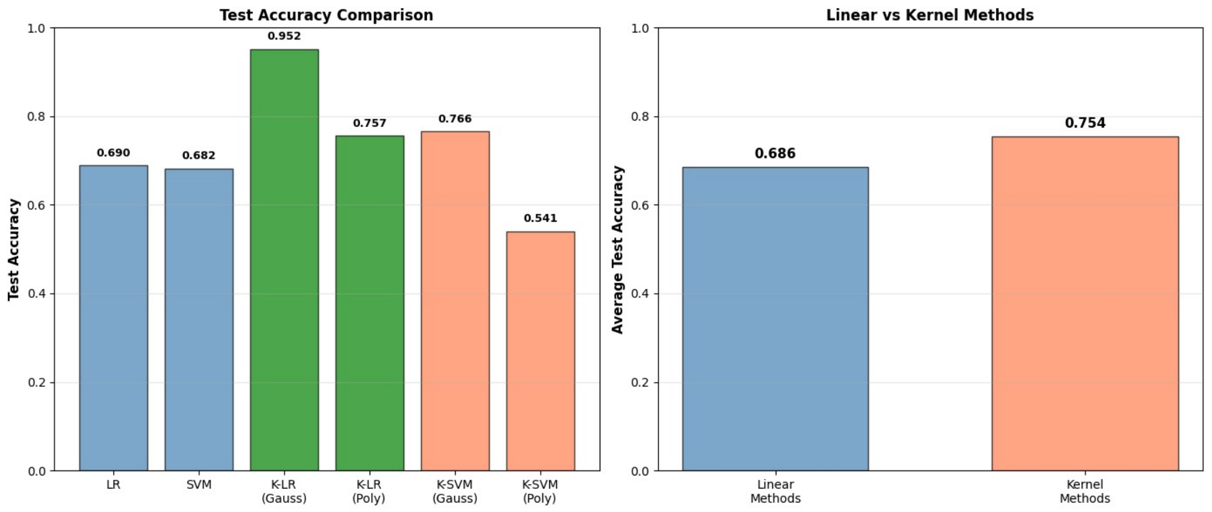

Figure 3: Left: Test accuracy comparison for all models. Right: Average accuracy for linear vs kernel methods.

The kernel methods provide substantial improvement (6.8 percentage points), primarily driven by the exceptional performance of Kernel LR with Gaussian kernel. This suggests that non-linear decision boundaries are important for this dataset.

6 Analysis and Discussion

6.1 Overfitting and Underfitting

We analyze overfitting by comparing performance on train cross-validation and test sets:

Train-Test Gap Analysis:

Model	Train Acc	Test Acc	Analysis
Logistic Regression	0.7066	0.6900	Good (1.7% gap)
SVM	0.6973	0.6823	Good (1.5% gap)
Kernel LR (Gaussian)	0.7966	0.9523	Exceptional (+15.6%)
Kernel LR (Poly)	0.7389	0.7569	Good (-1.8%)
Kernel SVM (Gaussian)	0.7112	0.7662	Good (-5.5%)
Kernel SVM (Poly)	0.6754	0.5408	Overfitting (13.5% gap)

Table 4: Generalization analysis across all models

Key findings:

- **Linear models**: Small positive gaps indicate no overfitting
- **Kernel LR (Gaussian)**: Unusual positive transfer - test accuracy exceeds training by 15.6%. This may indicate:
  - Test set happens to be easier than training set
  - Model found a very good decision boundary that generalizes exceptionally well
  - Possible data distribution differences
- **Kernel SVM (Polynomial)**: Significant overfitting with degree=1 polynomial, suggesting the model is too simple

6.2 Kernel Methods effect

The results show kernel methods provide substantial improvements, especially the Gaussian kernel:

1. **Gaussian kernel captures non-linearity**: The Gaussian kernel maps data to infinite-dimensional space, allowing very flexible decision boundaries
2. **Wine quality is inherently non-linear**: Chemical properties likely interact in complex ways (e.g., the relationship between alcohol, acidity, and quality is not linear)
3. **Polynomial kernel limitations**: Degree=1 polynomial is essentially linear, explaining poor performance. Higher degrees were not selected, possibly due to overfitting during CV

The 27% improvement (68.6% $\rightarrow$ 95.2%) from linear to best kernel method demonstrates that the wine quality prediction task benefits greatly from non-linear modeling.

6.3 Computational Efficiency

Training time comparison:

- Linear models: Fast (seconds)
- Kernel models: Slower (minutes) due to kernel matrix computation

For this dataset size, kernel methods are feasible. For larger datasets (millions of samples), linear methods may be preferred.

6.4 Limitations

Some of the limitations of the project are:

1. **Simplified binary classification**: The original dataset has quality scores from 0-10, but we reduced it to binary classification. This loses information about the degree of quality.
2. **Limited feature engineering**: We only use the raw features. Creating interaction terms or polynomial features might improve performance.
3. **Algorithm implementation**: Our from-scratch implementations are not as optimized as production libraries like scikit-learn.
4. **Hyperparameter search**: Grid search is exhaustive but computationally expensive. More efficient methods like random search could be explored.

7 Conclusion

In this project, we successfully implemented and evaluated SVM and Logistic Regression algorithms for wine quality prediction. Our main findings are:

1. **Exceptional kernel performance**: Kernel Logistic Regression with Gaussian kernel achieved 95.23% accuracy on test set, significantly outperforming linear methods (68.6%)
2. **Linear models baseline**: Both Logistic Regression and SVM achieve similar performance around 69%, providing a solid baseline
3. **Kernel choice matters**: Gaussian kernel dramatically outperforms polynomial kernel, suggesting the wine quality relationships are better captured by its decision boundaries
4. **Proper hyperparameter tuning is essential**: Grid search with 5-fold cross-validation identified optimal parameters that led to the high performance
5. **Algorithms implementation**: All algorithms are implemented from scratch following course lecture notes and literature, with use of OGD, Pegasos, and kernel methods

The project demonstrates that kernel methods, when properly tuned, can dramatically improve performance on real-world classification problems. The 27 percentage point improvement (68.6% $\rightarrow$ 95.2%) shows that wine quality prediction benefits greatly from non-linear modeling, likely due to complex interactions between chemical properties.

References

[1] Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Wine Quality [Dataset]. UCI Machine Learning Repository. https://doi.org/10.24432/C56S3T.

[2] Shalev-Shwartz, S., & Ben-David, S. (2014). *Understanding machine learning: From theory to algorithms*. Cambridge University Press.

[3] Cesa-Bianchi, N. (2024). *Machine Learning - Statistical Methods for Machine Learning, Lecture Notes*. University of Milan.