SVM and Logistic Regression For Wine Quality Classification

Mustecep Berca AKBAYIR / 52448A

June 24, 2025

Abstract

This study aims to classify red and white wines according to their quality scores with machine learning techniques. For modeling part, Support Vector Machines and Logistic Regression binary classifier have been implemented from scratch. The motivation behind this study is that showing implementation of machine learning algorithms from scratch and comparing them regarding to their performances and limitations.

1 Introduction

In this study, Wine Quality Dataset have been used from UCI Machine Learning Repository which includes fixed acidity, volatile acidity, citric acid, residual sugar, chlorides, free sulfure dioxide, total sulfur dioxide, density, pH, sulphates alcohol features of wine. Data has been received as two different dataset for wine and red wines. During analysis and modeling these datasets have been combined. Several statistical data analysis and visualization techniques have been used to show the mystery behind the data. After conducting deep-dive statistical analysis, dataset have been modeled using machine learning models to classify wines according to qualities.

2 Exploratory Data Analysis and Data Preprocessing

2.1 Descriptive Statistics

Wine quality dataset comes from UCI Machine Learning repository as 2 different datasets: red and white wines. Red wine dataset includes 1599 instance, white wine dataset includes 4898 instances. Red and white wine datasets include 12 columns as listed below:

- fixed acidity
- volatile acidity

- cidric acid
- residual sugar
- chlorides
- free sulfur dioxide
- total sulfur dioxide
- density
- pH
- sulphates
- alcohol
- quality

These datasets have been combined for better statistical analysis and modeling purposes.

Summary statistics are represented in the table below:

Column	**Mean**	**Std**	**Min**	**Max**
fixed acidity	7.22	1.30	3.80	15.90
volatile acidity	0.34	0.16	0.08	1.58
citric acid	0.32	0.15	0.00	1.66
residual sugar	5.44	4.76	0.60	65.80
chlorides	0.06	0.04	0.01	0.61
free sulfur dioxide	30.53	17.75	1.00	289.00
total sulfur dioxide	115.74	56.52	6.00	440.00
density	0.99	0.003	0.99	1.04
pH	3.22	0.16	2.72	4.01
sulphates	0.53	0.15	0.22	2.00
alcohol	10.49	1.19	8.00	14.90

Table 1: Summary statistics of wine dataset features

For better representation, distributions of features has been compared below regarding to wine types:

As can be seen in the graph above, many features do not differ between wine types. For example, the fixed acidity and volatile acidity features of wines are seen to have the same distribution and characteristics in red and white wines. However, some distributions differ between wine types. For example, alcohol and density distributions differ between wines. To be sure of the differences in the characteristics among the

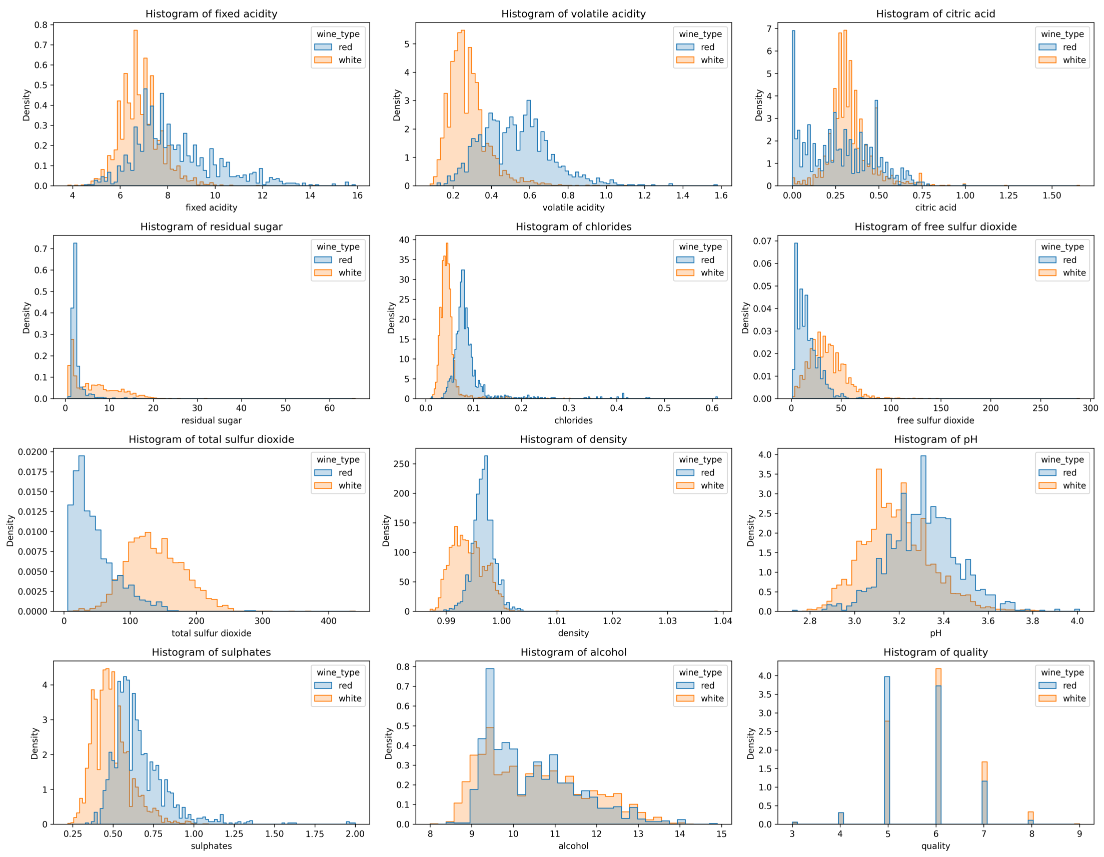

Figure 1: Data Distributions for Wines

wine types, the non-parametric group distribution test ManWhitney-U test was used. The following table shows the variables whose distributions between the groups are statistically significant or insignificant:

Feature	U-statistic	p-value	Significant
volatile acidity	7,059,623.5	0.000000e+00	No
chlorides	7,407,015.5	0.000000e+00	No
free sulfur dioxide	1,186,396.5	0.000000e+00	No
total sulfur dioxide	366,639.5	0.000000e+00	No
sulphates	6,509,961.0	0.000000e+00	No
fixed acidity	6,138,507.0	1.438930e-255	No
density	6,059,284.5	1.453091e-237	No
pH	5,681,839.5	5.472258e-162	No
residual sugar	2,569,687.0	5.634073e-95	No
citric acid	3,070,088.5	1.312558e-38	No
quality	3,311,514.0	3.634341e-23	No
alcohol	3,829,043.5	1.818451e-01	Yes

Table 2: Mann–Whitney U test results for each feature

As observed in the table above, many characteristics do not differ between wine types. However, alcohol content has a statistically significant difference between wine types.

2.2 Missing Value Analysis

After applying the missing value analysis to the dataset, it was observed that there was no missing data.

Column	Missing Percentage
fixed acidity	0.0%
volatile acidity	0.0%
citric acid	0.0%
residual sugar	0.0%
chlorides	0.0%
free sulfur dioxide	0.0%
total sulfur dioxide	0.0%
density	0.0%
pH	0.0%
sulphates	0.0%
alcohol	0.0%
quality	0.0%
wine_type	0.0%

Table 3: Percentage of missing values per column

2.3 Data Encoding

After merging the data sets of red and white wines into a single data set, the wine type was added to the data set as a new feature. Since this newly added feature is categorical data, it needs to be encoded. Since the data is nominal, it is encoded by applying One-Hot Encoding.

2.4 Correlation Analysis

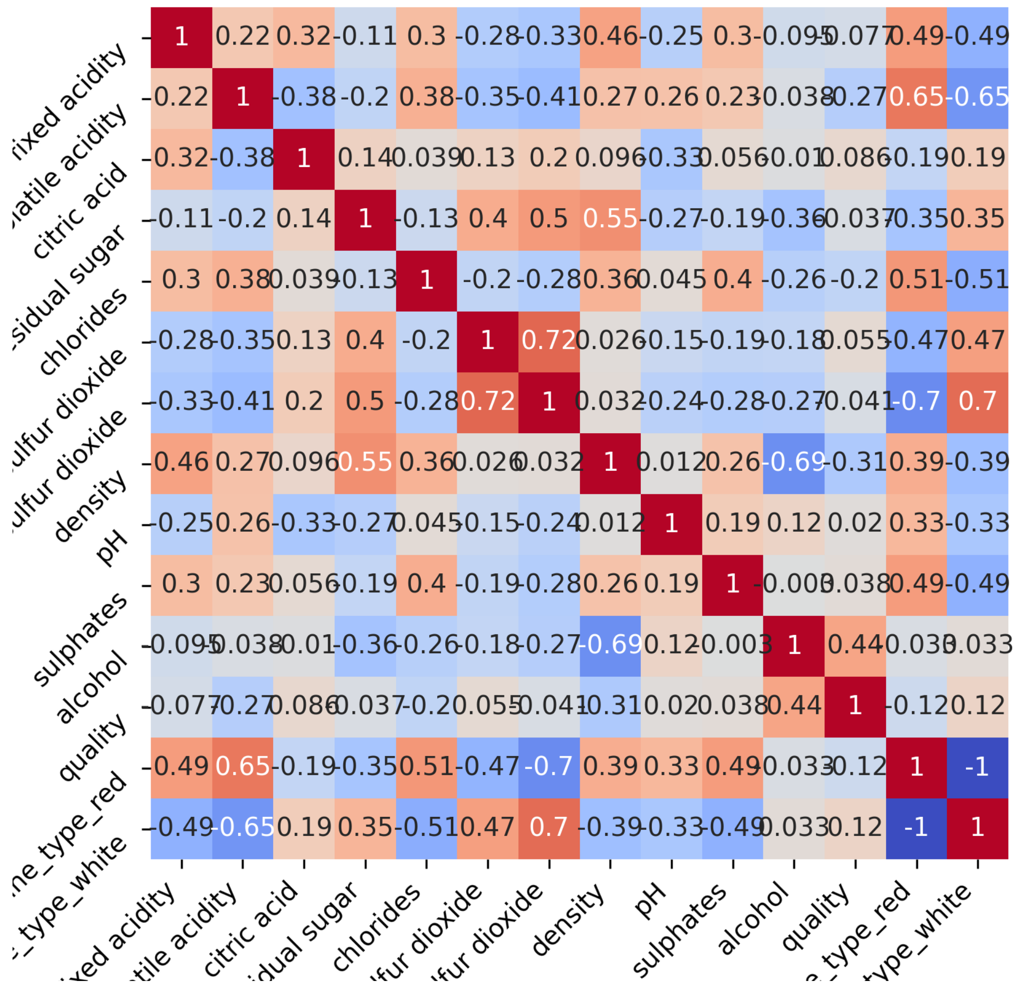

Figure 2: Correlation Heatmap

The graph above shows a heatmap showing the correlation values and directions between the variables. The most striking point in the graph is that the wine_type_red and wine_type_white features obtained after the data encoding stage are completely dependent on each other with a 100 percent negative correlation, which is already expected. Since a complete dependency is mentioned here, one of the two variables will be excluded from the modeling and the multicolinearity problem will be solved. However, high correlation values are also observed in some other variables. For example, a 70 percent negative correlation is observed between density and alcohol. Or, a 70 percent positive relationship is observed between the whiteness of the wine and the total sulfur dioxide rate. At these points, a multicolinearity problem can be suspected, but an additional analysis will be needed to be sure. This analysis will be explained in the following steps.

2.5 Target Base Analysis

In the dataset, quality scores are divided into good and bad. If the wine's quality score is 6 or higher, this wine is considered good. However, if the wine's quality score is lower than 6, the wine is considered bad.

The graph below shows how much the distribution of features changes between good and bad wines. The most important point that stands out is the difference in density and alcohol ratios between good and bad wines.

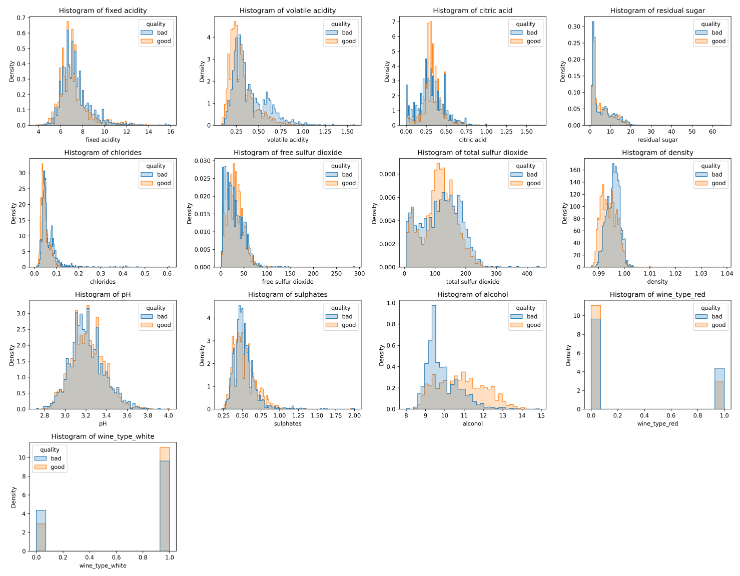

Figure 3: Quality Score Based Distributions

To make sure the distribution differences, ManWhitney-U test was applied to measure the distributions between good and bad wines. The results are as follows:

Feature	U-statistic	p-value	Significant
volatile acidity	7,059,623.5	0.000000e+00	No
chlorides	7,407,015.5	0.000000e+00	No
free sulfur dioxide	1,186,396.5	0.000000e+00	No
total sulfur dioxide	366,639.5	0.000000e+00	No
sulphates	6,509,961.0	0.000000e+00	No
fixed acidity	6,138,507.0	1.438930e-255	No
density	6,059,284.5	1.453091e-237	No
pH	5,681,839.5	5.472258e-162	No
residual sugar	2,569,687.0	5.634073e-95	No
citric acid	3,070,088.5	1.312558e-38	No
quality	3,311,514.0	3.634341e-23	No
alcohol	3,829,043.5	1.818451e-01	Yes

Table 4: Mann–Whitney U test results for each feature

As seen above, there is a statistically significant difference in alcohol content between good and bad wines.

2.6 Outlier Analysis

After applying outlier analysis with IQR, the outlier rates within the features are given in the table and boxplots below.

Column	Outlier Percentage
fixed acidity	1.32%
volatile acidity	0.72%
citric acid	0.23%
residual sugar	0.05%
chlorides	2.05%
free sulfur dioxide	0.12%
total sulfur dioxide	0.02%
density	0.02%
pH	0.03%
sulphates	0.54%
alcohol	0.00%

Table 5: Outlier percentages for each column

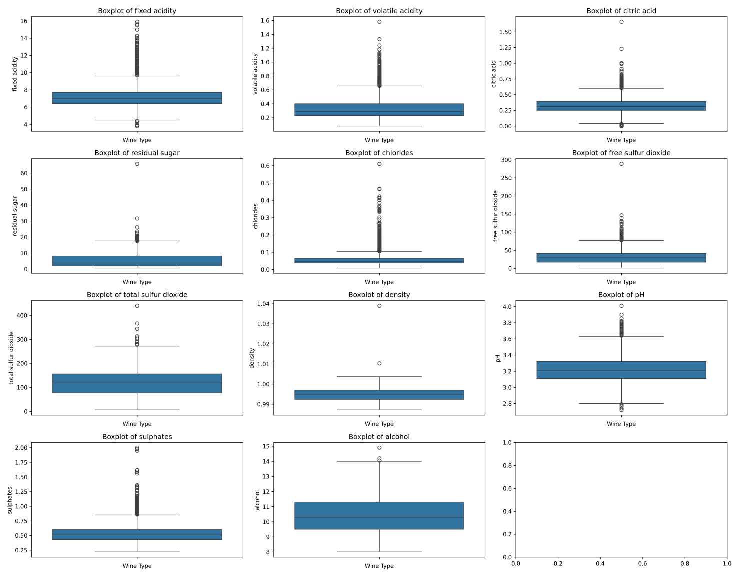

Figure 4: Boxplots

To avoid outliers, a capping strategy was applied. Outliers detected by the IQR method were imputed with the lower and upper limits determined by IQR to avoid outliers.

2.7 Data Scaling

Data scaling is a very important step for both SVM and logistic regression. While SVM calculates the distances between points in the modeling process, features with different scales can be directed to features with a wider range and can negatively affect performance. In statistical models such as logistic regression, features with higher values in the formulation can dominate the others and distort the probability calculations. Therefore, data scaling was applied for a healthy modeling process. The formulation applied for data scaling is given below:

$$x_i^{\text{scaled}} = \begin{cases} \dfrac{x_i - \mu}{\sigma}, & \text{if } \sigma \neq 0 \\ 0, & \text{if } \sigma = 0 \end{cases}$$

With this formulation, all continuous data are standardized and made ready for modeling.

2.8 Skewness

When skewness analysis was performed on the dataset, the results in the table below were obtained:

Column	**Skewness**
fixed acidity	1.31
volatile acidity	1.30
citric acid	0.26
residual sugar	1.16
chlorides	1.60
free sulfur dioxide	0.72
total sulfur dioxide	-0.01
density	0.04
pH	0.38
sulphates	1.09
alcohol	0.57

Table 6: Skewness values for each column

For features with a skewness value greater than 0.5, the skewness value can be considered large. Data transformation should be performed for these data. The data was transformed with the Boxcox transformation with the formula below:

$$y^{(\lambda)} = \begin{cases} \dfrac{x^{\lambda} - 1}{\lambda}, & \text{if } \lambda \neq 0 \\ \log(x), & \text{if } \lambda = 0 \end{cases}$$

The Box-Cox transformation is a way to make data look more "normal" or balanced, especially when the data is skewed. It works by applying a power transformation to the numbers, depending on a special parameter called lambda. If lambda is zero, it simply takes the log of the data; otherwise, it raises the data to the power of lambda in a smooth way. The goal is to find the best lambda that makes the data behave nicely, which helps improve the accuracy and reliability of many statistical analyses and machine learning models. It's especially useful when you want your data to meet assumptions like having constant variance or being normally distributed.

Applying data transformations such as Box-Cox transformation to data because skewed data may affect estimated coefficients in models and decrease the performance of models.

2.9 Multicolinearity Analysis

To look at the multicolinearity problem in the data, we first calculated the correlations between the variables. In this step, Variance Inflation Rate (VIF) was used to numerically express the multicolinearity problem.

Feature	**VIF**
fixed acidity	inf
wine_type_red	inf
wine_type_white	inf
density	3.62
total sulfur dioxide	3.36
free sulfur dioxide	2.07
pH	1.96
chlorides	1.77
alcohol	1.44
citric acid	1.27
residual sugar	1.00
volatile acidity	0.91
sulphates	0.31

Table 7: Variance Inflation Factor (VIF) for each feature

In the table above, the VIF ratios of the variables can be clearly observed. As observed in the table, wine types have a multicolinearity problem. In addition, the fixed acidity ratio is in the same situation. Before modeling, one of the variables representing the wine types will be removed from the dataset. In addition, the remaining multicolinearity problem will be smoothed out by using the l2 regularization term in the modeling phase.

2.10 Target Balance Analysis

The distribution of the wine quality score variable to be used as the target variable in the modeling phase is as follows:

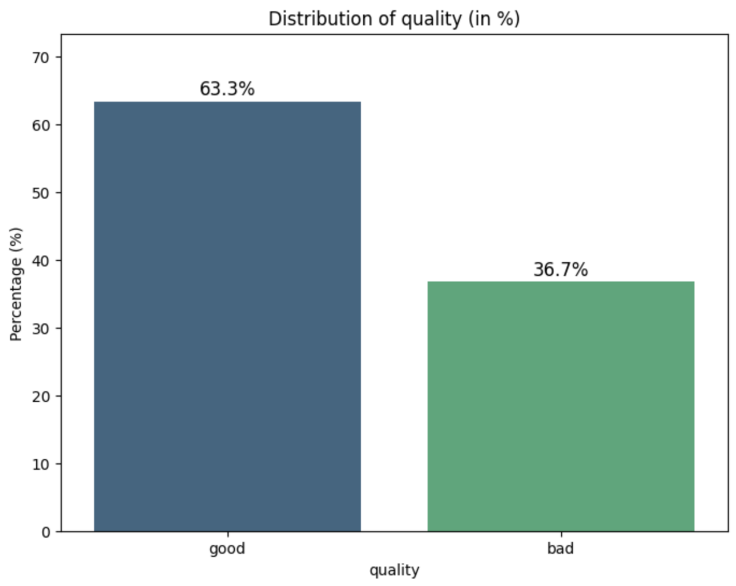

Figure 5: Correlation Heatmap

There seems to be a certain amount of imbalanced data set problem, but it is not enough to negatively affect the modeling. Shannon Entropy from Information Theory can be used to numerically express the imbalance of the data set:

$$H = -\sum_{i=1}^{k} p_i \log(p_i)$$

where

$$k = \text{number of unique classes in the sequence,}$$

$$p_i = \frac{n_i}{n} \quad \text{is the probability of class } i,$$

$$n_i = \text{count of class } i \text{ in the sequence,}$$

$$n = \text{total number of elements in the sequence.}$$

which scales the entropy to lie between 0 and 1.

The result of the Shannon entropy reached 0.94 for the data set. This means that the data set can be considered balanced.

3 Modeling

Support Vector Machines and Logistic Regression binary classifiers were created from scratch to classify the wines in the dataset according to their quality scores. Before modeling, 80% of the dataset was separated for training set and 20% for test set.

3.1 Support Vector Machines

Support Vector Machines (SVM) are widely used for binary classification tasks due to their ability to find an optimal separating hyperplane that maximizes the margin between classes. The margin is defined by the support vectors, which are the closest data points to the decision boundary. By maximizing this margin, better generalization to unseen data is achieved. Both linear and non-linear classification problems can be addressed by applying different kernel functions that implicitly map the data into higher-dimensional spaces, allowing a linear separation to be found.

In this implementation, an SVM model was developed to support both linear and kernelized classification. For the linear kernel, the model is trained using an iterative gradient descent approach with hinge loss, including L1 and L2 regularization terms to update weights and bias over multiple iterations. The learning rate is gradually decreased to improve convergence. For non-linear kernels such as polynomial and RBF, the dual optimization problem is solved using quadratic programming to identify support vectors and their coefficients (alphas). Additionally, prediction and probability estimation functions are provided, with probability estimates aligned to a specified positive class. Feature importance can also be extracted when using the linear kernel. This implementation allows customization of hyperparameters such as regularization strengths, kernel type, and kernel-specific parameters to adapt the model to different datasets and classification challenges.

3.1.1 Explanation and Pseudo-Code of SVM

Hyperparameters:

learning rate: Controls the step size during weight updates in the linear SVM training loop.

l2 term: Penalizes large weights to prevent overfitting by adding a term proportional to the squared magnitude of weights.

l1 term: Adds sparsity by penalizing the absolute values of weights. Can drive some weights exactly to zero.

iteration limit: Number of passes over the training data during gradient descent in linear kernel mode.

kernel: Maps data into a higher-dimensional feature space for classification. In Linear Kernel, basic dot product has been used. In ploynomial kernel, 2nd degree of linear kernel has been used. Radial Basis Function kernel is non-linear decision boundries.

C: Controls the trade-off between maximizing the margin and minimizing classification errors in the kernel version. Smaller C means more regularization (larger margin but more misclassifications), larger C tries to classify all points correctly but may overfit.

Gamma: Defines how far the influence of a single training example reaches in the Radial Basis Function kernel.

Initialization

If kernel is linear, set weights and bias as zero.

Training

Algorithm 1 Training with Linear Kernel

```
0: for epoch = 1 to $n_{\text{iters}}$ do
0:   for each training sample $(\mathbf{x}_i, y_i)$ do
0:     Compute margin:
           $$m \leftarrow y_i(\mathbf{w} \cdot \mathbf{x}_i + b)$$
0:     if $m \geq 1$ then
0:       Update weights with regularization only:
           $$\mathbf{w} \leftarrow \mathbf{w} - \eta\,(2\lambda_{l2}\mathbf{w} + \lambda_{l1}\,\text{sign}(\mathbf{w}))$$
0:     else
0:       Update weights and bias using hinge loss gradient:
           $$\mathbf{w} \leftarrow \mathbf{w} - \eta\,(2\lambda_{l2}\mathbf{w} + \lambda_{l1}\,\text{sign}(\mathbf{w}) - y_i\mathbf{x}_i)$$
           $$b \leftarrow b - \eta(-y_i)$$
0:     end if
0:   end for
0: end for=0
```

Else (kernel is nonlinear):

- Compute kernel matrix K for all pairs in the training set
- Solve quadratic programming problem to find Lagrange multipliers α
- Select support vectors with $\alpha_i >$ threshold
- Compute bias b using support vectors

Prediction

Given input vector $\mathbf{x}$:

If kernel is linear:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$$

Else (kernel is nonlinear):

$$f(\mathbf{x}) = \sum_i \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) + b$$

Assign class label $\hat{y}$ as:

$$\hat{y} = \begin{cases} y_+ & \text{if } f(\mathbf{x}) \geq 0 \\ \text{other class} & \text{otherwise} \end{cases}$$

3.1.2 Kernel Space Implementation

$$\text{Linear kernel:} \quad K(\mathbf{x}, \mathbf{y}) = \mathbf{x} \cdot \mathbf{y}^T$$

$$\text{Polynomial kernel (degree 2):} \quad K(\mathbf{x}, \mathbf{y}) = \left(1 + \mathbf{x} \cdot \mathbf{y}^T\right)^2$$

$$\text{Radial Basis Function (RBF) kernel:} \quad K(\mathbf{x}, \mathbf{y}) = \exp\left(-\gamma \|\mathbf{x} - \mathbf{y}\|^2\right)$$

3.1.3 Training Process of SVM

Training of Support Vector Machine for binary classification has been initialized with 0.0001 learning rate and 0.001 l2 regularization term with 100 epoch. These parameters have chosen manually to avoid overfitting and guarantee better converge at the first step. Converge of loss function and learning process can be observed in the plot below:

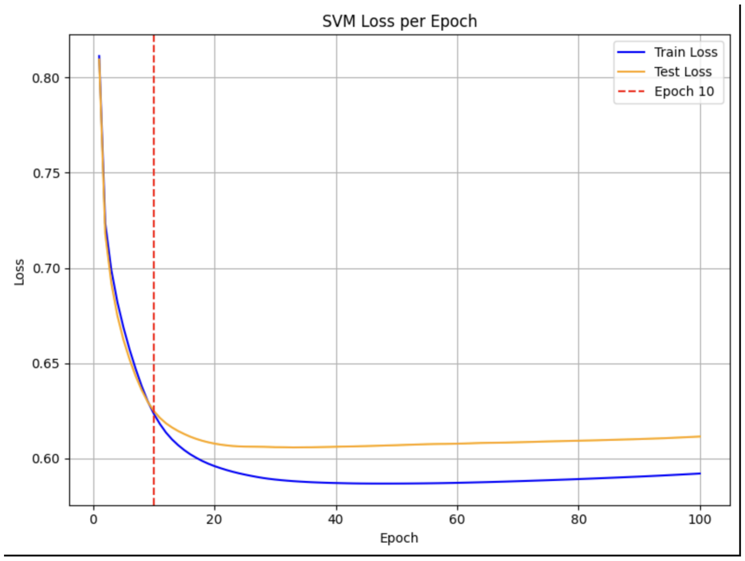

Figure 6: Train - Test Loss for every epoch

It can be observed that the SVM model shows a good convergence rate during the first 20 epochs. However, after the 20th epoch, the model's loss stabilizes and stops improving significantly. Additionally, no signs of overfitting are observed in the first 20 epochs, but after that point, overfitting becomes evident as the gap between the training and test losses increases noticeably. Therefore, selecting the 10th epoch as the optimal stopping point is a reasonable choice. At this point, the model achieves a good balance between performance and generalization, effectively avoiding overfitting.

Overall results can be seen in the tables below:

	Predicted Good	**Predicted Bad**
Actual Good	700	148
Actual Bad	201	250

Table 8: Confusion matrix of the classification results

Metric	**Score**
Accuracy	0.7313
Precision	0.7769
Recall	0.8255
F1-score	0.8005

Table 9: Evaluation metrics of the classifier

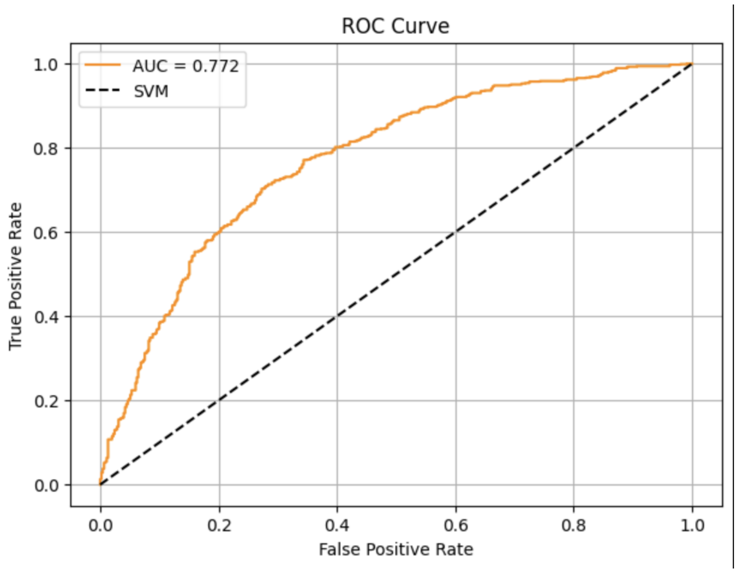

Figure 7: ROC-AUC of SVM

According to ROC-AUC curve, SVM is moderately strong, has good sensitivity and specificity trade-offs. AUC of 0.772 means that there's a 77.2% chance that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

The model uses the volatile acidity feature the most when classifying good and bad wines, as seen in the graph below. However, it has been observed that the alcohol content of the wine is an important variable in classifying good and bad wines. It has been observed that the pH value or cidric acid of the wine is not very important in classification.

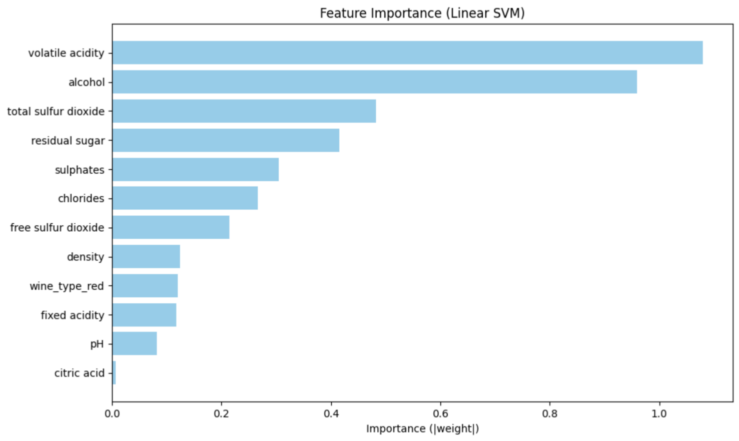

Figure 8: Feature Importance of SVM

3.2 Logistic Regression

Logistic regression is a supervised learning algorithm used for binary classification. It models the probability that a given input belongs to a particular class using the logistic (sigmoid) function, which maps any real-valued number to a value between 0 and 1. The algorithm learns weights for the input features to fit a linear decision boundary, and the output is interpreted as the probability of the positive class. A threshold (usually 0.5) is then applied to assign the input to one of the two classes.

3.2.1 Explanation and Pseudo-Code of SVM

Initialization

- Encode labels $y \in \{\text{positive\_class}, \text{other\_class}\}$ to $y_i \in \{+1, -1\}$
- If kernel is linear:
  - Initialize weight vector $\mathbf{w} = \mathbf{0}$, no explicit bias term (absorbed in $\mathbf{w}$)
- Else (nonlinear kernel):
  - Compute kernel matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
  - Initialize dual weights $\boldsymbol{\alpha} = \mathbf{0}$

Training
If kernel is linear:

Algorithm 2 Training with Linear Kernel (Logistic Loss)

```
0: for epoch = 1 to $n_{\text{iters}}$ do
0:     Compute prediction: $z_i \leftarrow \mathbf{w} \cdot \mathbf{x}_i$
0:     Compute probability: $\hat{y}_i \leftarrow \sigma(z_i) = \frac{1}{1+e^{-z_i}}$
0:     Compute error: $e_i \leftarrow \hat{y}_i - \frac{y_i+1}{2}$
0:     Compute gradient:
           $$\nabla \mathbf{w} \leftarrow \frac{1}{N} \sum_{i=1}^{N} e_i \mathbf{x}_i + \lambda_{l2} \mathbf{w} + \lambda_{l1} \operatorname{sign}(\mathbf{w})$$
0:     Update weights:
           $$\mathbf{w} \leftarrow \mathbf{w} - \eta \cdot \nabla \mathbf{w}$$
0: end for=0
```

Else (nonlinear kernel):
Prediction
Given a new sample $\mathbf{x}$:

Algorithm 3 Training with Kernel Trick (Dual Form)

```
0: Compute kernel matrix $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$
0: Initialize dual weights $\boldsymbol{\alpha} = \mathbf{0}$
0: for epoch = 1 to $n_{\text{iters}}$ do
0:     Compute prediction: $z = K \cdot \boldsymbol{\alpha}$
0:     Compute probability: $\hat{y} = \sigma(z)$
0:     Compute error: $\boldsymbol{e} = \hat{y} - \frac{y+1}{2}$
0:     Compute gradient:
           $$\nabla\boldsymbol{\alpha} \leftarrow \frac{1}{N} K^\top \boldsymbol{e} + \lambda_{l2}\boldsymbol{\alpha} + \lambda_{l1}\,\mathrm{sign}(\boldsymbol{\alpha})$$
0:     Update dual weights:
           $$\boldsymbol{\alpha} \leftarrow \boldsymbol{\alpha} - \eta \cdot \nabla\boldsymbol{\alpha}$$
0: end for=0
```

If kernel is linear:

$$f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x}$$

Else (nonlinear kernel):

$$f(\mathbf{x}) = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}, \mathbf{x}_i)$$

Predict probability:

$$\hat{p} = \frac{1}{1 + \exp(-f(\mathbf{x}))}$$

Assign class:

$$\hat{y} = \begin{cases} \text{positive\_class} & \text{if } \hat{p} \geq 0.5 \\ \text{negative\_class} & \text{otherwise} \end{cases}$$

3.2.2 Kernel Space Implementation

Linear Kernel When the linear kernel is selected, the algorithm operates in the original input space without applying any transformation. In this case, the feature mapping is the identity function, and the kernel function is simply the dot product:

- **Kernel function:**

$$K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$$

- **Feature mapping:**

$$\phi(\mathbf{x}) = \mathbf{x}$$

- **Decision function:**
$$f(\mathbf{x}) = \mathbf{w}^\top \mathbf{x} + b$$

Here, the model parameters w and b are learned directly through gradient-based updates using the hinge loss and regularization terms (L1 and L2). This implementation is efficient and interpretable, and it is particularly effective when the data is linearly separable or nearly so.

Nonlinear Kernel A nonlinear kernel function is implemented to represent the data and seperate them better in the solution space.. This allows the SVM to operate in a transformed feature space without explicitly computing the transformation identity function. Instead, it relies on the kernel trick, where inner products in the high-dimensional space are computed directly via the kernel function:

- **Kernel function:**
$$K(\mathbf{x}, \mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

Here, the parameters alpha are obtained by solving a quadratic programming problem, and only a subset of them corresponding to support vectors are non-zero. Common choices for nonlinear kernels include the polynomial kernel and the radial basis function (RBF) kernel.

This implementation is more computationally intensive due to the need to compute the kernel matrix and solve the dual optimization problem, but it provides superior performance on data with nonlinear structures.

Nonlinear kernel spaces that implemented in algorithm:

- **Polynomial Kernel (degree d):**
$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}' + c)^d$$

- **Radial Basis Function (RBF) Kernel:**
$$K(\mathbf{x}, \mathbf{x}') = \exp\left(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2\right)$$

3.2.3 Training Process of Logistic Regression

The training process of Logistic regression for binary classification has been initialized with 0.000001 learning rate, 0.001 l2 regularization term and 500 epochs, 0.1 gamma. These parameters have chosen manually to avoid overfitting and guarantee better converge at the first step. Converge of loss function and learning process can be observed in the plot below:

The learning process continues successfully for 500 epochs during the training of the logistic regression binary classifier. Although it has a slower converge than the SVM algorithm, it has similar success. After epoch 350, a certain amount of overfitting is observed because train loss is much lower than test loss. In order to prevent overfitting and shorten the train time, the best cut-off point was decided to be 350.

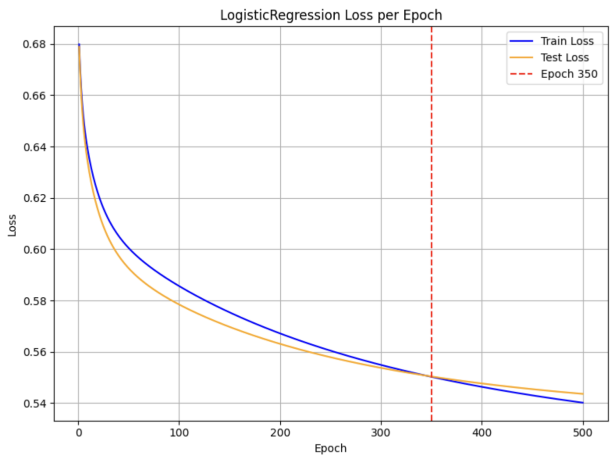

Figure 9: Train - Test Loss for every epoch

	Predicted Good	**Predicted Not-Good**
Actual Good	699	149
Actual Not-Good	227	224

Table 10: Confusion matrix of the classification results

Metric	**Value**
Accuracy	0.7105
Precision	0.7549
Recall	0.8243
F1-score	0.7880

Table 11: Evaluation metrics of classifier

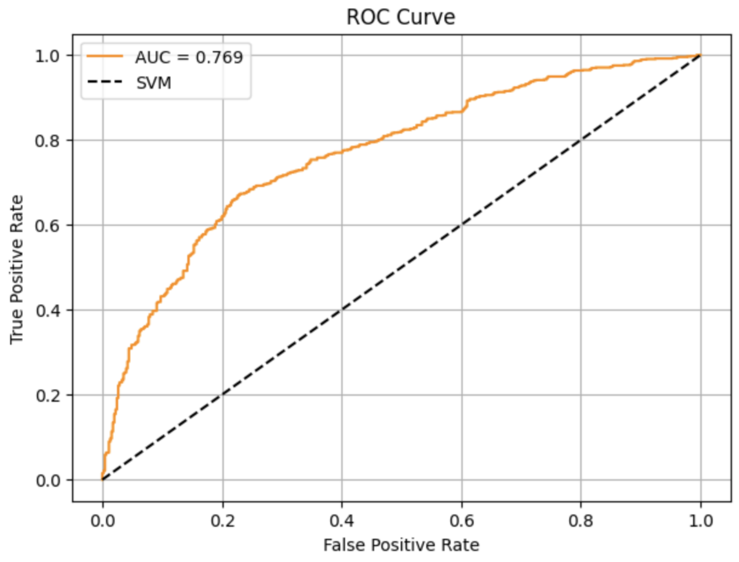

Figure 10: ROC-AUC of Logistic Regression

According to ROC-AUC curve, SVM is moderately strong, has good sensitivity and specificity trade-offs. AUC of 0.769 means that there's a 76.9% chance that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative one.

According to the table below, Logistic regression used alcohol ratios the most to separate good wines from bad wines. The most useful variable to detect bad wines is density. In this model, the variable that provides the least separation while classifying wines is the variable that indicates whether the wine is red or not.

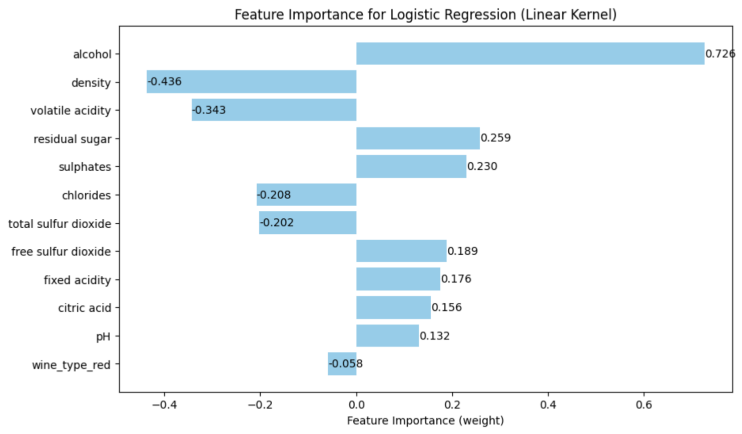

Figure 11: Feature Importances of Logistic Regression

3.3 Hyperparameter Optimization

The Nested Cross Validation function implemented in the project is seen in the pseudo code above. It copies the parameters of the given learning algorithm within itself and tries to find the best hyperparameter with 5-fold data in the inner loop. In the outer loop, the final performance is evaluated with the best parameters of the learning algorithm. Thus, both the best parameters are found and the performance of the model is evaluated in a cross-validation function.

Input: Dataset $D = \{(x_i, y_i)\}_{i=1}^{N}$, model f_θ, parameter grid Θ, outer folds K_{outer}, inner folds K_{inner}, scoring function s.

Output: List of outer scores $\{\text{Score}_{\text{outer}}^{(k)}\}_{k=1}^{K_{\text{outer}}}$, best hyperparameters per fold $\{\theta^{*(k)}\}_{k=1}^{K_{\text{outer}}}$.

```
1. For $k = 1$ to $K_{\text{outer}}$:
    (a) Split $S$ into $D_{\text{train}}^{(k)}$ and $D_{\text{test}}^{(k)}$
    (b) Initialize: $\theta^{*(k)} \leftarrow \emptyset$, best score $\leftarrow -\infty$
    (c) For each $\theta \in \Theta$:
        i. Initialize inner score list: $S_{\text{inner}} \leftarrow [\,]$
        ii. For $j = 1$ to $K_{\text{inner}}$:
            A. Split $D_{\text{train}}^{(k)}$ into $D_{\text{train}}^{(k,j)}$ and $D_{\text{val}}^{(k,j)}$
            B. Train $f_\theta^{(k,j)} \leftarrow \text{train}(D_{\text{train}}^{(k,j)}, \theta)$
            C. Compute score: $s^{(k,j)} \leftarrow s(f_\theta^{(k,j)}, D_{\text{val}}^{(k,j)})$
            D. Append $s^{(k,j)}$ to $S_{\text{inner}}$
        iii. Compute mean score:
                $\text{Score}_{\text{inner}}(\theta) = \frac{1}{K_{\text{inner}}} \sum_{j=1}^{K_{\text{inner}}} s^{(k,j)}$
        iv. If $\text{Score}_{\text{inner}}(\theta) > \text{best score}$:
            • best score $\leftarrow \text{Score}_{\text{inner}}(\theta)$
            • $\theta^{*(k)} \leftarrow \theta$
    (d) Train final model on outer train set:
                $f^{(k)} \leftarrow \text{train}(D_{\text{train}}^{(k)}, \theta^{*(k)})$
    (e) Compute test score on outer test set:
                $\text{Score}_{\text{outer}}^{(k)} = s(f^{(k)}, D_{\text{test}}^{(k)})$
2. Compute mean outer score:
                $\bar{s}_{\text{outer}} = \frac{1}{K_{\text{outer}}} \sum_{k=1}^{K_{\text{outer}}} \text{Score}_{\text{outer}}^{(k)}$
```

3.3.1 Cross-Validation Results

Hyperparameter optimization and performance evaluation were performed with non-linear kernel spaces, RBF and Polynomial kernels for each model. The aim is to compare the performance between kernel spaces and to determine the best kernel. The best results obtained for SVM and Logistic regression models are given below. SVM achieved the best result with RBF, while Logistic regression achieved the best result with polynomial kernel. Other parameters optimized for SVM are C, gamma, l2 term and learning rate. For Logistic regression, polynomial degree, gamma, l2 term and learning rate.

Model	**Best Fold Score**
SVM	0.7644
Logistic Regression	0.6449

Table 12: Best Fold Scores for SVM and Logistic Regression

The general performance comparisons of the models are given in the graph below. In general, SVM gave better and more stable results than logistic regression with kernel spaces.

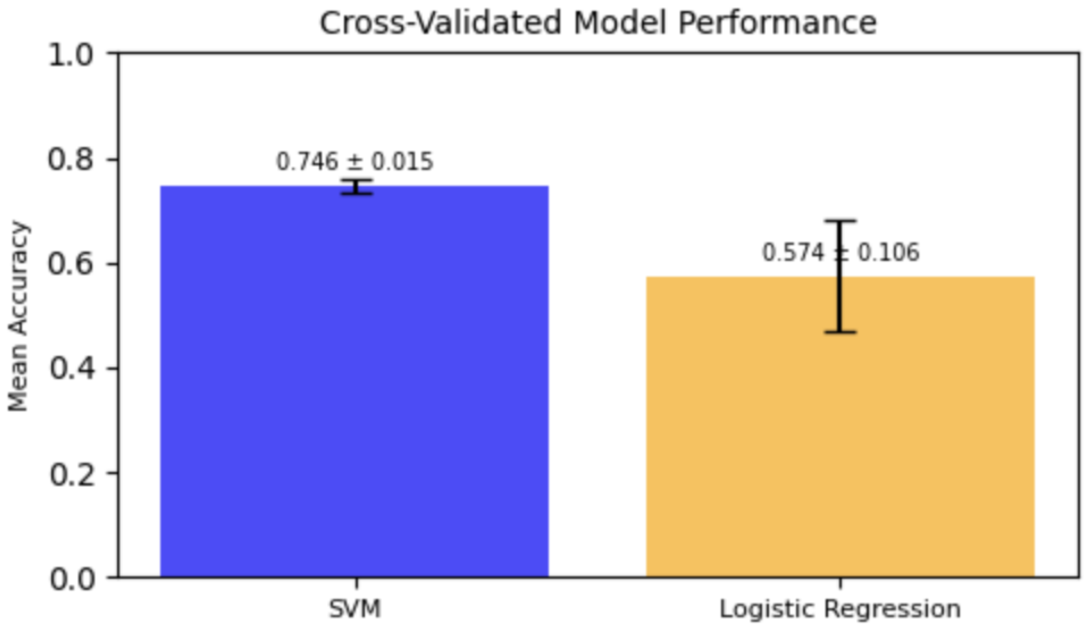

Figure 12: Model Comparison

4 Conclusion

In this study, it was shown how machine learning should be used to classify wines as good or bad according to their quality scores and how these models should be built from scratch. A comparison was made by witnessing the limitations of SVM and logistic regression models. The behaviors of each model in kernel spaces were observed. Problems such as overfitting and unstable gradient descent were addressed and solved with different strategies for models that handle the data differently during the model training process.

5 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.

References

[1] Scikit-learn, "Support Vector Machines," [Online]. Available: `https://scikit-learn.org/stable/modules/svm.html`. [Accessed: 24-Jun-2025].

[2] V. Vapnik, "The Nature of Statistical Learning Theory," IEEE, 1995. [Online]. Available: `https://ieeexplore.ieee.org/document/708428`. [Accessed: 24-Jun-2025].

[3] Scikit-learn, "Logistic Regression — sklearn.linear_model.LogisticRegression," [Online]. Available: `https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html`. [Accessed: 24-Jun-2025].

[4] Scikit-learn, "Nested vs. Non-Nested Cross-Validation," [Online]. Available: `https://scikit-learn.org/stable/auto_examples/model_selection/plot_nested_cross_validation_iris.html`. [Accessed: 24-Jun-2025].

[5] N. Cesa-Bianchi, "Mathematical Statistical Learning," University of Milan, Academic Year 2024–25. [Online]. Available: `https://cesa-bianchi.di.unimi.it/MSA/index_24-25.html#theory`. [Accessed: 24-Jun-2025].