Predicting Wine Quality with Machine Learning

Sara Mancini - 66458A

July 25, 2025

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.

Introduction

The aim of this project is to develop machine learning models from scratch to predict wine quality based on physicochemical properties such as acidity, pH, sulphates, and alcohol content. Logistic regression and support vector machines (SVM) were implemented in both linear and kernelized forms (RBF and polynomial), with all models designed to be scikit-learn-compatible to support systematic hyperparameter tuning. Each model's performance was evaluated in depth using performance metrics such as accuracy, precision, recall, and F1-score, computed through stratified k-fold cross-validation. Confusion matrices, training curves, and misclassification analyses were also used to assess generalization and interpret feature effects.

1. Data Exploration and Preprocessing

1.1 Dataset overview and observations

The dataset includes 6,497 Portuguese wines (1,599 red and 4,898 white). Each sample has 11 continuous numeric features from standard physicochemical tests (e.g., acidity, sugar, sulphates, alcohol), and a wine type label: 0 for red, 1 for white.

The target variable represents a wine's quality score, derived from human sensory evaluation, and is defined on a scale from 0 to 10. However, within this dataset, only scores from 3 to 9 are observed—there are no samples rated at the extreme ends of the scale. The majority of wines are rated between 5 and 7, with a clear mode at 6, indicating a concentration of average-quality wines. Figure 1 illustrates the overall distribution of scores, while Figure 2 presents the distribution broken down by wine type.

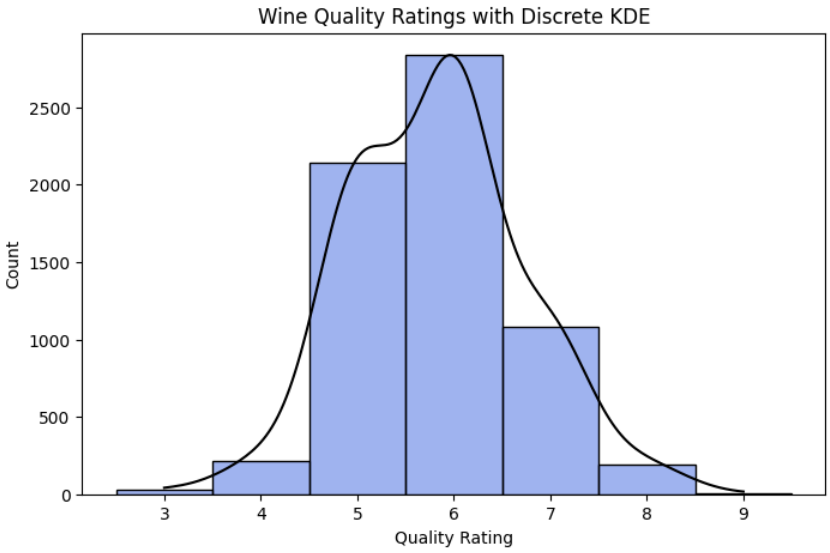

Figure 1: Distribution of wine quality scores

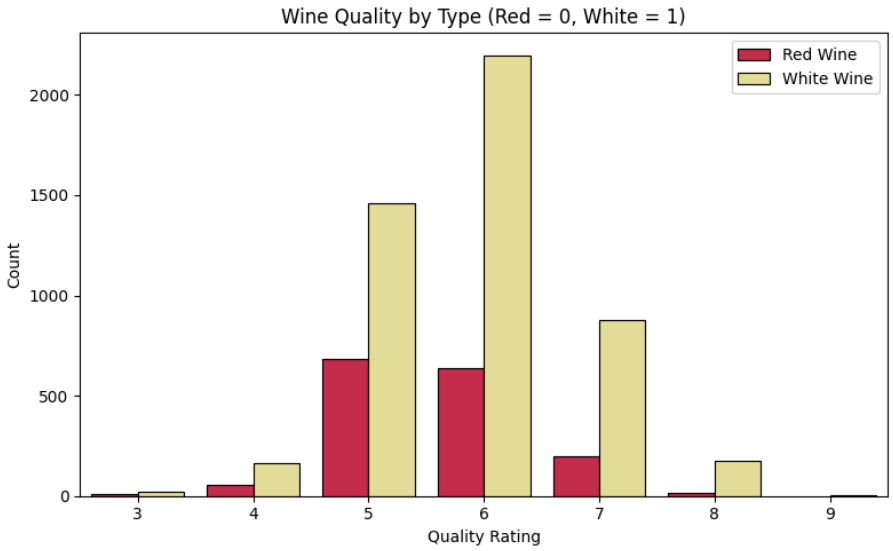

Figure 2: Comparison of wine quality by type (red vs white)

Feature Insights. A correlation heatmap was generated to examine linear relationships between physicochemical attributes and wine quality (Figure 3). Among all features, alcohol content showed the strongest positive correlation with quality ($r = 0.44$), suggesting that higher alcohol levels tend to be associated with better-rated wines. On the other hand, density ($r = -0.31$) and volatile acidity ($r = -0.27$) exhibited the strongest negative correlations, indicating that wines with lower density and lower levels of volatile acidity are more likely to receive higher quality scores. These insights provided useful guidance for model design and later interpretation of misclassified examples.

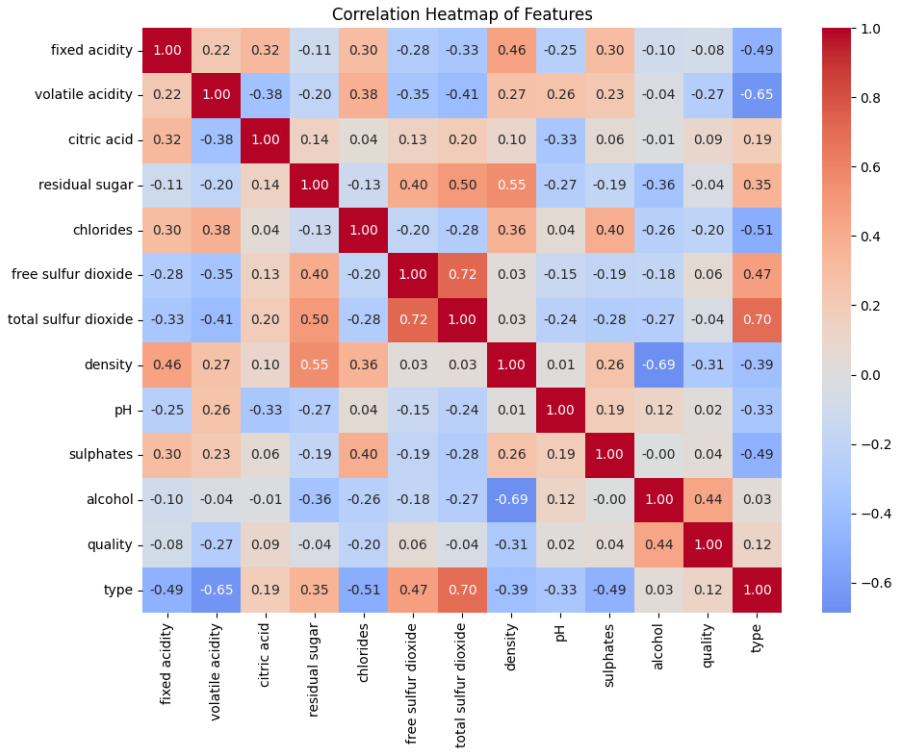

Figure 3: Pearson correlations between features and wine quality

1.2 Preprocessing

Each wine was assigned a type label — 0 for red and 1 for white. The quality score was transformed into a binary classification label:

- **Label = 1** for wine quality ≥ 6 (good wine)
- **Label = 0** for wine quality < 6 (not good)

This conversion resulted in a class distribution of about 63.3% good wines and 36.7% not-good wines, introducing moderate class imbalance. To address this, the **F1-score** was selected as the primary evaluation metric, balancing precision and recall more effectively than accuracy.

All input features were standardized using z-score normalization to ensure equal weighting across variables with different scales. Figure 4 illustrates the effect of scaling on a few representative features like `alcohol`, `pH`, and `sulphates`.

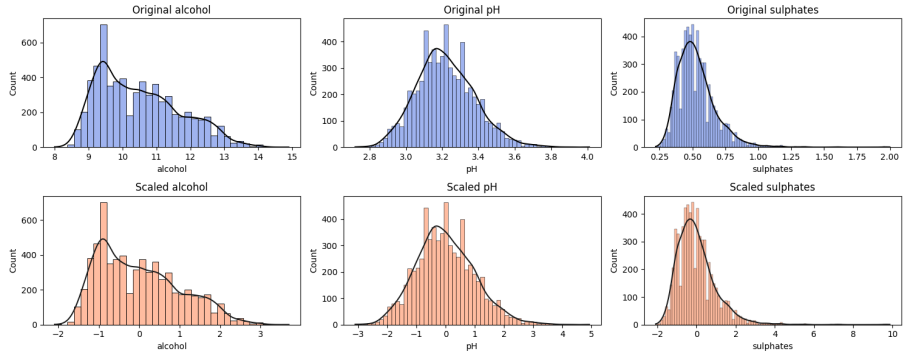

Figure 4: Comparison of raw vs. standardized features

The full dataset was then stratified into an 80/20 train-test split, preserving the class balance of good vs. not-good wines in both sets. Figure 5 confirms that the distributions remained consistent after the split.

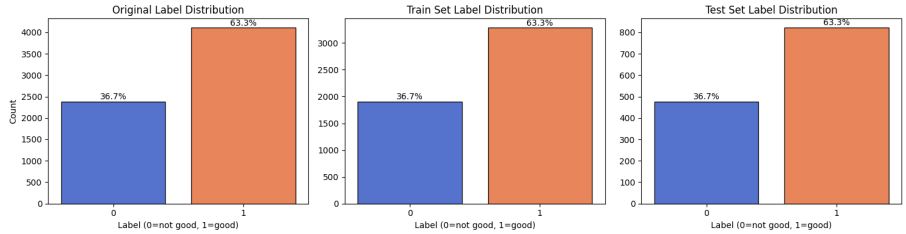

Figure 5: Class distribution before and after train–test split

Note on Data Splitting. Initially, 20% of the full dataset was held out using a stratified split to serve as an optional final test set. However, for consistency and a more comprehensive evaluation of model generalization, the final results reported in this work are based entirely on stratified k-fold cross-validation. This approach ensures that every data point contributes to both training and validation, mitigating the risk of biased results due to a single fixed split and better reflecting performance across the entire dataset. Additionally, data augmentation techniques were not applied in this project. Since the features are continuous physicochemical measurements derived from real chemical analysis, artificial perturbations could result in unrealistic or chemically invalid inputs.

2. SVM and Logistic Regression

2.1 Model Implementation and Hyperparameter Tuning

Logistic Regression

Logistic Regression is a linear classification algorithm that models the probability of a binary outcome by applying the logistic (sigmoid) function to a linear combination of input features. It estimates the likelihood that a given input belongs to a particular class, outputting probabilities between 0 and 1. These probabilities can then be thresholded—commonly at 0.5—to make discrete class predictions.

The logistic regression model was implemented from scratch and wrapped in a scikit-learn-compatible class to facilitate hyperparameter tuning. It uses L2 regularization and

optimizes the loss using batch gradient descent, meaning the weights are updated after processing the entire training dataset in each iteration. Key steps include:

- **Sigmoid Function:** The model uses a sigmoid activation function to map the raw linear combination of inputs to a probability score between 0 and 1. Numerical stability was ensured using clipping.

```
def _sigmoid(self, z):
    z = np.clip(z, -500, 500)
    return 1 / (1 + np.exp(-z))
```

- **Parameter Initialization and Training:** All weights (`w`) were initialized to zero. L2 regularization was implemented using a regularization strength parameter $\lambda = \frac{1}{C}$, where C is a tunable hyperparameter. For each training sample, gradients of the binary cross-entropy loss with regularization were used to update the weights and bias.

```
grad_w = x_i.T * (p_i - y_i) + lambda_reg * self.w
grad_b = p_i - y_i
```

- **Loss Function:** Binary cross-entropy loss was computed across all samples in each epoch, including the L2 penalty to discourage large weights.

```
loss = -np.mean(y * np.log(prob + 1e-15) +
                (1 - y) * np.log(1 - prob + 1e-15))
loss += 0.5 * lambda_reg * np.sum(self.w ** 2)
```

- **Prediction:** After training, the model computes probabilities using the sigmoid function, which are then thresholded at 0.5 to obtain final binary class predictions.

```
def predict(self, X):
    return (self.predict_proba(X) >= 0.5).astype(int)
```

Hyperparameters in LR. The model exposes the following hyperparameters:

- `lr` (learning rate): Controls the size of each parameter update. Higher value leads to faster convergence but risks overshooting; a lower value ensures stability but slower learning.
- `C`: Inverse regularization strength, with $\lambda = 1/C$. Larger values reduce regularization (allowing more complex models), while smaller ones enforce stronger regularization to prevent overfitting.
- `n_iter`: Number of training epochs, set to 1000 as a reliable upper limit for convergence.

Hyperparameter Tuning. Hyperparameters were selected using a 5-fold grid search (via `GridSearchCV`) over the following grid:

```
lr:  [0.001, 0.01, 0.1, 1]      C: [1, 10, 200, 300]
```

Performance was evaluated using the F1 score to account for class imbalance in the binary classification task. The best configuration was `lr = 0.1`, `C = 200`, with an average validation F1 of 0.8066. As shown in Figure 6, performance improved at moderate-to-large C values and higher learning rates.

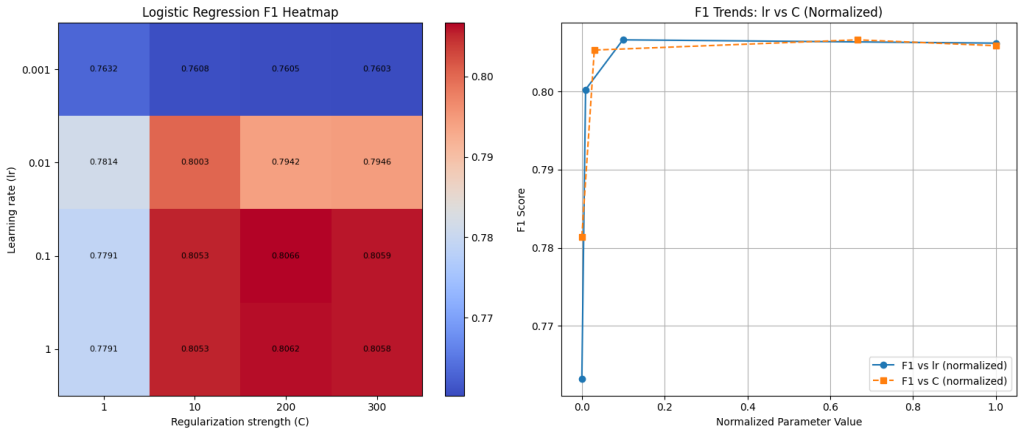

Figure 6: Grid search heatmap of F1 scores across learning rate and regularization values for the Logistic Regression model.

Support Vector Machine

Support Vector Machine (SVM) is a supervised classification algorithm that seeks to find the optimal hyperplane which best separates data points of two classes. This hyperplane maximizes the margin, defined as the distance between the hyperplane and the closest data points from each class, known as support vectors.

The SVM model was implemented from scratch and wrapped in a scikit-learn-compatible class to facilitate hyperparameter tuning. It is a linear classifier trained using stochastic gradient descent, updating the weights per training sample to minimize hinge loss with L2 regularization. Key steps include:

- **Label Encoding:** Target labels were converted from {0, 1} to {-1, 1}, which is required for computing hinge loss.

  ```
  y_ = np.where(y <= 0, -1, 1)
  ```

- **Initialization:** All weights and bias were initialized to zero.

  ```
  self.w = np.zeros(n_features)
  self.b = 0
  ```

- **Hinge Loss and Gradient Updates:** During each epoch, the model iterated over all training samples. If a sample violated the margin constraint, the following per-sample updates were applied:

```
if margin < 1:
    grad_w = self.w / self.C - y_[i] * X[i]
    grad_b = -y_[i]
else:
    grad_w = self.w / self.C
    grad_b = 0
```

- **Prediction:** Predictions were based on the sign of the decision function.

```
return np.where(self.project(X) >= 0, 1, 0)
```

Hyperparameters in SVM.

- `lr`: Learning rate for parameter updates.
- `C`: Inverse regularization strength. Smaller values increase the margin but allow more violations; larger values prioritize strict separation.
- `n_iter`: Number of training iterations (epochs).

Hyperparameter Tuning. Hyperparameter tuning was conducted using 3-fold cross-validation via `GridSearchCV` to select the best values of `lr` and `C` based on the F1-score. The number of training iterations was set at 100 due to time constraints and for reproducibility of results. In a less restricted setting, higher values (e.g. 1000 or more) and a higher number of folds would likely improve performance through better convergence.

```
lr:  [0.0001, 0.001, 0.01]      C: [1, 10, 100, 150]
```

The best performing combination was `lr = 0.0001` and `C = 100`, achieving an average F1 score of 0.8149. As shown in Figure 7, small learning rates and high C values yielded consistent improvements.

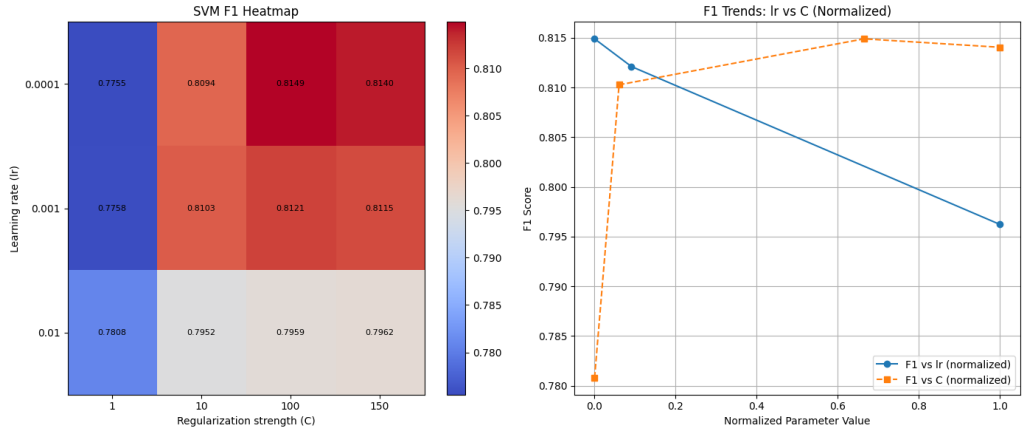

Figure 7: Grid search heatmap of F1 scores across learning rate and regularization values for the SVM model.

2.2 Training and Evaluation

To systematically assess model performance, both the Logistic Regression (LR) and Support Vector Machine (SVM) classifiers were trained and evaluated using stratified 5-fold cross-validation. The evaluation focused on classification accuracy and F1-score, which were appropriate metrics given the binary nature of the task and potential class imbalance. Additional metrics such as precision, recall, confusion matrices, and training time were also tracked for each fold.

Training Setup and Metrics. Both models were tuned using grid search prior to evaluation. The number of training iterations for both models was fixed at 1000 for computational feasibility and reproducibility. The evaluation pipeline was implemented via a `train_eval_cv` function. For each fold, models were trained on 80% of the data and validated on the remaining 20%, ensuring all samples contributed to both training and validation. This function computed:

- Final training and validation loss (log-loss or hinge-loss depending on model);
- Accuracy, precision, recall, and F1-score on validation data;
- Per-epoch training curves (loss and accuracy);
- Training duration and fold-level misclassification summaries.

2.2.1 Learning Behavior and Convergence.

The training curves (Figure 8) show that both models reached stable convergence within the 100-epoch limit:

- **Logistic Regression (LR):** The training loss decreased from 0.6931 to 0.5204 (24.9% drop), with accuracy rising from 63.3% to 74.1%. Noticeably, there was no significant change in either loss or accuracy during the final five epochs, indicating convergence;
- **Support Vector Machine (SVM):** Training loss dropped from 0.8951 to 0.5992 (33.1% reduction), while accuracy improved from 68.3% to 74.8%. Similarly, the curve flattened in the final epochs, suggesting stable training behavior.

Model Comparison. While both models achieved comparable final training accuracy ($\sim 74\%$), their learning dynamics differed:

- The LR model exhibited a smoother, more consistent improvement in both loss and accuracy, with a larger relative gain in accuracy over the training period (+17.05%).
- The SVM model had slightly better final accuracy and more aggressive loss reduction, although its accuracy gain was more modest (+9.5%).

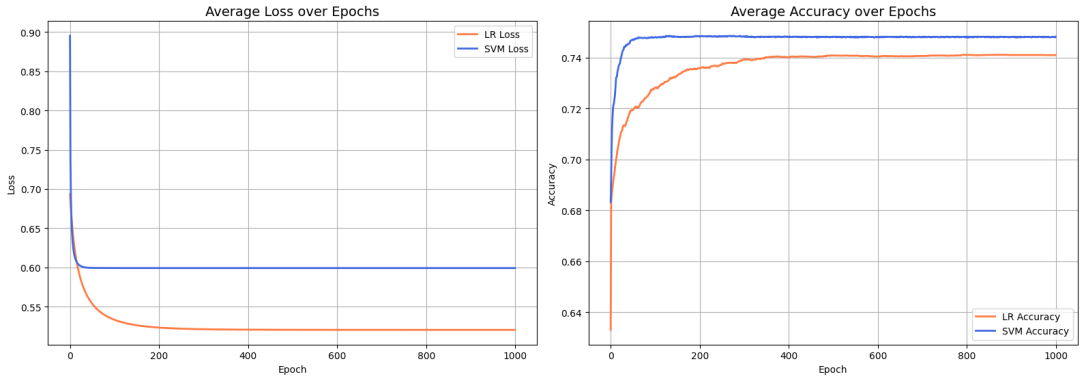

Figure 8: Training loss and accuracy per epoch, averaged across 5 folds.

2.2.2 Validation Performance

Key performance metrics - including validation accuracy, precision, recall, and F1-score - were recorded for each fold and averaged to capture overall trends. A visual summary is shown in Figure 9.

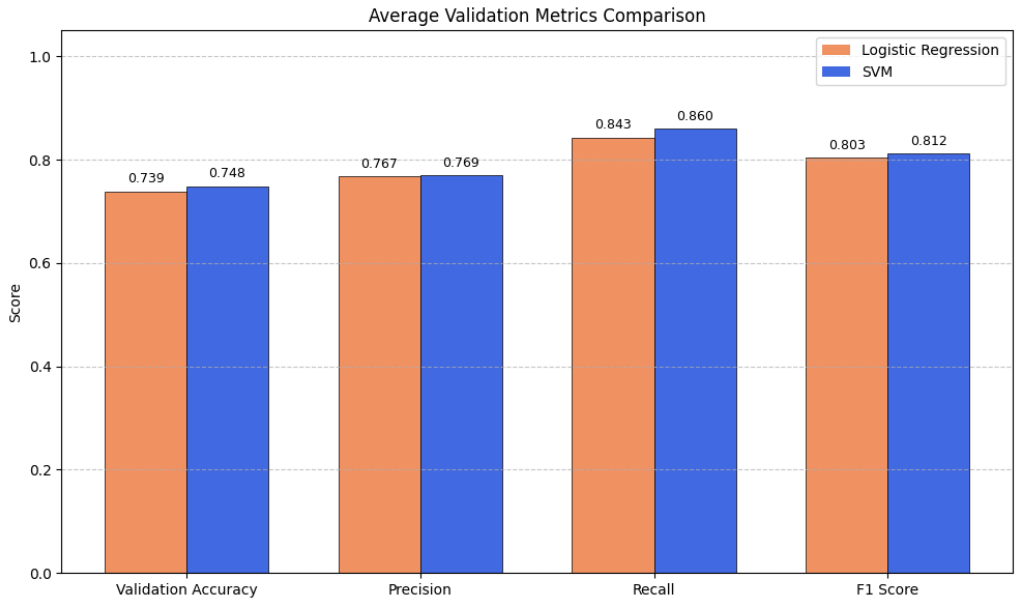

Figure 9: Average cross-validation metrics comparison between Logistic Regression and SVM. Bars reflect mean validation accuracy, precision, recall, and F1-score across 5 folds.

Validation Accuracy. SVM achieved a slightly higher average validation accuracy (74.80%) compared to Logistic Regression (73.88%). While the difference may seem modest ($\approx 0.92\%$), it remained consistent across all folds, indicating stronger predictive performance on unseen data. Both models showed small standard deviations in validation accuracy (~ 0.012), suggesting stable generalization across folds.

Precision. In terms of precision, SVM again slightly outperformed LR with an average of 0.7691 versus 0.7673. The difference, though minor, indicates that SVM predictions were marginally more likely to be correct when labeling a sample as "good" (positive class).

Recall. Recall scores were particularly strong for both models, with SVM achieving 0.8601 and LR 0.8431. These high values suggest that both models effectively captured the majority of positive cases, but SVM's edge here is notable. A higher recall implies fewer wines were missed, which is advantageous in contexts where failing to detect a true positive is costly.

F1 Score. F1-score encapsulates the overall balance between false positives and false negatives. SVM obtained a higher average F1-score (0.8120) than LR (0.8033), reinforcing its slightly better performance across both precision and recall. This indicates that SVM achieved a more favorable tradeoff between the two, making it more reliable overall.

Summary. While the performance gap between the models is relatively narrow, SVM consistently led across all validation metrics. These improvements suggest that SVM offers stronger classification quality - particularly in recall, where it surpasses LR by more than 1.7%. However, this improvement comes at the cost of significantly longer training time (over 50× slower), making Logistic Regression a more efficient choice in scenarios with strict time or resource constraints.

2.2.3 Misclassification and Class Balance Analysis

To better understand model behavior beyond overall accuracy, the average confusion matrices and class-level metrics produced during 5-fold cross-validation were analyzed. These reveal important differences in how each model performs across the two classes (good vs bad wines), especially with regard to misclassification patterns and biases.

Model Confusion Patterns and Class Imbalance Impact. As shown in Figure 10, both models demonstrate a clear asymmetry in class-wise performance.

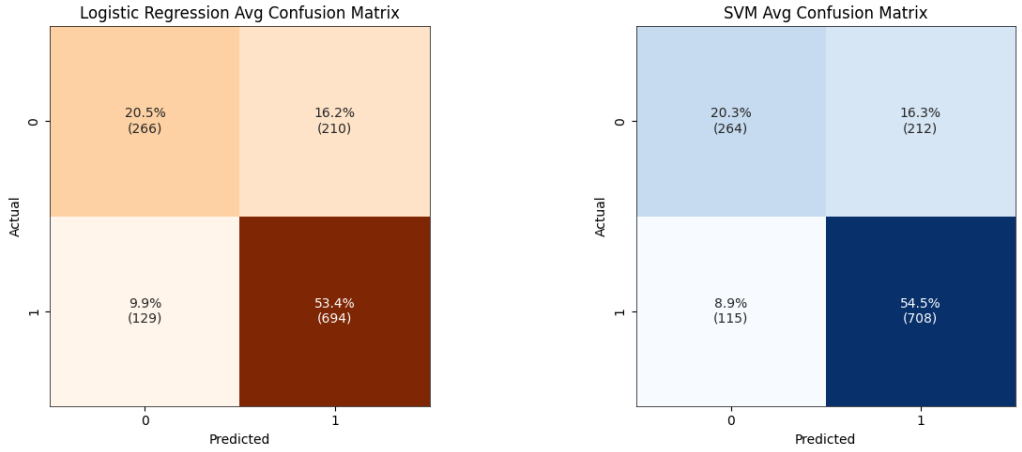

Figure 10: Average confusion matrix comparison between Logistic Regression and SVM.

This performance gap highlights a strong tendency to misclassify bad wines as good. The likely cause is a combination of class imbalance and overlapping feature distributions, which disproportionately favors correct prediction of the dominant or more separable class (in this case, good-quality wines).

Precision, Recall, and F1 Score Insights. From the average classification reports:

- **SVM** achieved higher precision, recall, and F1 across both classes:
  - Class 0 (bad wines): precision = 0.697, recall = 0.555, F1 = 0.618
  - Class 1 (good wines): precision = 0.769, recall = 0.860, F1 = 0.812
- **Logistic Regression** was slightly weaker, particularly in recall for class 0:
  - Class 0: precision = 0.674, recall = 0.559, F1 = 0.611
  - Class 1: precision = 0.767, recall = 0.843, F1 = 0.803

Macro-averaged metrics (which weight both classes equally) further support the advantage of SVM: macro F1 = 0.715 vs 0.707 for LR. This suggests SVM provides more balanced classification, even though the difference is modest.

Feature Difference Analysis To explore what features may contribute to model decisions and misclassifications, the mean feature values were analyzed for correctly and incorrectly classified instances. Specifically, for every feature, was computed:

$$\text{Diff} = \mu_{\text{misclassified}} - \mu_{\text{correctly classified}}$$

This value captures the average difference in feature representation between errors and correct predictions. A larger absolute difference implies that the feature has a stronger influence on classification decisions.

The most influential of the patterns, shown in Figure 11 are summarized below:

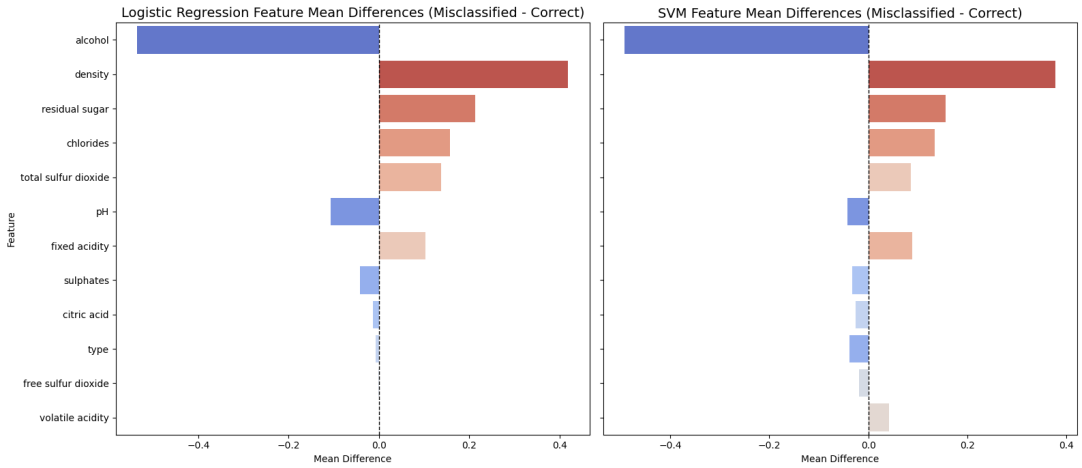

Figure 11: Mean feature value differences between correctly and incorrectly classified instances for each model.

- **Alcohol content** had the strongest discriminative power in both models:
  - Misclassified wines had significantly lower alcohol (˜-0.40), while correctly classified wines had much higher values (˜+0.14).
  - This suggests models heavily rely on alcohol to identify good wines, which is aligned with real-world wine quality indicators.

- **Density** and **residual sugar** also showed consistent trends:
  - Misclassified wines had higher density and sugar - typically associated with lower quality - yet the model may have underweighted these when conflicting with higher alcohol.
- Other relevant features include **chlorides**, **total sulfur dioxide**, and **pH**, but their influence was subtler.

Interestingly, SVM seemed to capture these patterns with slightly more nuance. The absolute difference between feature means for misclassified vs correct samples was generally smaller in SVM, suggesting smoother decision boundaries and possibly better generalization. This also aligns with the higher class 1 recall observed.

2.2.4 Overfitting and Generalization Analysis

To investigate the extent of overfitting, the accuracy gap between training and validation performance was examined across the five cross-validation folds.

For **Logistic Regression**, the mean overfitting gap was 0.0022, with a standard deviation of 0.0148, indicating very low bias and moderate variance. The minimum and maximum gaps were -0.0159 and 0.0211, respectively. The slightly negative values in some folds suggest cases where validation performance marginally exceeded training performance (potential underfitting), while the positive values point to mild overfitting. Nevertheless, the small mean and narrow spread demonstrate that logistic regression generalized reliably, with minimal divergence between training and validation results.

For the **Support Vector Machine (SVM)**, the mean overfitting gap was nearly zero at 0.0002, with a standard deviation of 0.0151. The gap ranged from -0.0224 to 0.0168, similarly showing a balance between slight over- and underfitting across folds. The extremely low bias estimate (0.0002) and comparable variance estimate (0.0121) further confirm that SVM maintained consistent generalization behavior throughout training.

These trends are visually represented in Figure 12, which displays the accuracy gap for each model across folds. Both models exhibit values close to zero, affirming their resistance to significant overfitting.

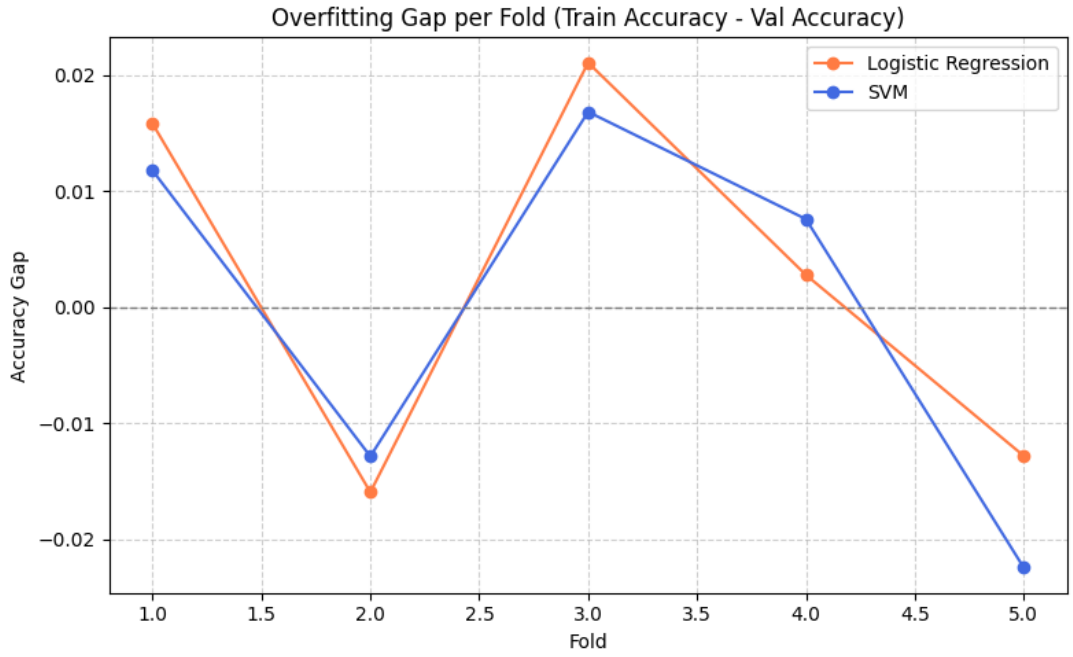

Figure 12: Training–Validation Accuracy Gap Across 5 Cross-Validation Folds for Logistic Regression and SVM.

3. Kernelized Models

3.1 Motivation and Implementation

While linear models such as logistic regression and linear SVM can be effective, they assume that the data is linearly separable in its original feature space. In many real-world cases, this assumption does not hold. To overcome this limitation, both models were extended to their kernelized counterparts, allowing them to model non-linear relationships by implicitly projecting the data into a higher-dimensional feature space.

Kernel methods work by replacing the inner product $\langle \mathbf{x}_i, \mathbf{x}_j \rangle$ in the learning algorithm with a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$, which computes the inner product in some feature space without explicitly performing the transformation. This technique is commonly referred to as the "kernel trick". Support was implemented for the following kernel functions:

- **Linear:** $K(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z}$
- **Polynomial:** $K(\mathbf{x}, \mathbf{z}) = (\gamma \cdot \mathbf{x}^\top \mathbf{z} + c_0)^d$
- **Radial Basis Function (RBF):** $K(\mathbf{x}, \mathbf{z}) = \exp(-\gamma \cdot \|\mathbf{x} - \mathbf{z}\|^2)$

Kernel Logistic Regression

The kernel logistic regression model was implemented from scratch and wrapped in a scikit-learn-compatible class for hyperparameter tuning. It optimizes a regularized logistic loss computed over the entire kernel matrix using batch gradient descent, updating parameters after processing all samples in each iteration. Key steps include:

- **Kernel Function:** The model supports three kernel types:
  - `linear`: Performs standard dot product $K(x, y) = x^\top y$.
  - `rbf (Radial Basis Function)`: Computes similarity using a Gaussian:

    ```
    K = exp(-gamma * ||x - y||^2)
    ```

  - `poly (Polynomial)`: Maps inputs into higher-degree polynomial space:

    ```
    K = (gamma * x.T @ y + coef0) ** degree
    ```

- **Training:** A dual formulation is used, optimizing coefficients α over the kernelized input space. A regularization term controlled by parameter C is applied to prevent overfitting.

  ```
  f = self.K.dot(self.alpha)
  prob = self._sigmoid(f)
  grad = self.K.T.dot(prob - y) / n_samples + lambda_reg * self.alpha
  self.alpha -= self.lr * grad
  ```

- **Loss Function:** Binary cross-entropy is computed on the predicted probabilities derived from the sigmoid-transformed kernel output $f = K\alpha$. An L2 penalty on α is added to encourage smaller coefficients.

  ```
  loss = -np.mean(y * np.log(prob + 1e-15) +
                  (1 - y) * np.log(1 - prob + 1e-15))
  loss += 0.5 * lambda_reg * np.sum(self.alpha ** 2)
  ```

- **Prediction:** After training, the model computes kernel similarities between new data and training examples, followed by a sigmoid transformation.

  ```
  def project(self, X):
      K = self._kernel(X, self.X_train)
      return np.dot(K, self.alpha * self.y_train) + self.b
  ```

Hyperparameters in KLR. The following hyperparameters were made tunable:

- `kernel`: One of `['linear', 'rbf', 'poly']`.
- `C`: Inverse regularization strength.
- `gamma`: Spread of RBF or polynomial kernel.
- `degree`: Degree of the polynomial kernel.
- `coef0`: Bias term added in polynomial kernel.
- `lr`: Learning rate for gradient descent.
- `n_iter`: Number of training iterations.

Although only the RBF kernel was used in final tuning, the design supports flexible experimentation with both polynomial and linear kernels.

Hyperparameter Tuning. To select the optimal combination of regularization strength (C) and kernel scale (gamma), a grid search was conducted using sklearn's `GridSearchCV` with 2-fold cross-validation due to computational limits. Performance was evaluated using the F1 score to account for class imbalance in the binary classification task.

The following hyperparameter space was explored for the kernelized logistic regression model:

`C: [50, 100]` `gamma: [scale, 0.01]`

The learning rate (`lr`) and the number of iterations (`n_iter`) were fixed at `0.01` and `100` respectively. These values were chosen based on prior experimentation: lower learning rates led to very slow convergence, while higher values caused instability; the number of training iterations was set at 100 due to time constraints and for reproducibility of results. In a less restricted setting, higher values would likely improve performance through better convergence.

The value `scale` for `gamma` is a heuristic that sets $\gamma = \frac{1}{d \cdot \mathrm{Var}(X)}$, where d is the number of features and $\mathrm{Var}(X)$ is the input variance. This provides a reasonable default kernel width that adapts to the dataset scale.

The best model was obtained with `C = 50` and `gamma = scale`, achieving an F1 score of 0.7845 on the validation folds. A slightly lower score of 0.7842 was observed with `C = 100`, showing minimal sensitivity to this parameter in the tested range. However, performance dropped significantly when `gamma = 0.01`, indicating that too large a kernel bandwidth led to underfitting.

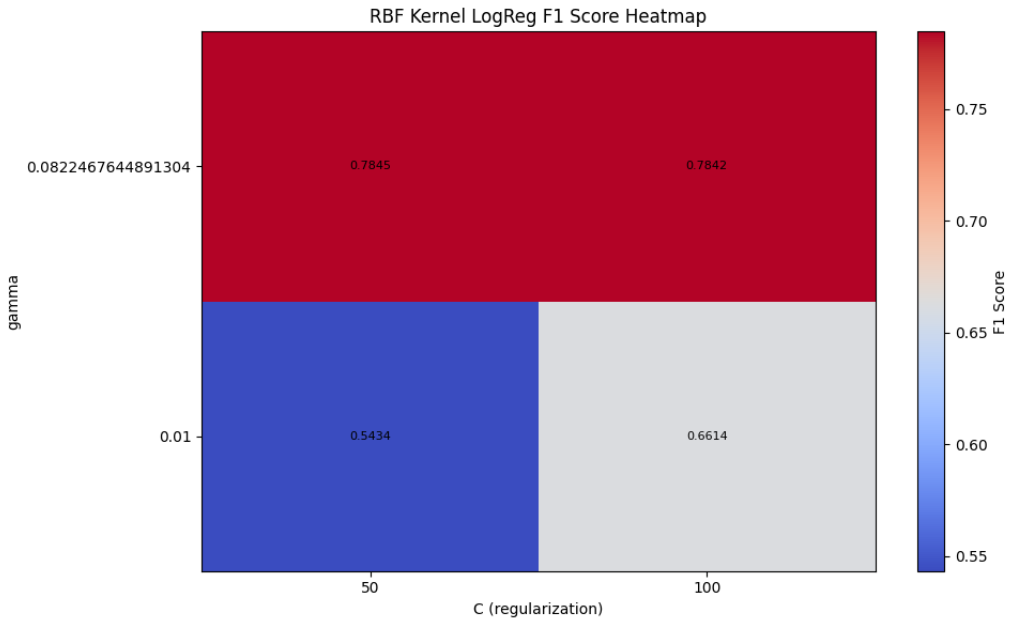

Figure 13: Grid search heatmap of F1 scores across spread and regularization values for the RBF-Kernel LR model.

Kernel Support Vector Machine

The Kernel SVM is a nonlinear extension of the standard linear SVM, implemented from scratch and wrapped for scikit-learn compatibility. It solves the dual optimization problem with stochastic updates on dual coefficients α, iterating over samples and updating coefficients individually based on hinge loss and regularization. Key steps include:

- **Kernel Function:** The same RBF kernel implementation was used as in the logistic regression kernel model.
- **Training:** Training was performed using stochastic gradient descent over the dual form. For each iteration, the model updates each coefficient α_i based on the margin violation:

```
margin = y_i * (dot(alpha * y, K[:, i]) + b)
if margin < 1:
    alpha[i] -= lr * (grad_i - 1)
    b += lr * y_i
```

- **Loss Function:** Total loss was computed as the sum of hinge loss and regularization over the kernelized coefficients:

```
reg = (1 / (2 * C)) * np.dot(alpha * y, K @ (alpha * y))
total_loss = reg + mean(hinge_losses)
```

- **Prediction:** Class predictions are made using the sign of the projected kernel decision function:

```
def predict(self, X):
    return np.where(self.project(X) >= 0, 1, 0)
```

Hyperparameters in KSVM. The model exposes the following hyperparameters:

- `kernel`: One of `['linear', 'rbf', 'poly']`.
- `C`: Inverse of regularization strength.
- `gamma`: Kernel scale.
- `degree`: Degree of polynomial kernel.
- `coef0`: Bias term added in polynomial kernel.
- `learning_rate`: Step size for gradient descent.
- `n_iter`: Total number of training iterations.

Though only RBF was explored in final tuning due to time constraints, the implementation fully supports polynomial and linear kernels, allowing flexibility for future experimentation.

Hyperparameter Tuning. For the kernelized SVM model, the following hyperparameter grid was evaluated:

`gamma: [scale, 0.01, 0.001]` `learning_rate: [0.001, 0.01, 0.1]`

Here, both `C` and `n_iter` were fixed to `100`, based on previous tuning results with the linear SVM model. Fixing `C` allowed the grid search to focus exclusively on the kernel and optimization dynamics, as this regularization value was already shown to perform well across various settings. The number of training iterations was again fixed for consistency.

Again, the `scale` option was used to automatically set the gamma value proportionally to feature space variance.

The best configuration achieved an F1 score of 0.8189 with `gamma = scale` and `learning_rate = 0.01`. Other combinations yielded scores between 0.800 and 0.815, indicating a degree of robustness to these hyperparameters. However, very low values such as `gamma = 0.001` and `lr = 0.001` led to a notable performance drop (F1 = 0.7700), suggesting underfitting and slow convergence.

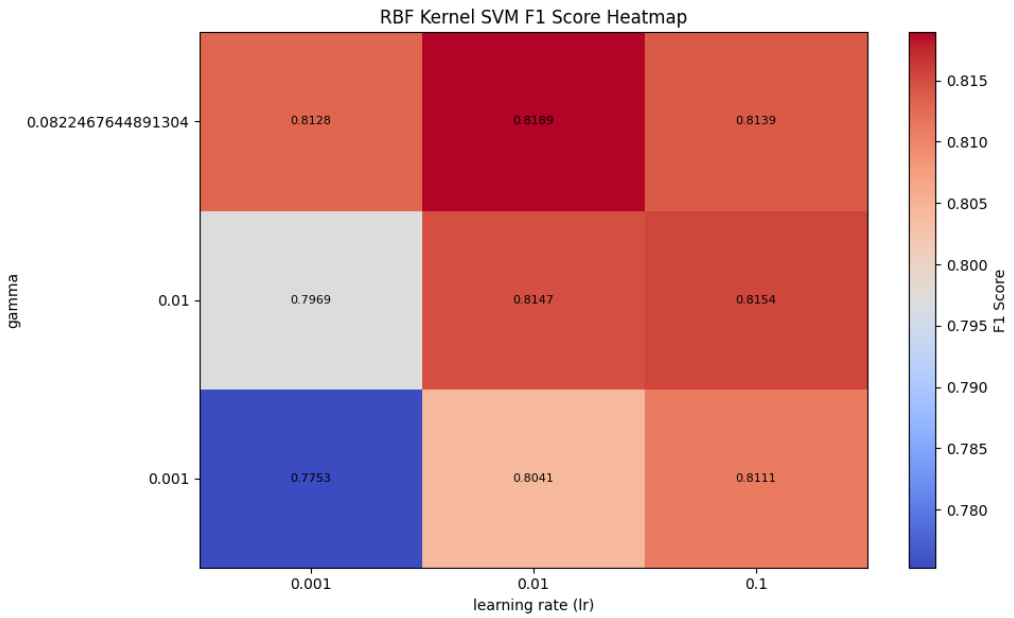

Figure 14: Grid search heatmap of F1 scores across spread and learning rate for the RBF-Kernel SVM model.

3.2 Training and evaluation of Kernelized Models

To evaluate the training dynamics of the kernelized models, the same `train_eval_cv` function was used as with the linear classifiers. Due to the substantially higher training time of the kernel methods, especially kernelized logistic regression, 3-fold cross-validation ($k = 3$) was used instead of 5-fold to reduce total computation time while still ensuring generalizability.

3.2.1 Learning Behavior and Convergence

Kernel Logistic Regression (KLR): The average training and validation accuracies across folds were approximately 71.1% and 71.9% respectively. The training curve, averaged over all folds, exhibited a smooth and stable convergence:

- **Train Loss:** Decreased from an initial value of 0.6932 to a final value of 0.5501. The minimum loss reached was 0.5501, representing a total decrease of 20.64%. The final 3-epoch change was minimal (0.0012), indicating convergence.
- **Train Accuracy:** Improved from 63.31% to 71.72%, yielding a gain of +13.29%. The final 5-epoch change was just 0.0004, showing that learning had stabilized.

Kernel Support Vector Machine (KSVM): The kernel SVM achieved higher training accuracy, averaging 80.0% across folds, with validation accuracy close behind at approximately 77.4%. However, its loss curve showed instability despite a stable accuracy trend:

- **Train Loss:** Started at 0.7274 but unexpectedly increased to 2.4056. The lowest loss recorded was 0.6977. The negative drop of -230.70% indicates that the regularization loss began to dominate as support vectors became too influential. The final 3-epoch change (0.0790) suggests that optimization was still ongoing.
- **Train Accuracy:** Increased significantly from 63.64% to 80.00%, for a total gain of +25.71%. Despite the unstable loss, accuracy stabilized with minimal final variation (0.0002), suggesting a well-fitted margin.

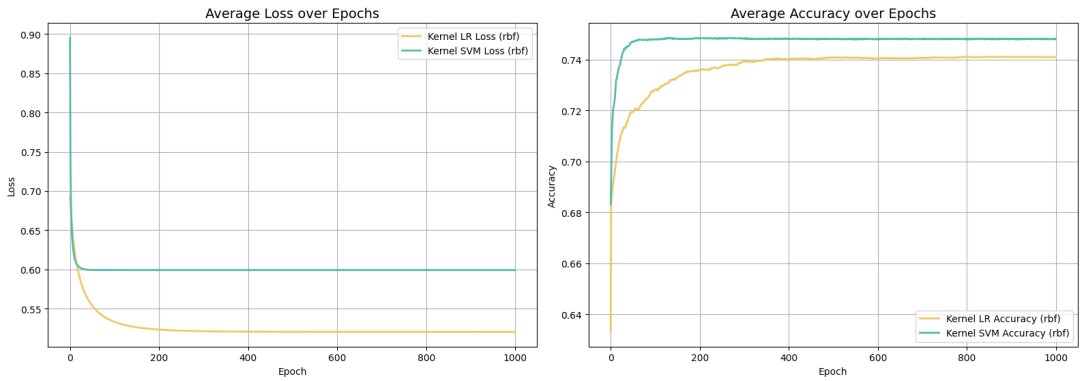

Figure 15: Per-epoch training curves of Kernel Logistic Regression (left) and Kernel SVM (right), averaged over 3-folds.

Comparison with Standard Models Compared to their linear counterparts, both kernelized models showed improved fitting on the training data:

- The linear logistic regression model plateaued at a training accuracy of around 70.1%, slightly below the 71.7% achieved by the kernelized variant.

- The kernel SVM achieved a much stronger training fit (80% vs. 73.4% for the linear SVM), demonstrating the benefit of the RBF kernel in capturing nonlinear patterns.

However, the stability of the training loss was better in the linear models. The instability in the KSVM loss suggests that while kernelization allows for more complex decision boundaries, it also introduces optimization challenges that require careful tuning.

3.2.2 Comparison of Validation Metrics Across All Models

Key performance metrics—including validation accuracy, precision, recall, and F1-score—were recorded for each fold and averaged to capture overall trends. A visual summary is shown in Figure 16.

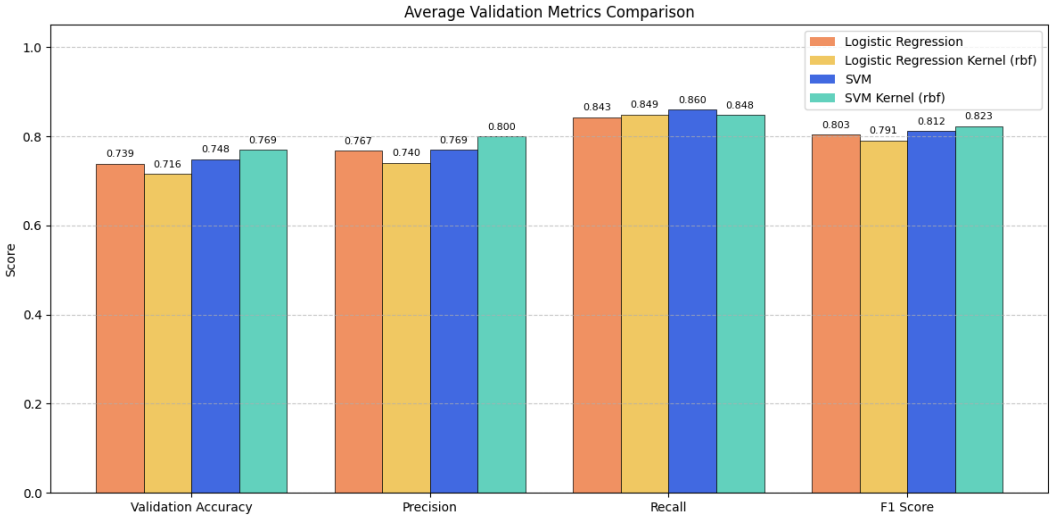

Figure 16: Average cross-validation metrics comparison between linear and kernelized models. Bars reflect mean validation accuracy, precision, recall, and F1-score.

Validation Accuracy. Among all models, the Kernel SVM (KSVM) achieved the highest validation accuracy (76.91%), followed by standard SVM (74.80%), Logistic Regression (73.88%), and finally Kernel Logistic Regression (KLR) at (71.16%). This trend suggests that kernelization significantly boosted SVM's predictive performance, while it provided limited gain for logistic regression.

Precision. KSVM also led in precision, with an average of 0.7997, outperforming both the standard SVM (0.7691) and Logistic Regression (0.7673). Interestingly, KLR showed slightly lower precision (0.7667), suggesting it was more prone to false positives than its counterparts. This again supports the superiority of the RBF kernel in the SVM setting compared to its impact on logistic regression.

Recall. All models achieved high recall values, but standard SVM achieved the highest average recall at 0.8601, slightly outperforming the kernelized version (0.8478) and Logistic Regression (0.8431). Kernel Logistic Regression lagged behind at 0.7965. This suggests that while KSVM improved other metrics, linear SVM remained the most effective at capturing true positives.

F1 Score. F1-score balances both precision and recall, and again KSVM led with 0.8230, ahead of standard SVM (0.8120), LR (0.8033), and KLR (0.7814). This confirms that kernelized SVM offers the best overall trade-off between minimizing false positives and capturing true positives. The marginal F1 gain from linear SVM to KSVM ($\sim 1.1\%$) further illustrates the benefits of non-linear decision boundaries in complex data settings.

Summary. In short, SVMs—especially the kernelized variant—were the most effective classifiers in this study, with KSVM achieving the highest validation accuracy and F1-score. However, these gains came at the cost of significantly longer training time (over 70× slower than LR), making linear SVM still appealing when interpretability or efficiency is prioritized, given that it also achieved the best recall, making it slightly better at minimizing false negatives. Logistic Regression—while lightweight and fast—showed weaker results, particularly in recall. KLR, although kernelized, underperformed compared to its linear counterpart, suggesting that non-linear transformation alone may not always guarantee improvement without sufficient optimization or regularization.

3.2.3 Misclassification and Class Balance Analysis

To complement the previous evaluation, the average confusion matrices for both RBF Kernel Logistic Regression (KLR) and RBF Kernel SVM (KSVM) were plotted to better understand class-level performance across folds. The visualizations in Figure 17 highlight notable differences in how each model handles good versus bad wine predictions.

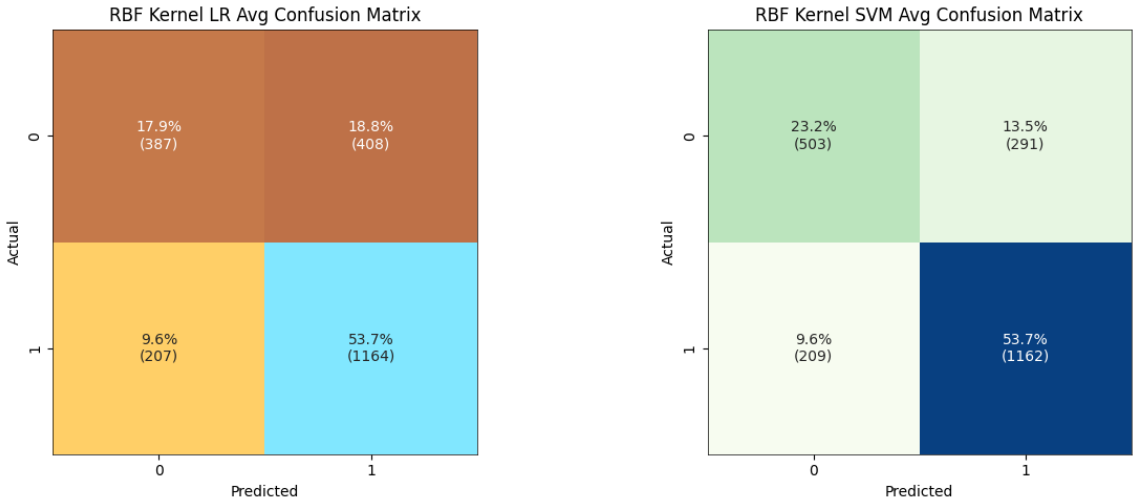

Figure 17: Average confusion matrices across 5 folds for RBF Kernel Logistic Regression (left) and RBF Kernel SVM (right). Each cell shows the percentage of predictions along with absolute counts.

Summary. Looking at the left matrix (KLR), only **48.6%** of class 0 (bad wines) were correctly classified — a steep drop compared to **63.4%** in the KSVM model on the right. This large improvement in class 0 recall (nearly +15%) strongly suggests that the SVM's margin-based approach is better at detecting harder, minority-class cases.

Despite this, both kernel models continued the pattern seen earlier — strong preference for class 1 predictions. Over **84%** of good wines were correctly identified by both

models, with KSVM slightly ahead at **84.8%** compared to KLR's **84.9%**. However, KLR made significantly more false positives, evident from the large off-diagonal entry in the top-right of the left matrix.

Comparison with Standard Models Compared to their linear counterparts, the kernelized versions show modest but meaningful improvements in handling the minority class. While standard Logistic Regression only achieved **55.9%** recall for class 0, and linear SVM achieved **55.5%**, the KSVM clearly outperforms both with **63.4%**.

Classification Report Comparison. The detailed metrics further confirm these trends:

- **Kernel SVM (KSVM):**
  - Class 0 — precision: **0.707**, recall: **0.634**, F1: **0.668**
  - Class 1 — precision: **0.800**, recall: **0.848**, F1: **0.823**
  - Macro F1: **0.746**, Accuracy: **0.769**
- **Kernel Logistic Regression (KLR):**
  - Class 0 — precision: **0.652**, recall: **0.486**, F1: **0.557**
  - Class 1 — precision: **0.740**, recall: **0.849**, F1: **0.791**
  - Macro F1: **0.674**, Accuracy: **0.716**

Comparison with Standard Models. The best-performing standard model (linear SVM) had a macro F1 of **0.715** and accuracy of **0.765**. Kernel SVM slightly exceeds this with a macro F1 of **0.746** and accuracy **0.769**, confirming that kernelization adds value. KLR, meanwhile, lags behind even the linear SVM, suggesting that Logistic Regression benefits less from kernelization in this setup.

Feature Importance via Misclassification Analysis To further interpret the model behavior, feature-wise mean differences were calculated between misclassified and correctly classified samples. This analysis offers insights into which input variables most strongly influence decision boundaries.

The most influential of the patterns, shown in Figure 18 are summarized below:

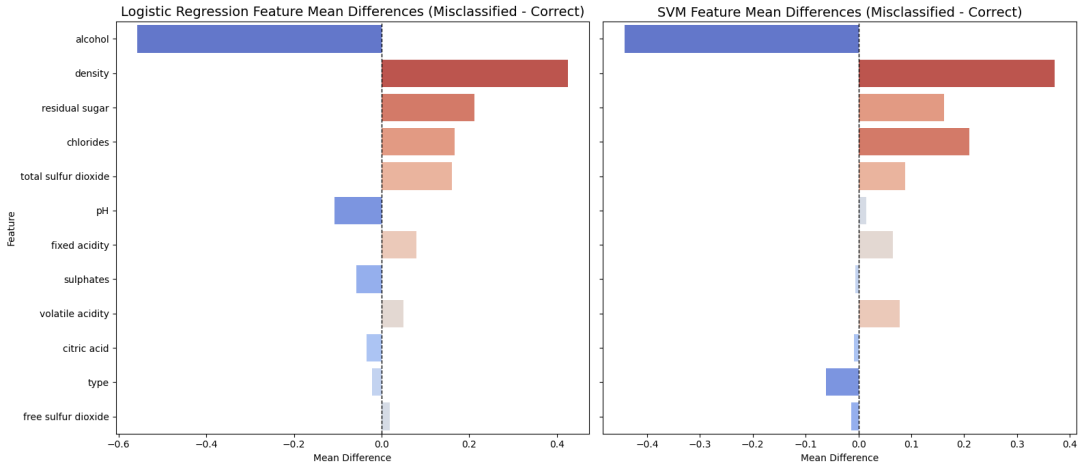

Figure 18: Mean feature value differences between correctly and incorrectly classified instances for each model.

- **Alcohol content** was the most discriminative feature for both models:
  - KLR: difference = **-0.557**
  - KSVM: difference = **-0.443**

  Misclassified wines tended to have lower alcohol levels, which aligns with lower perceived quality. Both models appeared to rely heavily on this signal.

- **Density** and **residual sugar** also exhibited large differences:
  - Higher in misclassified samples — consistent with confusion when high sugar or density occurs alongside high alcohol.

Summary. Interestingly, KSVM showed smaller magnitude differences across many features (e.g., sulphates, pH, type), indicating that its classification boundary may generalize better and rely less on extreme feature separations. KLR, on the other hand, showed larger swings in variables like alcohol and sugar — suggesting more binary decision tendencies.

Comparison with Standard Models. The standard Logistic Regression model had similar feature reliance, with alcohol and density again dominating the differences. However, compared to the kernel models, the standard versions showed even stronger feature swings — e.g., alcohol difference of over **-0.60** — which may signal sharper but less flexible decision boundaries.

3.2.4 Overfitting and Generalization Comparison.

To assess generalization and model complexity, the overfitting behavior of the kernelized models was analyzed via train–validation accuracy gaps, bias and variance estimates.

KLR showed virtually no overfitting, with an average accuracy gap of only **0.0018** and a standard deviation of **0.0115**. In fact, its minimum overfitting gap was slightly negative, indicating occasional cases where validation accuracy even surpassed training

accuracy. This, along with a very low bias estimate of **0.0018**, suggests that KLR is highly conservative — likely underfitting slightly — especially given its lower validation metrics and weaker minority-class performance. Its variance estimate (**0.0080**) and gap variance (**0.0001**) further support this stability, but also reflect limited capacity to capture complex patterns.

In contrast, the **RBF Kernel SVM** showed a higher mean overfitting gap of **0.0309**, with a comparable standard deviation of **0.0120**. Its gap remained positive across all folds (min = **0.0216**, max = **0.0477**), indicating consistent overfitting. This contrasts strongly with the standard SVM, whose mean gap was virtually zero. The kernelized version thus sacrifices some generalization in favor of higher model capacity, as also reflected in its stronger class-wise performance and feature learning depth.

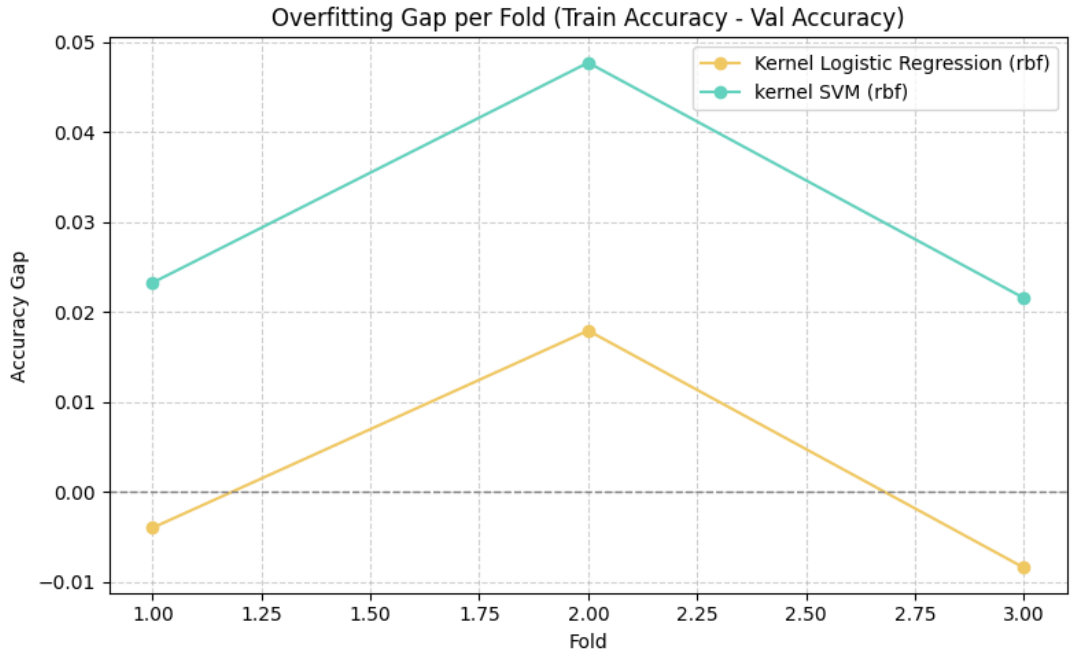

Figure 19: Training–Validation Accuracy Gap Across 5 Cross-Validation Folds for Kernel Logistic Regression and Kernel SVM.

Comparison with Standard Models. Overall, the kernelized SVM exhibits the most overfitting among all models, but it also achieved the highest validation performance, particularly for the minority class. Kernel logistic regression, while more flexible than its linear counterpart, retained the underfitting tendencies and conservative generalization profile of standard logistic regression. These comparisons highlight that the RBF kernel improved modeling capacity more significantly for SVM than for logistic regression, but at the cost of some overfitting.

4. Conclusion

This project involved implementing and evaluating two fundamental machine learning models: Support Vector Machine and Logistic Regression. The findings show that SVM slightly outperformed Logistic Regression in terms of test accuracy, likely due to its ability to maximize the decision boundary margin. However, Logistic Regression remained a

strong baseline, offering faster convergence. Overall, this comparative study highlights the importance of understanding model behavior, tuning, and preprocessing when solving classification tasks in practical settings.

Further Developments. Although an initial 20% of the data was reserved as a hold-out test set, the final experiments relied entirely on stratified k-fold cross-validation for both training and evaluation. This choice provided a more comprehensive and reliable view of model generalization. However, future work could involve training a final model on the entire training data and evaluating it on the unused test set to simulate deployment conditions and confirm generalization on truly unseen data.