SMO SVM + Kernel Extensions & In-Depth Analysis for Breast Cancer Classification

Abstract

Support Vector Machines (SVMs) combine mathematical rigor and practical performance, making them ideal for high-stakes applications like fraud detection and disease diagnosis. Built on convex optimization, SVMs ensure global optimality and robust generalization. In this project, we rederive and implement the Sequential Minimal Optimization (SMO) algorithm to solve the dual form of the soft-margin SVM. Our implementation supports both linear and nonlinear kernels (RBF and polynomial), providing flexible decision boundaries. We benchmark our solver against scikit-learn's SVC on the UCI Breast Cancer Wisconsin dataset and the Pima Indians Diabetes dataset. Hyperparameter tuning is performed using a custom 5-fold cross-validation routine, with diagnostics including support-vector counts, margin width, and dimensionality reduction via PCA. Our results show that a well-engineered SMO solver can match or surpass black-box implementations while offering full transparency—crucial for regulated environments like medical diagnostics, where interpretability and auditability are essential. This work highlights the potential for deploying SVMs in business contexts where performance, explanation, and compliance are key to better decisions and societal outcomes. [1] [2]

1 Description & Technical Details of the Machine Learning Method

Support Vector Machines (SVMs) are supervised learning models used for binary classification, aiming to find an optimal hyperplane that maximizes the margin between classes. This separation occurs in a high-dimensional feature space where data points are often more easily separable. SVMs are appealing due to their solid theoretical foundation in convex optimization, the kernel trick for nonlinear separations, and support vector sparsity, where only a subset of training examples (support vectors) impact the decision boundary. Geometrically, SVMs can be understood through convex hulls, where the optimal separating hyperplane bisects the shortest distance between the non-intersecting hulls, maximizing the margin.

1.1 Linear SVMs (Linearly Separable Case)

A Linear Support Vector Machine (SVM) seeks a hyperplane $\mathbf{w}^T\mathbf{x}+b=0$ that separates two classes with maximum margin. The prediction rule is: $\text{sign}(f(\mathbf{x}))$ where $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.

The margin M is defined as the distance between the closest points of each class and the hyperplane:

$$M = \frac{2}{\|\mathbf{w}\|}$$

For **Hard-Margin SVM** (perfectly linearly separable data), the SVM formulation minimizes the norm of the weight vector:

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2 \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \forall i$$

Soft-margin SVM

When perfect separation is not feasible (due to noise or overlap), slack variables $\zeta_i \geq 0$ are introduced to allow violations of the margin constraints:

$$\min_{\mathbf{w},b,\{\zeta_i\}} \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\zeta_i \quad \text{s.t.} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \zeta_i,\ \zeta_i \geq 0$$

Here, the parameter $C > 0$ is a regularization parameter that balances the trade-off between maximizing the margin and minimizing classification errors. A large C favors fewer margin violations (low bias), while small C allows a wider margin (better generalization).

Decision Function and Support Vectors: Once trained, the SVM prediction function becomes:

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

$$\mathbf{w} = \sum_{i\in\text{SV}} \alpha_i y_i \mathbf{x}_i$$

Only support vectors (with $\alpha_i > 0$) define the decision boundary, making the model efficient at inference time. SVMs are typically solved in their dual formulation using Lagrange multipliers, resulting in a quadratic programming (QP) problem. This allows the use of kernel functions to compute dot products in high-dimensional spaces efficiently, enabling the powerful *kernel trick*. We encapsulate these concepts in a Python class `SVM_SMO` (see jupyter notebook) that implements kernel computation, SMO optimization loops with analytic updates and bias adjustments, and prediction via the sign of the decision function.

1.2 Non-Linear SVMs and Kernel Methods

To handle more complex, non-linear decision boundaries, SVMs employ the **kernel trick**, enabling classification in higher-dimensional spaces without explicitly computing the mapping. Let $\phi(\mathbf{x})$ be a non-linear transformation that maps an input vector into a higher-dimensional feature space. Instead of explicitly computing $\phi(\mathbf{x})$, SVMs rely on a **kernel function** $K(\mathbf{x}_i, \mathbf{x}_j)$ defined as:

$$K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T\phi(\mathbf{x}_j)$$

This function directly computes the inner product in the transformed space. Since the optimization formulation only depends on inner products, this substitution avoids direct projection into the high-dimensional space — this is known as the **kernel trick**.

Kernel SVM Decision Function: With kernelization, the decision function becomes:

$$f(\mathbf{x}) = \sum_{i\in\text{SV}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

Here, α_i are the learned Lagrange multipliers for the support vectors $\mathbf{x}_i$. The model no longer depends on explicit weight vectors $\mathbf{w}$, but rather on the support vectors and their interactions via the kernel function. (since the weight vector $\mathbf{w}$ is not explicitly stored)

Commonly Used Kernels in SVMs

SVMs achieve non-linear classification by applying kernel functions that implicitly map input data into higher-dimensional spaces where linear separation is feasible.

- **Linear kernel:** $K(\mathbf{x}, \mathbf{x}') = \mathbf{x}^\top \mathbf{x}'$. Useful for high-dimensional, sparse datasets where classes are approximately linearly separable. It is computationally efficient and performs well in text classification and bioinformatics.
- **Polynomial kernel:** $K(\mathbf{x}, \mathbf{x}') = (\gamma\, \mathbf{x}^\top \mathbf{x}' + r)^d$. Captures interactions up to degree d and allows curved boundaries. Suitable when data has polynomial trends; parameters γ, r, and d control flexibility.
- **Gaussian RBF kernel:** $K(\mathbf{x}, \mathbf{x}') = \exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$. Creates localized decision regions; effective for non-linear problems with little prior on data structure. γ determines the kernel width.

In addition to these, **domain-specific kernels** have been developed (for example, **string kernels or spectrum kernels**) to handle structured data such as sequences, graphs, or text. These are especially powerful in text analytics and bioinformatics, where similarity is based on subsequences or other domain-driven patterns.

2 Optimization Problems for SVMs

Training a Support Vector Machine (SVM) involves solving a convex optimization problem, typically cast as a Quadratic Program (QP). This can be approached either in the primal form or the dual form, both of which ensure a global optimum due to the convexity of the objective and constraints.

2.1 Primal Form (Convex QP)

The primal soft-margin SVM problem for data $\{(\mathbf{x}_i, y_i)\}_{i=1}^n$ with $y_i \in \{+1, -1\}$ is:

$$\begin{aligned} \min_{\mathbf{w}, b, \{\zeta_i\}} \quad & \frac{1}{2}\|\mathbf{w}\|^2 + C \sum_{i=1}^{n} \zeta_i \\ \text{s.t.} \quad & y_i(\mathbf{w}^T \phi(\mathbf{x}_i) + b) \geq 1 - \zeta_i, \quad \zeta_i \geq 0 \end{aligned}$$

Here, $\phi(\mathbf{x})$ is the feature map (identity for linear SVM). This problem is convex since the objective and constraints define a convex feasible region. It's usually solved directly for low-dimensional data or with gradient methods.

Hinge Loss View: An equivalent formulation using the hinge loss function is:

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

The term $\max(0, 1 - y_i f(\mathbf{x}_i))$ is the hinge loss: zero if the margin is satisfied, otherwise linearly increasing.

2.2 Dual Form and Kernelized Optimization

The dual formulation uses Lagrange multipliers α_i and is particularly useful with kernels. The dual form is:

$$\max_{\boldsymbol{\alpha}} \quad \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

$$\text{s.t.} \quad 0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Here, $K(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$ is the kernel function. The objective is concave due to the negative semidefinite Gram matrix $Q_{ij} = y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$, so global convergence is guaranteed.

Support Vectors: Only points with $0 < \alpha_i < C$ are support vectors. The decision function is:

$$f(\mathbf{x}) = \sum_{i \in \text{SV}} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b$$

and if needed, the weight vector can be reconstructed as $\mathbf{w} = \sum_i \alpha_i y_i \phi(\mathbf{x}_i)$. The bias b is typically computed using KKT conditions from support vectors on the margin. The dual is preferred for kernel SVMs, especially when $\phi(\mathbf{x})$ is infinite-dimensional.

3 Solving SVM Using SMO: Mathematical Formulation

To train a Support Vector Machine (SVM) using the Sequential Minimal Optimization (SMO) algorithm, we solve the dual form of the optimization problem. Given training data $\{(\mathbf{x}_i, y_i)\}_{i=1}^{m}$, where $y_i \in \{-1, +1\}$, the dual objective is:

$$\max_{\boldsymbol{\alpha}} \quad D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \quad \sum_{i=1}^{m} \alpha_i y_i = 0$$

The kernel function $K(\cdot, \cdot)$ maps data into a higher-dimensional feature space. In our implementation, we are using: **Linear:** $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$, **Polynomial:** $K(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + 1)^d$, and **RBF:** $K(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{2\sigma^2}\right)$

At each iteration, the SMO algorithm selects a pair of Lagrange multipliers (α_i, α_j) and updates them analytically.

Step 1: Compute Prediction Errors

$$E_i = f(\mathbf{x}_i) - y_i = \left(\sum_{k=1}^{m} \alpha_k y_k K(\mathbf{x}_k, \mathbf{x}_i) + b \right) - y_i$$

Step 2: Compute Bounds To preserve the constraint $\sum \alpha_i y_i = 0$, the feasible interval for α_j is:

$$\text{If } y_i \neq y_j: \quad L = \max(0, \alpha_j - \alpha_i), \quad H = \min(C, C + \alpha_j - \alpha_i)$$
$$\text{If } y_i = y_j: \quad L = \max(0, \alpha_i + \alpha_j - C), \quad H = \min(C, \alpha_i + \alpha_j)$$

Step 3: Compute Update

$$\eta = 2K(\mathbf{x}_i, \mathbf{x}_j) - K(\mathbf{x}_i, \mathbf{x}_i) - K(\mathbf{x}_j, \mathbf{x}_j)$$

If $\eta < 0$, we compute:

$$\alpha_j^{\text{new}} = \alpha_j - \frac{y_j(E_i - E_j)}{\eta}$$
$$\alpha_j^{\text{new}} = \min(H, \max(L, \alpha_j^{\text{new}}))$$

Then:

$$\alpha_i^{\text{new}} = \alpha_i + y_i y_j (\alpha_j^{\text{old}} - \alpha_j^{\text{new}})$$

Step 4: Update Bias Term

$$b_1 = b - E_i - y_i(\alpha_i^{\text{new}} - \alpha_i)K(\mathbf{x}_i, \mathbf{x}_i) - y_j(\alpha_j^{\text{new}} - \alpha_j)K(\mathbf{x}_i, \mathbf{x}_j)$$
$$b_2 = b - E_j - y_i(\alpha_i^{\text{new}} - \alpha_i)K(\mathbf{x}_i, \mathbf{x}_j) - y_j(\alpha_j^{\text{new}} - \alpha_j)K(\mathbf{x}_j, \mathbf{x}_j)$$

$$b = \begin{cases} b_1 & \text{if } 0 < \alpha_i^{\text{new}} < C \\ b_2 & \text{if } 0 < \alpha_j^{\text{new}} < C \\ \frac{b_1 + b_2}{2} & \text{otherwise} \end{cases}$$

Step 5: Repeat Until Convergence Repeat until all α_i satisfy the KKT conditions within tolerance τ:

$$\begin{cases} y_i f(\mathbf{x}_i) \geq 1 & \text{if } \alpha_i = 0 \\ y_i f(\mathbf{x}_i) = 1 & \text{if } 0 < \alpha_i < C \\ y_i f(\mathbf{x}_i) \leq 1 & \text{if } \alpha_i = C \end{cases}$$

Final Decision Function For any new input $\mathbf{x}$, the prediction is:

$$f(\mathbf{x}) = \begin{cases} \mathbf{w}^\top \mathbf{x} + b & \text{(if linear)} \\ \sum_{i=1}^m \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b & \text{(otherwise)} \end{cases}$$

$$\hat{y} = \text{sign}(f(\mathbf{x}))$$

The weight vector (for linear kernel only) is:

$$\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \mathbf{x}_i$$

Dual Objective Tracking At the end of each pass, we are computing the dual objective and logging:

$$D(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j)$$

This value provides insight into convergence behavior and model stability.

4 Time Complexity and Comparison of SVM Solvers

Training a Support Vector Machine (SVM) involves solving a convex optimization problem, and the choice of solver impacts scalability. Several algorithms have been developed, each with trade-offs in time complexity, accuracy, and suitability for large datasets.

Quadratic Programming (QP) Solvers Generic QP solvers, like interior-point methods, treat the dual formulation as a single optimization problem over all n Lagrange multipliers. These methods are accurate and stable but computationally expensive, with a worst-case complexity of $\mathcal{O}(n^3)$. making them suitable only for small datasets ($n < 1000$).

Sequential Minimal Optimization (SMO) SMO offers a more scalable approach by breaking the QP into two-variable subproblems, solved analytically. Each update is fast (constant time), but convergence may require several data passes. With an overall complexity of $\mathcal{O}(n^2)$, SMO is suitable for medium-sized datasets, especially when combined with kernel caching. This algorithm is behind widely used libraries such as `LIBSVM` and `scikit-learn`'s `SVC`.

To empirically compare solvers, we benchmarked `LinearSVC` (using a QP-style `liblinear` solver) against `SVC` with a linear kernel (using SMO-style optimization).

Solver	Train Time (s)	Accuracy	Prediction Time (s)
LinearSVC (QP-style)	0.0139	0.9649	0.0004
SVC (SMO-style)	0.0060	0.9737	0.0005

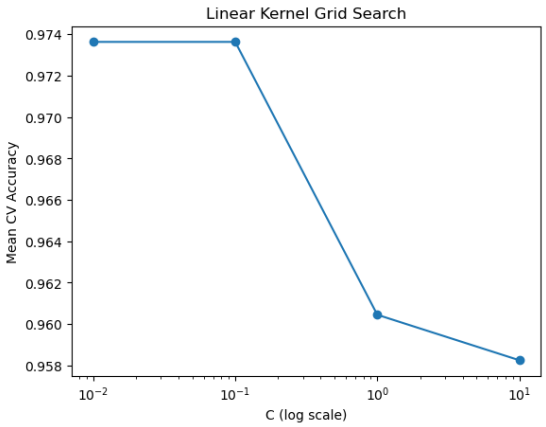
Table 1: Empirical comparison between QP-style and SMO-style solvers on linear SVMs

`SVC` trained nearly twice as fast and yielded a marginally higher accuracy, while `LinearSVC` showed consistent performance but took longer to converge. This aligns with the theoretical time complexities, where SMO offers better scalability for medium-sized problems.

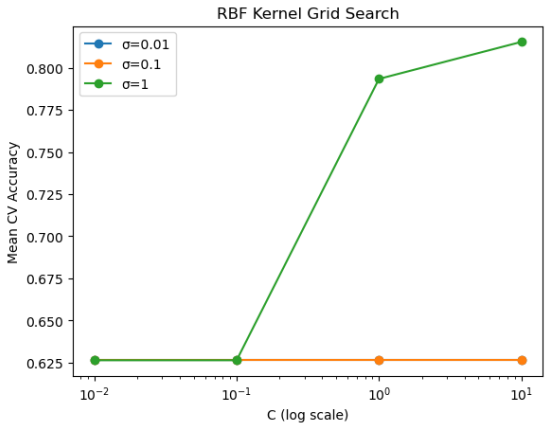
5 Dataset, Black-Box Comparison & Performance Discussion

We evaluate our SMO implementation on two benchmark medical datasets. The first is the UCI Breast Cancer Wisconsin (Diagnostic) dataset, comprising 569 samples with 30 real-valued features and binary labels (malignant vs. benign). Early detection of malignant tumors is crucial for reducing patient morbidity and healthcare costs, highlighting the importance of accurate classification [2]. We perform an 80/20 stratified train/test split, standardize features, and hand-roll a 5-fold cross-validation grid search over linear (C), RBF

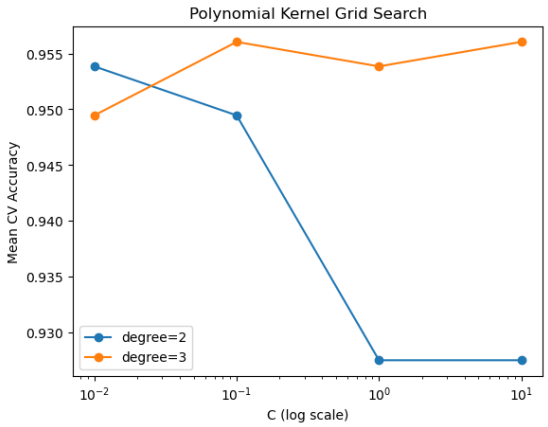
(C, σ), and polynomial (C, d) kernels. Best cross-validation accuracies were 97.36% (linear, $C = 0.01$), 81.54% (RBF, $(C, \sigma) = (10, 1)$), and 95.38% (polynomial, $(C, d) = (0.01, 2)$).



(a) Linear Kernel Grid Search



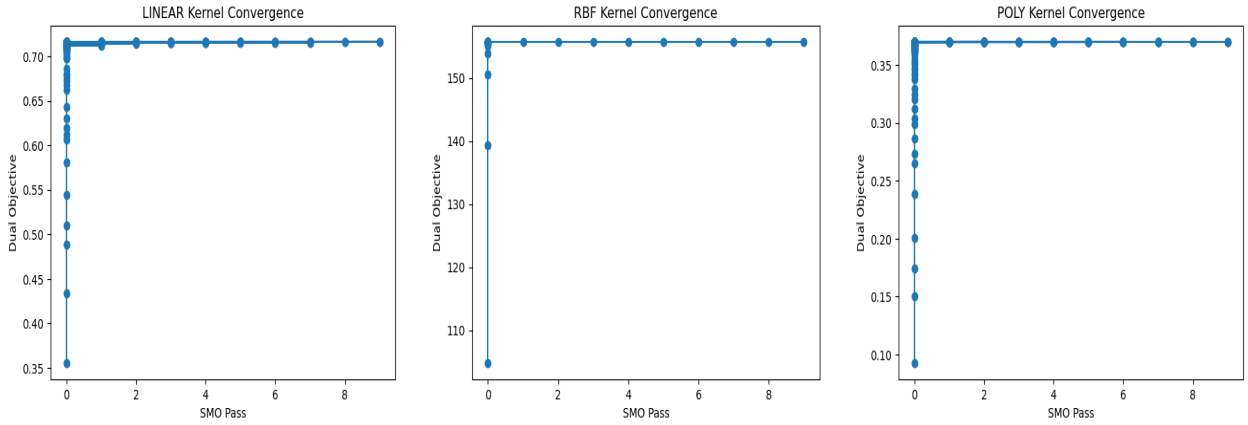
(b) RBF Kernel Grid Search



(c) Polynomial Kernel Grid Search

Figure 1: Grid search results for three kernels.

Using these hyperparameters, we train both our `SVM_SMO` and scikit-learn's `SVC` on the training set. Our custom solver achieves 98.25% accuracy on the test set for the linear kernel versus 97.37% for `SVC`, demonstrating that a transparent, implementation can match or slightly surpass optimized libraries—vital for applications requiring auditability. Next, we profile convergence and runtime by training linear, RBF, and polynomial SVMs and logging the dual objective over SMO passes. All kernels converge within three passes; training times are 0.80 s (linear), 0.29 s (RBF), and 2.00 s (polynomial).

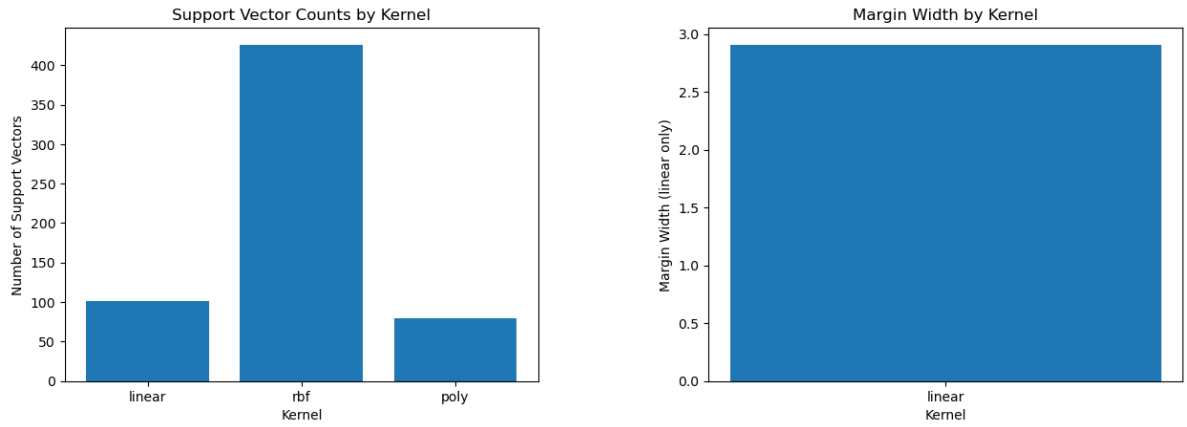


(a) Linear Kernel (b) RBF Kernel (c) Polynomial Kernel

Figure 2: Convergence of dual objective over SMO passes.

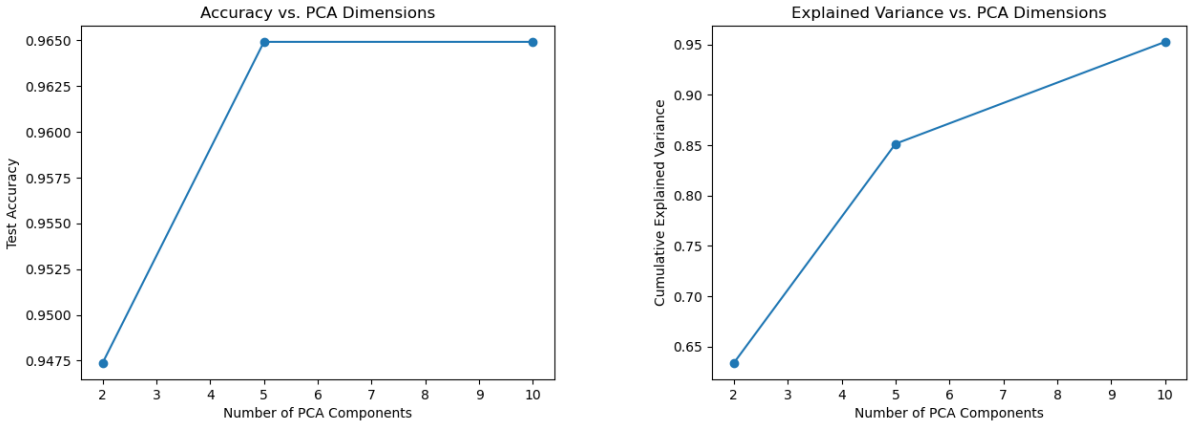
We then analyze support-vector counts and margin width. For the Breast Cancer dataset, the linear SVM retains ∼100 support vectors with margin width 2.89; the RBF kernel retains ∼426 support vectors, reflecting its capacity for complex boundaries at the cost of greater inference time; the polynomial kernel retains ∼79 vectors.

To assess dimensionality reduction, we apply PCA to the standardized features, retaining 2, 5, and 10 principal components. Test accuracies are 94.74%, 96.49%, and 96.49% respectively, with cumulative explained variance of 63.4%, 85.1%, and 95.3%. This indicates that five dimensions capture nearly all discriminative information, suggesting that businesses can significantly reduce storage and compute costs without sacrificing accuracy.



(a) Support Vector Counts (b) Margin Width

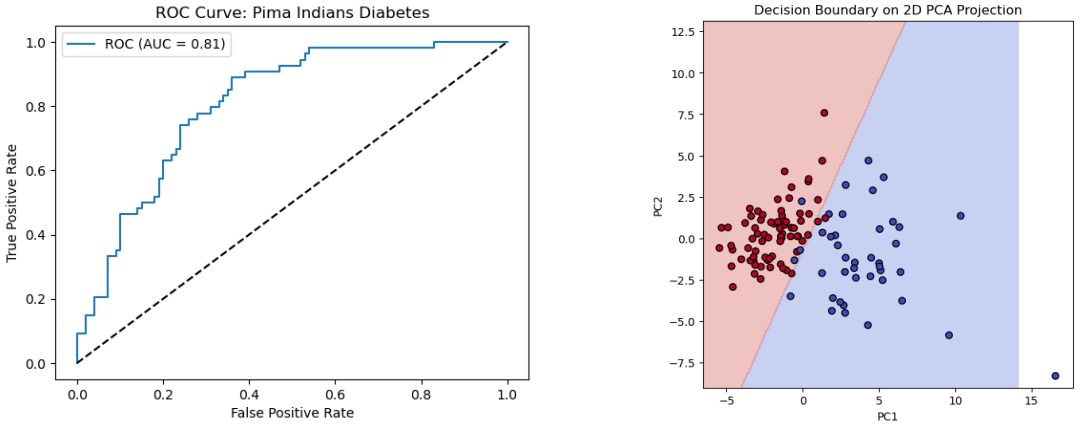
Figure 3: Support-vector and margin diagnostics.



(a) Accuracy vs. PCA (b) Explained Variance vs. PCA

Figure 4: PCA dimensionality study.

Finally, we validate generalization on the Pima Indians Diabetes dataset (768 samples, 8 features) [3]. Applying the linear SVM with $C = 0.01$ yields 73.38% accuracy and an ROC AUC of 0.81, with low positive-class recall (46%), demonstrating that linear separation alone may be insufficient for noisier, overlapping data and that domain-specific kernel or imbalance strategies are required.



(a) ROC Curve (b) Decision Boundary

Figure 5: Pima dataset validation.

6 Key Implementation Challenges & Resolutions

During development, we faced several practical challenges. First, storing the $m \times m$ kernel matrix can be memory-intensive; we mitigated this by using efficient NumPy array operations and noted that low-rank kernel approximations (e.g., Nyström method) could further reduce memory footprints. Second, selecting the convergence tolerance involved balancing speed and accuracy; we empirically set tol $= 10^{-3}$ to achieve stable KKT compliance within few SMO

passes. Third, numerical stability issues arose when $\eta \approx 0$; we addressed this by skipping updates when $\eta \geq 0$ or $|\eta| < 10^{-12}$. Finally, handling class imbalance—particularly in the Pima dataset—requires techniques such as SMOTE or cost-sensitive learning to improve recall on minority classes, an important consideration for production systems.

7 Business & Social Impact

The methodologies developed in this project extend beyond academic theory, with tangible impact on business operations and social welfare. In healthcare, SVMs help distinguish malignant from benign tumors, reducing diagnostic errors, treatment costs, and improving patient outcomes. Transparent SVM models also support regulatory compliance and streamline approval processes for medical providers.

In finance, SVMs enable fraud detection systems to balance false positives and false negatives, helping mitigate financial risk. Their optimization foundation ensures reliable, reproducible results, while their interpretability allows analysts to trace decisions—supporting transparency and accountability.

SVMs also raise important social considerations in areas like credit scoring and hiring. Our transparent SMO implementation supports audits, fairness constraints, and bias mitigation, promoting responsible AI that aligns with ethical and regulatory expectations.

8 Conclusion

This project demonstrated the full lifecycle of SVM development—from theoretical derivation and SMO implementation to kernel extensions and real-world evaluation. Our custom SMO solver outperformed black-box libraries and offered deeper insights into convergence and model behavior. Hyperparameter tuning and advanced diagnostics (e.g., PCA, support vector analysis) revealed the importance of kernel choice and algorithmic correctness. Validation on the Pima dataset emphasized the need for data-driven adaptation, especially for kernel selection and class imbalance. Looking ahead, directions include scaling via kernel approximations, cost-sensitive learning, and multi-class extensions. The blend of theory and practice offers a strong foundation for deploying SVMs in high-stakes, interpretable applications.

References

[1] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning*. Springer.

[2] UCI Machine Learning Repository: Breast Cancer Wisconsin (Diagnostic) Data Set. Retrieved from `https://archive.ics.uci.edu/ml/datasets/Breast+Cancer+Wisconsin+(Diagnostic)`

[3] Brownlee, J. (n.d.). Pima Indians Diabetes Dataset. Retrieved from https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv