A Comparative Study of SVM Solvers and Their Implementations

Arshia Izadyari, Parsa Ghezelbash

February 6, 2025

Abstract

Support Vector Machines (SVMs) remain a robust choice for classification tasks. In this paper, we examine several solvers for linear SVM—ranging from exact quadratic programming to iterative methods such as Gradient Descent, Cutting Plane, Newton-based methods, and Pegasos—and explore their theoretical foundations and practical implementations. We discuss how each solver builds a secondary problem variation (often a dual or surrogate formulation) from the primary soft-margin SVM formulation. We also describe wrappers for binary and multi-class SVM (OvO, OvR) and present benchmarking results on the MNIST dataset, illustrating the impact of standard scaling and Principal Component Analysis (PCA) for dimensionality reduction.

Contents

1 Introduction

Support Vector Machines (SVMs) are widely used for binary classification by maximizing the margin between two classes in a high-dimensional space. The standard soft-margin SVM problem adds slack variables to handle noisy or non-separable data, governed by a regularization parameter C. Over the years, many approaches have emerged to solve or approximate the SVM optimization problem. These approaches differ in how they transform the *primary SVM problem* to a *secondary formulation* which may be more amenable to efficient optimization.

The primary (or primal) soft-margin SVM problem is formulated as:

$$\begin{aligned} \min_{w,b,\{\xi_i\}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y^i\left[w^T x^i + b\right] - 1 + \xi_i \geq 0 \qquad i = 1,\ldots,l \\ & \xi_i \geq 0 \qquad i = 1,\ldots,l \end{aligned}$$

From this formulation, various solvers derive a secondary formulation—either via dualization or through surrogate loss approximations—which can be optimized by specific algorithms. In this paper we detail these transformations and discuss the underlying theory for each solver.

2 Deriving the Secondary SVM Formulation

The idea behind a secondary SVM problem variation is to recast the original constrained optimization problem in a form that is easier to solve by a particular algorithm. Two common strategies are:

- **Dual Formulation:** By forming the Lagrangian and applying Karush-Kuhn-Tucker (KKT) conditions, the primal problem is transformed into a dual quadratic programming (QP) problem. The dual problem leverages the fact that the data appears in

inner products, which allows the use of kernel functions. Some solvers (such as those implemented in scikit-learn) use dual coordinate descent methods.

- **Surrogate Loss Minimization:** Many iterative solvers instead replace the hard constraints with a hinge-loss penalty, resulting in an unconstrained minimization problem:

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

  This form is amenable to gradient-based or subgradient-based methods, as the constraint violations are "penalized" in the objective.

Below we explain for each solver how we obtain this secondary formulation from the primary formulation, and we detail the theory behind each method.

3 SVM Solvers and Their Theoretical Foundations

3.1 Gradient Descent Solver

Secondary Problem Derivation: Starting from the primary SVM formulation, we replace the inequality constraints by introducing the hinge loss function:

$$\ell_{\text{hinge}}(w, b; x_i, y_i) = \max\{0,\, 1 - y_i(w^T x_i + b)\}.$$

Thus, the optimization id equivalent to:

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

Theory: The gradient descent solver applies (sub)gradient descent to this unconstrained minimization problem. The objective is differentiable almost everywhere (except at the hinge point where the subgradient is used). The The gradient with respect to w is given by:

$$\nabla_w = w + C \sum_{i:\, y_i(w^T x_i + b) < 1} (-y_i x_i),$$

and similarly for b:

$$\nabla_b = C \sum_{i:\, y_i(w^T x_i + b) < 1} (-y_i).$$

The update step then follows a standard gradient descent rule:

$$w \leftarrow w - \eta\, \nabla_w, \quad b \leftarrow b - \eta\, \nabla_b.$$

The solver iteratively adjusts w and b, handling the non-differentiability of the hinge loss using subgradient methods.

Algorithm 1 Gradient Descent Solver

1: **Initialize:** $w \leftarrow$ random, $b \leftarrow 0$
2: **for** $t = 1$ to n_iters **do**
3: Compute margins $m_i = y_i(w^T x_i + b)$
4: Define hinge gradient:

$$\nabla_w = w + C \sum_{i:m_i<1} (-y_i x_i), \quad \nabla_b = C \sum_{i:m_i<1} (-y_i)$$

5: Update

$$w \leftarrow w - \eta \nabla_w, \quad b \leftarrow b - \eta \nabla_b.$$

6: **end for**

3.2 Cutting Plane Solver

Secondary Problem Derivation: Consider the primal soft-margin SVM problem with $b = 0$ where the error term is divided by the number of elements in the training set:

$$\begin{array}{lll} \min\limits_{w,b,\{\xi_i\}} & \frac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{l} \xi_i & \\ \text{subject to} & y^i \left[w^T x^i + b\right] - 1 + \xi_i \geq 0 & i = 1, \ldots, l \\ & \xi_i \geq 0 \qquad i = 1, \ldots, l & \end{array}$$

we will introduce an equivalent formulation called Structural Classification for this problem SVM as below:

$$\begin{array}{lll} \min\limits_{w,b,\xi} & \frac{1}{2}\|w\|^2 + C\xi & \\ \text{subject to} & \forall c \in \{0,1\}^l : & \frac{1}{l} w^T \sum_{i=1}^{l} c_i y_i x_i \geq \frac{1}{l} \sum_{i=1}^{l} c_i - \xi \\ & \xi \geq 0. & \end{array}$$

Proof: c_i corresponds to different set of possible constraints. then we prove that these two problems are equivalent by showing that any solution of the Secondary Problem w^* is also a solution of the Primal Problem.

For a given w^*, the minimum feasible value of the objective for the primal problem is achieved by $\xi_i^* = \max\{0, 1 - y_i w^T x_i\}$.

For a given w^*, the minimum feasible value of the objective for the secondary problem is achieved by $\xi^* = \max_{c \in \{0,1\}^l} \{\frac{1}{l} \sum_{i=1}^{l} c_i - \frac{1}{l} w^T \sum_{i=1}^{l} c_i y_i x_i\}$.

The above equation can be written as:

$$\begin{aligned}\min \xi &= \max_{c\in\{0,1\}^l} \{\frac{1}{l}\sum_{i=1}^{l} c_i - \frac{1}{l} w^T \sum_{i=1}^{l} c_i y_i x_i\} \\ &= \frac{1}{l} \sum_{i=1, 1-w^T y_i x_i \geq 0}^{l} (c_i - w^T c_i y_i x_i) \\ &= \sum_{i=1}^{l} \max\{0, \frac{1}{l} - \frac{1}{l} w^T y_i x_i\} \\ &= \min \frac{1}{l} \sum_{i=1}^{l} \xi_i\end{aligned}$$

Then the equivalence of these two problems is proven.

Algorithm 2 Cutting Plane Method

Input: Training set TS, regularization parameter C, precision ϵ
Initialize: Constraint set $W \leftarrow \emptyset$, randomly initialize w, set $\xi \leftarrow 0$
repeat
  Solve the reduced optimization problem:

$$\min_{w,\xi} \frac{1}{2}\|w\|^2 + C\xi$$

  subject to:

$$\forall c \in W: \quad \frac{1}{l} w^T \sum_{i=1}^{l} c_i y_i x_i \geq \frac{1}{l} \sum_{i=1}^{l} c_i - \xi.$$

  for $i = 1$ to l **do**
    Compute:

$$c_i = \begin{cases} 1, & \text{if } y_i w^T x_i < 1, \\ 0, & \text{otherwise.} \end{cases}$$

  end for
  Update constraint set: $W \leftarrow W \cup \{c\}$
until

$$\frac{1}{l} \sum_{i=1}^{l} c_i - \frac{1}{l} \sum_{i=1}^{l} c_i y_i w^T x_i \leq \xi + \epsilon.$$

return (w, ξ)

The algorithm iteratively constructs a sufficient subset W of the set of constraints. The algorithm starts with an empty set of constraints W. In each iteration, it first computes the optimum over the current working set W. Then it then finds the most violated constraint and adds it to the working set W.

It is proven that the number of iterations is at most:

$$O\left(\max\left\{\frac{2}{\epsilon}, \frac{8CR^2}{\epsilon^2}\right\}\right)$$

where:

- ϵ is the precision level (smaller ϵ means higher accuracy).
- C is the SVM regularization parameter.
- $R = \max_i \|x_i\|$ is the maximum norm of the input features.

The key insight is that the number of iterations is independent of the dataset size n. This makes it highly scalable compared to traditional quadratic programming (QP) approaches. Each iteration involves solving a reduced SVM optimization problem.

Using Dual Optimization (Joachims et al., 2009) Instead of solving the primal problem at each iteration, the dual problem is solved, reducing computation complexity. The dual formulation allows the use of kernelized SVMs, enabling nonlinear classification.

Reducing Feature Space (Joachims & Yu, 2009) Instead of using all support vectors to construct w, they use a sparse subset of basis vectors:

$$F' = \text{span}\{\phi(b_1), ..., \phi(b_k)\}$$

where k is much smaller than the number of training samples l. This reduces the **per-iteration cost** significantly.

Approximate Constraint Selection (Franc & Sonnenburg, 2008) Instead of adding only the most violated constraint, they introduced batch updates where multiple violated constraints are added per iteration. This reduces the total number of iterations needed for convergence.

After these optimizations, the cutting plane method achieves:

- **Linear complexity in sparse settings:** $O(sl)$, where s is the number of nonzero features.
- **For nonlinear kernels:** The complexity worsens to $O(m^3 + ml^2)$, where m is the number of constraints added.
- **With basis vector reduction:** The time complexity per iteration improves to $O(m^3 + mk^2 + kl)$, where k is the number of basis vectors used.
- **Traditional Quadratic Programming (QP)** methods for SVM scale as $O(l^2)$ or $O(l^3)$ in training time, making them infeasible for large datasets.
- The **cutting plane method** scales **independently of** n and focuses only on a small subset of constraints, leading to **faster convergence**.

Method	Complexity Per Iteration	Number of Iterations	Total Complexity
Standard QP	$O(l^3)$	1	$O(l^3)$
Cutting Plane (Basic)	$O(sl)$	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{sl}{\epsilon}\right)$
Cutting Plane (Dual)	$O(m^3 + ml^2)$	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{m^3+ml^2}{\epsilon}\right)$
Cutting Plane (Basis Vectors)	$O(m^3 + mk^2 + kl)$	$O\left(\frac{1}{\epsilon}\right)$	$O\left(\frac{m^3+mk^2+kl}{\epsilon}\right)$

3.3 CVXPY OSQP Solver

Secondary Problem Derivation:

The primal formulation of the soft-margin SVM is given by the optimization problem:

$$\begin{aligned} \min_{w,b,\{\xi_i\}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y^i\left[w^T x^i + b\right] - 1 + \xi_i \geq 0 \qquad i = 1,\ldots,l \\ & \xi_i \geq 0 \qquad i = 1,\ldots,l \end{aligned}$$

The Lagrangian function is:

$$\begin{aligned} \mathcal{L}(w,b,\xi,\alpha,\mu) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ - \sum_{i=1}^{n}\alpha_i\left[y_i(w^T x_i + b) - 1 + \xi_i\right] \\ - \sum_{i=1}^{n}\mu_i\xi_i. \end{aligned}$$

Stationary Conditions To eliminate w, b, ξ, we take derivatives and set them to zero:

$$\frac{\partial\mathcal{L}}{\partial w} = w - \sum_{i=1}^{n}\alpha_i y_i x_i = 0$$

$$\Rightarrow w = \sum_{i=1}^{n}\alpha_i y_i x_i.$$

$$\frac{\partial\mathcal{L}}{\partial b} = -\sum_{i=1}^{n}\alpha_i y_i = 0.$$

$$\sum_{i=1}^{n}\alpha_i y_i = 0.$$

$$\frac{\partial\mathcal{L}}{\partial \xi_i} = C - \alpha_i - \mu_i = 0.$$

$$\mu_i \geq 0 \Rightarrow 0 \leq \alpha_i \leq C.$$

Substituting w into $\mathcal{L}$, we get:

$$\begin{aligned} \mathcal{L} = \frac{1}{2}\left(\sum_{i=1}^{n}\alpha_i y_i x_i\right)^T\left(\sum_{j=1}^{n}\alpha_j y_j x_j\right) - \sum_{i=1}^{n}\alpha_i y_i x_i^T w \\ + \sum_{i=1}^{n}\alpha_i - \sum_{i=1}^{n}\alpha_i\xi_i - \sum_{i=1}^{n}\mu_i\xi_i. \end{aligned}$$

Using $w = \sum_{i=1}^{n} \alpha_i y_i x_i$, we simplify:

$$\mathcal{L} = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j).$$

Thus, the dual optimization problem is:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j (x_i^T x_j) \\ \text{subject to} \quad & \sum_{i=1}^{n} \alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1, \ldots, n. \end{aligned}$$

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T X^T X \alpha - 1^T \alpha \\ \text{subject to} \quad & \sum_{i=1}^{l} \alpha_i y_i = 0 \\ & 0 \leq \alpha_i \leq C, \end{aligned}$$

where:

$$X = \left[y^1 x^1, \ldots, y^l x^l \right]$$

Then both the priamal and dual problem are QP.
The OSQP solver solves the QP problem:

$$\begin{aligned} \min_{x,z} \quad & \frac{1}{2} x^T P x + q^T x \\ \text{subject to} \quad & Ax = z \\ & z \in C \end{aligned}$$

where P is a positive semi-definite (PSD) matrix that defines the quadratic term.

OSQP solves QP problems using the Alternating Direction Method of Multipliers (ADMM), which is well-suited for large-scale sparse problems.

Theory: OSQP reformulates the constrained QP into an unconstrained problem using an Augmented Lagrangian approach. The given QP problem can be written as:

$$\begin{aligned} \min_{\tilde{x}, \tilde{z}} \quad & \frac{1}{2} \tilde{x}^T P \tilde{x} + q^T \tilde{x} + \mathcal{I}_{Ax=z}(\tilde{x}, \tilde{z}) + \mathcal{I}_{\mathcal{C}}(z) \\ \text{subject to} \quad & (\tilde{x}, \tilde{z}) = (x, z) \end{aligned}$$

where $\mathcal{I}_{Ax=z}$ and $\mathcal{I}_{\mathcal{C}}$ are the indicator functions given by

$$\mathcal{I}_{Ax=z}(x, z) = \begin{cases} 0 & Ax = z \\ +\infty & \text{otherwise} \end{cases}, \qquad \mathcal{I}_{\mathcal{C}}(z) = \begin{cases} 0 & z \in \mathcal{C} \\ +\infty & \text{otherwise} \end{cases}.$$

An iteration of ADMM for solving this problem consists of the following steps:

$$
\begin{aligned}
(\tilde{x}^{k+1}, \tilde{z}^{k+1}) \leftarrow \operatorname{argmin}_{(\tilde{x},\tilde{z}):A\tilde{x}=\tilde{z}} & \frac{1}{2}\tilde{x}^T P\tilde{x} + q^T\tilde{x} \\
& + \frac{\sigma}{2}\|\tilde{x} - x^k + \sigma^{-1}w^k\|_2^2 + \frac{\rho}{2}\|\tilde{z} - z^k + \rho^{-1}y^k\|_2^2 \\
x^{k+1} &\leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k + \sigma^{-1}w^k \\
z^{k+1} &\leftarrow \Pi\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \rho^{-1}y^k\right) \\
w^{k+1} &\leftarrow w^k + \sigma\left(\alpha\tilde{x}^{k+1} + (1-\alpha)x^k - x^{k+1}\right) \\
y^{k+1} &\leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\right)
\end{aligned}
$$

where $\sigma > 0$ and $\rho > 0$ are the *step-size parameters*, $\alpha \in (0, 2)$ is the *relaxation parameter*, and Π denotes the Euclidean projection onto $\mathcal{C}$.

As you can see the above update rules involve solving the below problem:

$$
\begin{array}{ll}
\text{minimize} & (1/2)\tilde{x}^T P\tilde{x} + q^T\tilde{x} + (\sigma/2)\|\tilde{x} - x^k\|_2^2 + (\rho/2)\|\tilde{z} - z^k + \rho^{-1}y^k\|_2^2 \\
\text{subject to} & A\tilde{x} = \tilde{z}.
\end{array}
$$

The optimality conditions for this equality constrained QP are

$$
\begin{aligned}
P\tilde{x}^{k+1} + q + \sigma(\tilde{x}^{k+1} - x^k) + A^T\nu^{k+1} &= 0, \\
\rho(\tilde{z}^{k+1} - z^k) + y^k - \nu^{k+1} &= 0, \\
A\tilde{x}^{k+1} - \tilde{z}^{k+1} &= 0,
\end{aligned}
$$

where $\nu^{k+1} \in \mathbb{R}^m$ is the Lagrange multiplier associated with the constraint $Ax = z$. By eliminating the variable $\tilde{z}^{k+1}$ from, the above linear system reduces to

$$
\begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1}y^k \end{bmatrix},
$$

with $\tilde{z}^{k+1}$ recoverable as

$$
\tilde{z}^{k+1} = z^k + \rho^{-1}(\nu^{k+1} - y^k).
$$

Algorithm 3 OSQP Solver for Quadratic Programming

Given: initial values x^0, z^0, y^0 and parameters $\rho > 0$, $\sigma > 0$, $\alpha \in (0, 2)$
repeat
  $(\tilde{x}^{k+1}, \nu^{k+1}) \leftarrow$ solve linear system:
  $\begin{bmatrix} P + \sigma I & A^T \\ A & -\rho^{-1}I \end{bmatrix} \begin{bmatrix} \tilde{x}^{k+1} \\ \nu^{k+1} \end{bmatrix} = \begin{bmatrix} \sigma x^k - q \\ z^k - \rho^{-1}y^k \end{bmatrix}$
  $\tilde{z}^{k+1} \leftarrow z^k + \rho^{-1}(\nu^{k+1} - y^k)$
  $x^{k+1} \leftarrow \alpha\tilde{x}^{k+1} + (1-\alpha)x^k$
  $z^{k+1} \leftarrow \Pi\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k + \rho^{-1}y^k\right)$
  $y^{k+1} \leftarrow y^k + \rho\left(\alpha\tilde{z}^{k+1} + (1-\alpha)z^k - z^{k+1}\right)$
until termination criterion is satisfied

3.4 Sklearn SVM Solver

Secondary Problem Derivation: LIBSVM, which `sklearn.svm.SVC` is based on, employs the Optimization via Sequential Minimal Optimization (SMO) algorithm. The core idea of SMO is:

Instead of solving for all α_i simultaneously, it optimizes two variables at a time while keeping others fixed. Given two chosen multipliers α_i and α_j, it finds an optimal step within their feasible range. The updates are computed analytically by solving a reduced quadratic programming (QP) problem. This approach significantly speeds up optimization and is particularly effective for large-scale datasets.

C-SVC: This formulation controls the trade-off between margin maximization and misclassification through the parameter C.

Given training vectors $x_i \in^n$, $i = 1, \ldots, l$, in two classes, and an indicator vector $y \in^l$ such that $y_i \in \{1, -1\}$, C-SVC (Boser et al., 1992; Cortes and Vapnik, 1995) solves the following primal optimization problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2} w^T w + C \sum_{i=1}^{l} \xi_i \\ \text{subject to} \quad & y_i(w^T \phi(x_i) + b) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \quad i = 1, \ldots, l, \end{aligned} \tag{1}$$

where $\phi(x_i)$ maps x_i into a higher-dimensional space and $C > 0$ is the regularization parameter. Due to the possible high dimensionality of the vector variable w, usually they solve the following dual problem:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - e^T \alpha \\ \text{subject to} \quad & y^T \alpha = 0, \\ & 0 \leq \alpha_i \leq C, \quad i = 1, \ldots, l, \end{aligned} \tag{2}$$

where $e = [1, \ldots, 1]^T$ is the vector of all ones, Q is an l by l positive semidefinite matrix, in which $Q_{ij} \equiv y_i y_j K(x_i, x_j)$, and $K(x_i, x_j) \equiv \phi(x_i)^T \phi(x_j)$ is the kernel function.

After problem (2) is solved, using the primal-dual relationship, the optimal w satisfies

$$w = \sum_{i=1}^{l} y_i \alpha_i \phi(x_i) \tag{3}$$

and the decision function is

$$\left(w^T \phi(x) + b\right) = \left(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right).$$

They store $y_i \alpha_i \; \forall i$, b, label names, support vectors, and other information such as kernel parameters in the model for prediction.

ν-SVC: An alternative formulation where $\nu \in (0, 1]$ controls the fraction of support vectors and margin errors explicitly.

The ν-support vector classification (Schölkopf et al., 2000) has introduced a new parameter $\nu \in (0, 1]$. It is proved that ν is an upper bound on the fraction of training errors and a lower bound of the fraction of support vectors.

Given training vectors $x_i \in \mathbb{R}^n$, $i = 1, \ldots, l$, in two classes, and a vector $y \in \mathbb{R}^l$ such that $y_i \in \{1, -1\}$, the primal optimization problem is

$$\min_{w,b,\xi,\rho} \frac{1}{2} w^T w - \nu\rho + \frac{1}{l} \sum_{i=1}^{l} \xi_i$$

subject to

$$y_i(w^T \phi(x_i) + b) \geq \rho - \xi_i, \quad \xi_i \geq 0, \quad i = 1, \ldots, l, \quad \rho \geq 0.$$

The dual problem is

$$\min_{\alpha} \frac{1}{2} \alpha^T Q \alpha$$

subject to

$$0 \leq \alpha_i \leq \frac{1}{l}, \quad i = 1, \ldots, l, \quad e^T \alpha \geq \nu, \quad y^T \alpha = 0,$$

where $Q_{ij} = y_i y_j K(x_i, x_j)$. Chang and Lin (2001) show that problem (5) is feasible if and only if

$$\nu \leq \frac{2 \min(\#y_i = +1, \#y_i = -1)}{l} \leq 1,$$

so the usable range of ν is smaller than $(0, 1]$.

In LIBSVM, any integer can be a label name, so we map label names to ± 1 by assigning the first training instance to have $y_1 = +1$.

The decision function is

$$\operatorname{sgn}\left(\sum_{i=1}^{l} y_i \alpha_i K(x_i, x) + b \right).$$

It is shown that $e^T \alpha \geq \nu$ can be replaced by $e^T \alpha = \nu$ (Crisp and Burges, 2000; Chang and Lin, 2001). In LIBSVM, they solve a scaled version of problem (5) because numerically α_i may be too small due to the constraint $\alpha_i \leq \frac{1}{l}$:

$$\min_{\bar{\alpha}} \frac{1}{2} \bar{\alpha}^T Q \bar{\alpha}$$

subject to

$$0 \leq \bar{\alpha}_i \leq 1, \quad i = 1, \ldots, l, \quad e^T \bar{\alpha} = \nu l, \quad y^T \bar{\alpha} = 0.$$

If α is optimal for the dual problem (5) and ρ is optimal for the primal problem (4), Chang and Lin (2001) show that α/ρ is an optimal solution of C-SVM with $C = 1/(\rho l)$. Thus, in LIBSVM, we output $(\alpha/\rho, b/\rho)$ in the model.

3.5 Newton Method Solver

The primal problem for soft-margin SVM is:

$$\begin{aligned} \min_{w,b,\{\xi_i\}} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l}\xi_i \\ \text{subject to} \quad & y^i\left[w^T x^i + b\right] - 1 + \xi_i \geq 0 \qquad i = 1,\ldots,l \\ & \xi_i \geq 0 \qquad i = 1,\ldots,l \end{aligned}$$

Using the Lagrangian method, we derive the dual optimization problem:

$$\begin{aligned} \max_{\alpha} \quad & \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j K_{(x_i,x_j))} \\ \text{subject to} \quad & \sum_{i=1}^{n}\alpha_i y_i = 0, \quad 0 \leq \alpha_i \leq C, \quad \forall i = 1,\ldots,n. \end{aligned}$$

where:

- α_i are the Lagrange multipliers.
- $K(x_i, x_j)$ is the kernel function (for linear SVM, $K(x_i, x_j) = x_i^T x_j$).

This is a convex quadratic programming (QP) problem, which Newton's method can efficiently solve.

Newton's Method for Solving the Dual SVM

Newton's method iteratively refines the solution using the Newton-Raphson update rule:

$$\alpha^{(k+1)} = \alpha^{(k)} - H^{-1}\nabla f(\alpha^{(k)}).$$

where:

- H is the Hessian matrix (second derivative of the objective function).
- $\nabla f(\alpha)$ is the gradient of the objective function.

Gradient Computation The gradient of the dual SVM objective is:

$$\nabla f(\alpha) = \mathbf{1} - Q\alpha.$$

where:

- $Q_{ij} = y_i y_j K(x_i, x_j)$ is the Gram matrix.
- $\mathbf{1}$ is an all-ones vector.

Hessian Matrix Computation The Hessian matrix H is:

$$H = Q.$$

Since Q is a positive semi-definite matrix, Newton's method ensures quadratic convergence.

Newton's Update Rule for SVM Newton's method updates α using:

$$\alpha^{(k+1)} = \alpha^{(k)} + Q^{-1}(\mathbf{1} - Q\alpha^{(k)}).$$

However, computing Q^{-1} is expensive for large datasets. To handle this, we use approximate Newton methods such as:

- Quasi-Newton methods (e.g., BFGS).
- Conjugate gradient solvers.

Algorithm 4 Newton's Method for SVM

1: **Input:** Data (x_i, y_i), kernel $K(x_i, x_j)$, regularization C.
2: **Initialize:** Set $\alpha^{(0)}$ (e.g., random or zero).
3: **repeat**
4: Compute gradient: $\nabla f(\alpha) = \mathbf{1} - Q\alpha$.
5: Compute Hessian: $H = Q$.
6: Solve $Hd = -\nabla f(\alpha)$ for d.
7: Update $\alpha^{(k+1)} = \alpha^{(k)} + d$.
8: **until** Convergence
9: **Output:** Optimal dual variables α^*.

Method	Complexity Per Iteration	Iterations	Total Complexity
Gradient Descent	$O(n^2)$	$O(1/\epsilon)$	$O(n^2/\epsilon)$
Newton's Method	$O(n^3)$	$O(\log(1/\epsilon))$	$O(n^3 \log(1/\epsilon))$
Quasi-Newton (BFGS)	$O(n^2)$	$O(\log(1/\epsilon))$	$O(n^2 \log(1/\epsilon))$

Table 1: Comparison of Complexity for Different Optimization Methods

Newton's method has faster convergence than gradient descent but is computationally expensive for large datasets due to matrix inversion.

Advantages

- Quadratic convergence (faster than gradient descent).
- Better stability than first-order methods.

Disadvantages

- Computing Hessian Q^{-1} is expensive for large datasets.

3.6 Pegasos

Pegasos is a stochastic sub-gradient descent method for solving the Support Vector Machine (SVM) optimization problem. Pegasos updates the model parameters using only a single example per iteration, significantly reducing computational cost.

The primal form of the SVM optimization problem is given by:

$$\min_{w \in \mathbb{R}^d} \quad f(w) = \frac{\lambda}{2}\|w\|^2 + \frac{1}{m}\sum_{i=1}^{m} \ell(w; (x_i, y_i))$$

where:

- $\lambda > 0$ is the regularization parameter.
- m is the number of training samples.
- $\ell(w; (x_i, y_i))$ is the hinge loss function defined as:

$$\ell(w; (x_i, y_i)) = \max\{0, 1 - y_i \langle w, x_i \rangle\}.$$

Since the hinge loss function is non-differentiable at $y_i\langle w, x_i\rangle = 1$, Pegasos applies a sub-gradient descent approach. The sub-gradient of the objective function at iteration t is computed as:

$$\nabla_t = \lambda w_t - \mathbf{1}[y_i \langle w_t, x_t \rangle < 1] y_i x_t.$$

The model update rule follows a stochastic sub-gradient descent with a step size:

$$w_{t+1} = w_t - \eta_t \nabla_t,$$

where the step size is set as:

$$\eta_t = \frac{1}{\lambda t}.$$

It is proven that that $\|w^*\| \leq \frac{1}{\sqrt{\lambda}}$. To ensure bounded norm, Pegasos includes a projection step:

$$w_{t+1} \leftarrow \min\left(1, \frac{1}{\sqrt{\lambda}} \frac{1}{\|w_{t+1}\|}\right) w_{t+1}.$$

Pegasos provides a convergence rate of:

$$f(w_T) - f(w^*) \leq \mathcal{O}\left(\frac{1}{\lambda T}\right).$$

This is significantly faster than conventional stochastic gradient methods, which require $\mathcal{O}(1/\lambda\epsilon^2)$ iterations to reach an ϵ-accurate solution.

Pegasos can also be extended to a mini-batch setting where k examples are used per iteration. The sub-gradient update rule then becomes:

Algorithm 5 Pegasos Algorithm for SVM

Input: Training set $S = \{(x_i, y_i)\}_{i=1}^{m}$, regularization parameter λ, number of iterations T
Initialize: Set $w_1 = 0$
for $t = 1$ to T **do**
  Choose a random training example (x_t, y_t)
  Set $\eta_t = \frac{1}{\lambda t}$
  if $y_t \langle w_t, x_t \rangle < 1$ **then**
    Update $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t + \eta_t y_t x_t$
  else
    Update $w_{t+1} \leftarrow (1 - \eta_t \lambda) w_t$
  end if
  Apply projection step:
  $w_{t+1} \leftarrow \min\left(1, \frac{1}{\sqrt{\lambda}} \frac{1}{\|w_{t+1}\|}\right) w_{t+1}$
end for
Output: Final weight vector w_T

$$\nabla_t = \lambda w_t - \frac{1}{k} \sum_{i \in A_t} 1[y_i \langle w_t, x_i \rangle < 1] y_i x_i.$$

where A_t is a randomly sampled batch of size k.
The update rule remains:

$$w_{t+1} = w_t - \eta_t \nabla_t.$$

Computational Complexity
Pegasos achieves a total runtime complexity of:

$$\mathcal{O}\left(\frac{d}{\lambda \epsilon}\right).$$

where:

- d is the number of non-zero features in each training example.
- λ is the regularization parameter.
- ϵ is the desired accuracy.

This is independent of the dataset size m, making Pegasos highly scalable for large datasets.

3.7 Least Squares SVM (LS-SVM) Solver

Least Squares Support Vector Machine (LS-SVM) is a reformulation of standard SVM where the hinge loss is replaced with a least squares loss function, converting the problem into a

linear system rather than a quadratic programming (QP) problem. This significantly reduces computational complexity.

The primal optimization problem for LS-SVM is given by:

$$\begin{aligned} \min_{w,b,e} \quad & \tfrac{1}{2}\|w\|^2 + \tfrac{C}{2}\sum_{i=1}^{n} e_i^2 \\ \text{subject to} \quad & y_i(w^T x_i + b) = 1 - e_i, \quad \forall i = 1, \ldots, n. \end{aligned}$$

Lagrangian Formulation

To solve this constrained optimization problem, we introduce Lagrange multipliers α_i, and construct the Lagrangian function:

$$\begin{aligned} \mathcal{L}(w, b, e, \alpha) = \frac{1}{2}\|w\|^2 + \frac{C}{2}\sum_{i=1}^{n} e_i^2 \\ - \sum_{i=1}^{n} \alpha_i \left[y_i(w^T x_i + b) - 1 + e_i \right]. \end{aligned} \tag{1}$$

where α_i are the Lagrange multipliers.

To eliminate w, b, e, we take derivatives and set them to zero.

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{n} \alpha_i y_i x_i = 0.$$

$$\Rightarrow w = \sum_{i=1}^{n} \alpha_i y_i x_i.$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0.$$

$$\frac{\partial \mathcal{L}}{\partial e_i} = C e_i - \alpha_i = 0.$$

$$\Rightarrow e_i = \frac{\alpha_i}{C}.$$

$$\frac{\partial \mathcal{L}}{\partial \alpha_i} = y_i(w^T x_i + b) - 1 + e_i = 0.$$

$$\Rightarrow \alpha_i = C - C y_i(w^T x_i + b).$$

Substituting the optimal values into the Lagrangian, we get the dual formulation:

$$\begin{bmatrix} 0 & Y^T \\ Y & K + \frac{1}{C} I \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}$$

where:

- $Y = [y_1, y_2, \ldots, y_n]^T$,

- $K = XX^T$,
- $X = [y^1x^1, \ldots, y^nx^n]$
- I is the identity matrix,
- $\mathbf{1}$ is an all-ones vector.

This results in a linear system rather than a quadratic programming (QP) problem.

Algorithm 6 LS-SVM Solver

1: **Input:** Training set $S = \{(x_i, y_i)\}_{i=1}^n$, regularization parameter C
2: **Compute:** Kernel matrix $K = XX^T$
3: Construct the linear system:

$$\begin{bmatrix} 0 & Y^T \\ Y & K + \frac{1}{C}I \end{bmatrix} \begin{bmatrix} b \\ \alpha \end{bmatrix} = \begin{bmatrix} 0 \\ \mathbf{1} \end{bmatrix}$$

4: Solve for α, b
5: Compute weight vector: $w = \sum_{i=1}^n \alpha_i y_i x_i$
6: **Output:** w, b

Computational Complexity

Solving the linear system requires $O(n^3)$ operations. However, using iterative solvers (such as conjugate gradient methods), the cost can be reduced to $O(n^2)$ for large datasets.

Advantages of LS-SVM

- Solves a linear system instead of a QP problem, making it computationally efficient.
- Replaces hinge loss with squared loss, simplifying optimization.
- Supports kernelization, allowing for non-linear classification.
- Efficient for large-scale problems using iterative solvers.

4 SVM Wrappers: OvO and OvR

Support Vector Machines (SVMs) are inherently binary classifiers, designed to separate data into two classes. However, many real-world problems involve multi-class classification, requiring methods to extend SVMs to handle multiple classes. Two widely used strategies for multi-class SVM classification are One-vs-One (OvO) and One-vs-Rest (OvR). These strategies act as wrappers around binary SVM classifiers, enabling them to address multi-class problems.

4.1 One-vs-One (OvO)

Concept: The One-vs-One (OvO) strategy involves training a binary SVM classifier for every pair of classes. Given K classes, this results in $\binom{K}{2} = \frac{K(K-1)}{2}$ binary classifiers. Each classifier is trained to distinguish between two specific classes, ignoring the other classes.

Mathematical Formulation: For each pair of classes (i, j), where $i \neq j$, a binary SVM is trained on the subset of the data belonging to classes i and j. The decision function for the SVM separating these classes is:

$$f_{ij}(x) = \text{sign}(w_{ij}^T \phi(x) + b_{ij}) \tag{2}$$

where w_{ij} is the weight vector, $\phi(x)$ is the feature mapping function, and b_{ij} is the bias term.

Classification: During prediction, each binary classifier votes for one of the two classes it was trained on. The class receiving the most votes across all classifiers is selected as the final prediction. This is known as the "max-wins" strategy.

Use Cases:

- Useful when the number of classes K is relatively small, as the number of classifiers grows quadratically with K.
- Beneficial when binary classifiers are computationally expensive to train, as each classifier is trained on a smaller subset of the data.

Convex Optimization Perspective: Each binary SVM in the OvO approach solves a convex optimization problem:

$$\min_{w_{ij}, b_{ij}} \frac{1}{2}|w_{ij}|^2 + C \sum_{k=1}^{N_{ij}} \xi_k \tag{3}$$

subject to:

$$y_k(w_{ij}^T \phi(x_k) + b_{ij}) \geq 1 - \xi_k, \quad \xi_k \geq 0 \tag{4}$$

where C is the regularization parameter, ξ_k are slack variables, and N_{ij} is the number of training samples for class pair (i, j).

4.2 One-vs-Rest (OvR)

Concept: The One-vs-Rest (OvR) strategy, also known as One-vs-All (OvA), involves training K binary SVM classifiers, one for each class. Each classifier is trained to distinguish one class from all the others combined.

Mathematical Formulation: For each class i, a binary SVM is trained where the positive class consists of samples from class i, and the negative class consists of all other classes $j \neq i$. The decision function is:

$$f_i(x) = \text{sign}(w_i^T \phi(x) + b_i) \tag{5}$$

where w_i is the weight vector and b_i is the bias term.

Classification: During prediction, each binary classifier outputs a confidence score (e.g., the signed distance from the decision boundary). The class with the highest confidence score is selected as the final prediction.

Use Cases:

- More scalable than OvO when the number of classes K is large, as it only requires K classifiers.
- Preferred when binary classifiers are relatively fast to train, as each classifier is trained on the entire dataset.

Convex Optimization Perspective: Each binary SVM in the OvR approach solves:

$$\min_{w_i, b_i} \frac{1}{2}|w_i|^2 + C \sum_{k=1}^{N} \xi_k \tag{6}$$

subject to:

$$y_k(w_i^T \phi(x_k) + b_i) \geq 1 - \xi_k, \quad \xi_k \geq 0 \tag{7}$$

where N is the total number of training samples, and $y_k = +1$ if x_k belongs to class i, otherwise $y_k = -1$.

4.3 Origin and Theoretical Background

The OvO and OvR strategies are general methods for extending binary classifiers to multi-class problems. These strategies originated from the need to handle multi-class classification using binary classifiers, which are simpler and more efficient to train.

OvO is based on pairwise comparison, where each classifier distinguishes between two specific classes. This often results in higher accuracy due to simpler decision boundaries.

OvR separates one class from all others, making it more scalable and easier to implement.

4.4 Comparison and Practical Considerations

- **Accuracy:** OvO often achieves higher accuracy, especially when classes are not well-separated.
- **Computational Complexity:** OvO requires $\mathcal{O}(K^2)$ classifiers, making it less scalable than OvR, which requires $\mathcal{O}(K)$ classifiers.
- **Training Efficiency:** OvR trains each classifier on the entire dataset, making it more memory-intensive but often faster for large-scale problems.

Overall, both approaches have their advantages and trade-offs. The choice between OvO and OvR depends on the specific dataset characteristics, computational constraints, and accuracy requirements of the problem at hand.

5 Dataset, Preprocessing, and PCA

5.1 Dataset Description

For our benchmarking experiments, we use the **MNIST** dataset [7]. MNIST consists of 70 000 grayscale images of handwritten digits, where each image is 28×28 pixels. This results in 784 features per sample. The dataset is well-known in the machine learning community for its balanced classes and standardized format, making it a suitable candidate for comparing different SVM solvers.

5.2 Data Properties and Preprocessing

The MNIST dataset has the following properties:

- **Size:** 70 000 samples
- **Dimensionality:** Originally 784 features per sample.
- **Data Range:** Pixel intensity values typically range from 0 to 255.

Given these properties, preprocessing is essential for efficient optimization:

1. **Standard Scaling:** Each feature is scaled to have zero mean and unit variance. Standard scaling is important because it prevents features with large numeric ranges from dominating the optimization process. This is particularly crucial for gradient-based methods where the scale of the features affects the magnitude of the gradient and, hence, the convergence behavior.
2. **Principal Component Analysis (PCA):** Although MNIST images are 784-dimensional, many of these dimensions are highly correlated. PCA is used to project the data onto a lower-dimensional subspace (e.g., $k = 50$ dimensions) while preserving the majority of the variance. This dimensionality reduction reduces computational complexity, mitigates the curse of dimensionality, and can even improve the generalization performance of the SVM by removing noise and redundant features.

6 Benchmark Experiments

6.1 Setup

- **Dataset:** MNIST, with a 80/20 split of training and testing data.
- **Preprocessing:** We apply standard scaling to ensure all features have a similar range. PCA is then used to reduce the dimensionality from 784 to a lower dimension (e.g., 50), maintaining most of the variance.
- **Solvers:** We compare Gradient Descent, Cutting Plane, CVXPY QP, Scikit-learn SVC, Newton Method, and Pegasos.

- **Multi-class:** Multi-class classification is performed using One-vs-Rest (OvR) wrapper.
- **Metrics:** We measure training time, the number of support vectors (when applicable), and test accuracy.

6.2 Results and Comparison

Table 2 illustrates a hypothetical comparison:

Table 2: Benchmark Results on MNIST (example).

Solver	**Train Time (s)**	**Accuracy (%)**	**# Support Vectors**
Gradient Descent	14.76	82.36	1615
Cutting Plane	12.69	79.52	13130
CVXPY QP	779.09	78.11	2770
Sklearn SVC	552.74	79.10	39204
Newton Method	11.95	83.23	36853
LS-SVM	635.41	80.14	188155
Pegasos	1.61	74.52	503395

Observations:

- *Exact Methods (CVXPY QP):* These solvers provide near-optimal accuracy but are computationally expensive for large datasets.
- *Iterative Methods (Gradient Descent, Pegasos, Cutting Plane):* They scale better and are more efficient when dealing with large-scale data.
- *Pegasos:* Particularly efficient when the regularization parameter λ is chosen appropriately.

Our benchmark experiments on MNIST reveal distinct trade-offs between computational efficiency, predictive performance, and model complexity across solvers. **Newton's method** emerges as a strong performer, achieving the highest test accuracy (83.23%) with relatively fast training (11.95 seconds). This aligns with its theoretical advantage: quadratic convergence ensures fewer iterations to reach a high-quality solution, particularly effective in the reduced 50-dimensional PCA space where Hessian inversion remains tractable. However, its large number of support vectors (36,853) suggests limited sparsity, likely due to the method's focus on minimizing the loss globally rather than enforcing strict margin separation.

Gradient Descent (GD) strikes a favorable balance, delivering near-optimal accuracy (82.36%) while maintaining a compact model (1,615 support vectors) and moderate training time (14.76 seconds). The smaller support vector count reflects GD's implicit bias toward solutions with wider margins, as gradient updates progressively discard non-critical training points. In contrast, **Pegasos**, despite its remarkable speed (1.61 seconds), achieves the lowest accuracy (74.52%) and an implausibly high number of support vectors (503,395). This aligns with its stochastic design: while optimized for scalability, its reliance on random

mini-batches and fixed learning rate may hinder convergence to a precise solution, especially if the regularization parameter λ is suboptimal or the problem is non-separable.

Traditional QP-based solvers (**CVXPY QP**, **Scikit-learn SVC**) exhibit significant computational bottlenecks, with training times exceeding 500 seconds. CVXPY's poor scalability stems from its generic interior-point algorithm, which solves the dual SVM problem with $O(n^3)$ complexity. Scikit-learn's SVC, while tailored for SVMs, struggles with the multi-class wrapper's overhead, accumulating 39,204 support vectors across classes. Both methods prioritize exact solutions over scalability, rendering them impractical for large-scale applications.

The **Cutting Plane** method, though efficient (12.69 seconds), underperforms in accuracy (79.52%) and sparsity (13,130 support vectors). This highlights a limitation of constraint-based approximations: iteratively refining a working set of constraints risks discarding critical support vectors early, leading to suboptimal margins.

As explained the **LS-SVM** is relaxation of the original problem therefore we expect for it to underperforms in accuracy as it does. it also takes alot of time to solve the linear equation proposed by this method$O(n^3)$ where n is number of samples in dataset therefore this is very time consuming algorithm.

The divergence in support vector counts underscores algorithmic biases. Methods like GD and CVXPY QP, which solve the primal or dual problem with exact (or near-exact) constraints, promote sparsity by design. In contrast, Newton's method and Pegasos optimize unconstrained or penalized objectives, allowing more data points to influence the decision boundary. The stark contrast between Pegasos' extreme support vector count and its low accuracy further suggests inadequate regularization or premature stopping, emphasizing the need for careful hyperparameter tuning in stochastic settings.

In summary, **Newton's method** and **Gradient Descent** stand out as the most effective solvers for this task, balancing speed and accuracy. However, the choice between them depends on the application's priorities: Newton for maximal accuracy, GD for sparse models. The poor scaling of exact QP solvers and the instability of stochastic methods like Pegasos reinforce the importance of aligning solver properties with problem scale and structure.

6.3 Theoretical Interpretation

The observed results can be explained through the theoretical underpinnings of each solver. **Newton's method** leverages second-order information (Hessian matrix) to achieve rapid convergence, making it highly accurate but computationally intensive per iteration. Its high number of support vectors arises from its focus on minimizing the global loss, which often includes more data points in the decision boundary. **Gradient Descent**, on the other hand, relies on first-order information, making it less precise per iteration but more scalable. Its implicit regularization promotes sparsity, resulting in fewer support vectors.

Pegasos, a stochastic gradient method, is designed for large-scale problems but suffers from variance in its updates due to mini-batching. This stochasticity can lead to suboptimal solutions if the learning rate or regularization parameter λ is not carefully tuned. The extremely high number of support vectors in Pegasos suggests that many data points are marginally contributing to the decision boundary, a consequence of its unconstrained optimization approach.

Exact QP solvers like **CVXPY QP** and **Scikit-learn SVC** solve the dual SVM problem with high precision but at the cost of scalability. Their $O(n^3)$ complexity makes them impractical for large datasets, and their reliance on multi-class wrappers (e.g., One-vs-Rest) further exacerbates computational demands. The **Cutting Plane** method, while efficient, approximates the solution by iteratively refining constraints, which can lead to suboptimal margins and higher support vector counts.

These theoretical insights highlight the importance of selecting solvers based on the problem's scale, dimensionality, and desired trade-offs between accuracy, sparsity, and computational efficiency.

7 Conclusion

In this paper, we presented a comparative study of several SVM solvers by discussing the derivation of secondary formulations from the primary SVM problem. Each method leverages a different theoretical foundation, whether it is the duality principle, surrogate loss minimization, or second-order optimization. Our experiments on the MNIST dataset—with preprocessing via standard scaling and PCA—highlight the trade-offs between computational speed and model accuracy. This study provides insights into choosing the appropriate solver based on problem size and application requirements.

8 Future Work

There are several promising directions for future research:

- **Mini-batch and Variance Reduction:** Extending stochastic solvers (e.g., Pegasos) to use mini-batch updates or variance-reduced techniques such as SVRG or SAGA.
- **Kernel Methods:** Incorporating approximate kernel techniques (e.g., Random Fourier Features, Nyström method) to enable scalable non-linear SVM training.
- **Parallel and Distributed Training:** Investigating parallel implementations (e.g., parallel coordinate descent) or distributed frameworks for large-scale SVM problems.
- **Adaptive Hyperparameter Tuning:** Developing methods for automatic tuning of learning rates and regularization parameters during training.
- **Integration with Deep Learning:** Combining SVM-based classifiers with deep feature representations to leverage both model interpretability and high-performance classification.

9 References

References

[1] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT*, 2010.

[2] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the ACM SIGKDD*, 2006.

[3] S. Diamond and S. Boyd. CVXPY: A python-embedded modeling language for convex optimization. *Journal of Machine Learning Research*, 17(83):1–5, 2016.

[4] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.* Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[5] J. Nocedal and S. Wright. *Numerical Optimization*. Springer Series in Operations Research, 2006.

[6] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th ICML*, 2007.

[7] Y. LeCun, C. Cortes, and C.J. Burges. The MNIST database of handwritten digits. `http://yann.lecun.com/exdb/mnist/`, 1998.

[8] Chih-Chung Chang and Chih-Jen Lin. Library for Support Vector Machines `https://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf`, 2001.