Chapter 5: Support Vector Machines

Latreche Sara

Contents

1 Introduction to Classification 3

2 Linear Classifiers 4

3 Margins and Motivation 4

4 Primal SVM Formulation 6

5 Dual Formulation 7

6 Dual Formulation 7

7 The Kernel Trick 10

8 Soft Margin and Regularization 13

9 Hinge Loss and Empirical Risk Minimization 14

10 Computational Considerations 14

11 Solution Part 2: SMO Algorithm Instead of Quadratic Solvers 15

12 Exercises **20**

Introduction to Support Vector Machines

Support Vector Machines (SVMs) are one of the most reliable and well-established algorithms in supervised learning. Originally developed by Vladimir Vapnik and later extended by Corinna Cortes and Vapnik in the 1990s, SVMs have become widely used across a variety of domains due to their effectiveness and solid theoretical foundations.

The central idea behind SVMs is to find a decision boundary that not only separates classes but does so with the largest possible gap — also known as the margin — between the classes. This margin-based approach leads to better generalization and more confident predictions.

This chapter introduces the core concepts and techniques behind SVMs. We begin with the motivation for maximizing the margin, then move to the mathematical formulation of the problem. We explore how to derive a dual formulation, which plays a key role in enabling efficient computation.

To deal with more complex, non-linearly separable data, we introduce the kernel trick — a powerful idea that allows SVMs to operate in high-dimensional feature spaces. We also consider soft margin SVMs, which offer flexibility in the presence of noisy or overlapping data. Finally, we discuss the SMO algorithm, which provides a fast and practical way to train SVMs.

The goal of this chapter is to give you both the intuition and the tools to understand how SVMs work and when to apply them effectively.

1 Introduction to Classification

A binary classifier is a function $h : \mathbb{R}^d \to \{-1, +1\}$. Given a dataset of feature vectors $x^{(i)}$ and their labels $y^{(i)} \in \{-1, +1\}$, we aim to find a function h that predicts unseen inputs well.

Question: What is the difference between a training error and a test error?

2 Linear Classifiers

A linear classifier makes predictions based on:

$$h(x) = \text{sign}(w^\top x + b)$$

where $w \in \mathbb{R}^d$, $b \in \mathbb{R}$. The hyperplane $w^\top x + b = 0$ separates the space into two halves.

Study Question: If you flip the sign of w and b, what happens to the decision boundary?

Example

Let $w = \begin{bmatrix} -1 \\ 1.5 \end{bmatrix}$, $b = 3$. Then:

$$h(x^{(1)}) = \text{sign}([-1, 1.5] \cdot [3, 2]^\top + 3) = \text{sign}(3) = +1$$

3 Margins and Motivation

Before we write down any equations, let's ask: **what makes a classifier confident?**

Suppose we have two classes: suns (labeled $+1$) and moons (-1). A good classifier doesn't just separate them correctly—it does so with **confidence**.

Confidence grows when points are **far away from the decision boundary**. The further away a point is from the boundary, the more certain we are about its label. So ideally, we want to place the boundary such that even the closest points to it are still **as far away as possible**.

Let's say we draw a line $w^\top x + b = 0$ to separate the classes. We then ask:

What is the maximum margin we can get while still classifying all training points correctly?

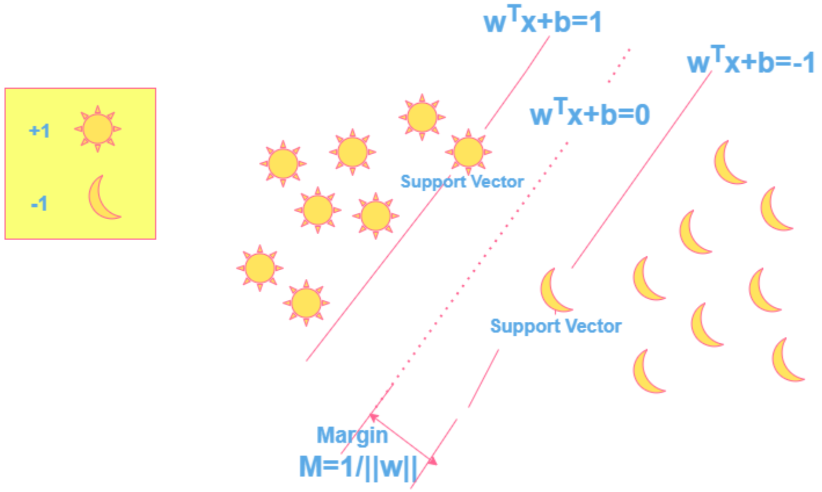

Figure 1: Support vectors lie on the margin; the goal is to push the margin as wide as possible while still separating the classes.

This leads to the SVM objective:

$$\text{maximize margin} \quad \Rightarrow \quad \text{minimize } \frac{1}{2}\|w\|^2$$

subject to:

$$y^{(i)}(w^\top x^{(i)} + b) \geq 1 \quad \forall i$$

This constraint ensures that each point is: - On the correct side of the hyperplane, and - At least distance $\frac{1}{\|w\|}$ away from it

Key Idea

SVM doesn't just separate the data—it chooses the boundary that's farthest away from the closest points. This margin-based reasoning leads to better generalization.

4 Primal SVM Formulation

From our discussion on margins, we saw that maximizing the geometric margin leads to more confident and robust predictions. We now formalize this idea into an optimization problem.

We assume the data is linearly separable, and seek a hyperplane that separates the classes with the **maximum possible margin**. Using a scaling trick (fixing the functional margin to 1), this problem becomes:

$$\min_{w,b} \quad \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y^{(i)}(w^\top x^{(i)} + b) \geq 1 \quad \forall i$$

Optimization Goal

We minimize $\frac{1}{2}\|w\|^2$ to maximize the margin, while ensuring that every training example is correctly classified with at least margin 1.

This is a convex quadratic programming (QP) problem: - The objective $\frac{1}{2}\|w\|^2$ is convex (quadratic, bowl-shaped) - The constraints $y^{(i)}(w^\top x^{(i)}+b) \geq 1$ are linear

Introducing the Lagrangian

To solve this constrained optimization problem, we apply the method of Lagrange multipliers. We introduce one multiplier $\alpha_i \geq 0$ per constraint, and form the Lagrangian:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{n} \alpha_i \left[y^{(i)}(w^\top x^{(i)} + b) - 1 \right]$$

This converts the constrained problem into an unconstrained one over the Lagrangian. To find the optimum, we use the Karush-Kuhn-Tucker (KKT) conditions.

KKT Conditions

The optimal solution (w^*, b^*, α^*) satisfies the following:

(K1) **Stationarity:** $\nabla_w \mathcal{L} = 0$, $\nabla_b \mathcal{L} = 0$

(K2) **Primal feasibility:** $y^{(i)}(w^\top x^{(i)} + b) \geq 1$

(K3) **Dual feasibility:** $\alpha_i \geq 0$

(K4) **Complementary slackness:** $\alpha_i \left[y^{(i)}(w^\top x^{(i)} + b) - 1 \right] = 0$

These conditions are not just theoretical: they will guide us in deriving the dual formulation next, and in designing efficient algorithms like SMO.

5 Dual Formulation

We derive the dual to express the optimization in terms of dot products:

$$\max_\alpha \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y^{(i)} y^{(j)} x^{(i)\top} x^{(j)}$$

$$\text{s.t. } \sum_i \alpha_i y^{(i)} = 0, \quad \alpha_i \geq 0$$

Key Idea: Sparsity

Only a subset of the α_i's are non-zero. These correspond to support vectors — the data points that “matter”.

6 Dual Formulation

In the previous section, we formulated the primal optimization problem for the optimal margin classifier:

$$\min_{w,b} \frac{1}{2}\|w\|^2 \quad \text{subject to} \quad y^{(i)}(w^\top x^{(i)} + b) \geq 1 \quad \forall i$$

This problem is convex and can be solved directly, but there are two important reasons we instead move to its **dual form**:

1. The dual reveals that only a few training points (support vectors) determine the solution.
2. It allows us to express everything in terms of **dot products** between input vectors—critical for enabling the kernel trick later.

Constructing the Dual

We define the Lagrangian by introducing one multiplier $\alpha_i \geq 0$ per constraint:

$$\mathcal{L}(w, b, \alpha) = \frac{1}{2}\|w\|^2 - \sum_{i=1}^{m} \alpha_i \left[y^{(i)}(w^\top x^{(i)} + b) - 1 \right]$$

To form the dual, we minimize $\mathcal{L}$ with respect to w and b while keeping α fixed:

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)} = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

$$\frac{\partial \mathcal{L}}{\partial b} = -\sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Plugging these back into the Lagrangian, and using the fact that the term involving b vanishes, we obtain the dual objective:

$$\mathcal{W}(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y^{(i)} y^{(j)} \langle x^{(i)}, x^{(j)} \rangle$$

subject to:

$$\alpha_i \geq 0, \quad \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Key Idea: The Dual

We transformed a constrained problem over w and b into an unconstrained problem over α, where the data appears only via dot products $\langle x^{(i)}, x^{(j)} \rangle$.

Support Vectors and Sparsity

Thanks to the KKT conditions, only the points that lie exactly on the margin (i.e., those for which the constraint is active) will have $\alpha_i > 0$. These are the **support vectors**.

- Most α_i's are zero
- The final decision boundary depends only on the support vectors

Making Predictions

Once we've solved the dual and found the optimal α^*, we recover the optimal w as:

$$w = \sum_{i=1}^{m} \alpha_i y^{(i)} x^{(i)}$$

To classify a new input x, compute:

$$f(x) = w^\top x + b = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

Since only support vectors have non-zero α_i, this sum is typically over just a few terms.

Summary

By moving to the dual:

- We simplify optimization - Enable the use of kernels (by replacing dot products with kernel functions) - Gain interpretability: only a small subset of points—the support vectors—define the classifier

7 The Kernel Trick

In the dual formulation of SVM, we found that the optimal classifier can be written entirely in terms of inner products between data points:

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} \langle x^{(i)}, x \rangle + b$$

Key Insight: Since the data only appears through dot products $\langle x^{(i)}, x^{(j)} \rangle$, we are free to compute these inner products in some new feature space without ever explicitly computing the feature vectors.

Let $\phi(x)$ be a mapping from the input space to a high-dimensional (possibly infinite-dimensional) feature space:

$$\phi : \mathbb{R}^n \to \mathbb{R}^D$$

Instead of working with x, we work with $\phi(x)$, and define the **kernel function**:

$$K(x, z) = \langle \phi(x), \phi(z) \rangle$$

This allows us to rewrite the classifier as:

$$f(x) = \sum_{i=1}^{m} \alpha_i y^{(i)} K(x^{(i)}, x) + b$$

The Kernel Trick

By computing $K(x, z)$ directly, we can learn in a high-dimensional feature space without ever computing $\phi(x)$. This makes nonlinear classification efficient and scalable.

An Example: Polynomial Kernel

Suppose we define a kernel:

$$K(x, z) = (x^\top z)^2$$

This corresponds to mapping the inputs to a quadratic feature space. For instance, if $x \in \mathbb{R}^3$, then $\phi(x) \in \mathbb{R}^9$, and includes terms like x_1x_2, x_2x_3, etc.

Even though $\phi(x)$ is high-dimensional, we avoid computing it by using the simple dot product $K(x, z) = (x^\top z)^2$.

Another Example: $(x^\top z + c)^2$

Let:

$$K(x, z) = (x^\top z + c)^2$$

This corresponds to a feature space that includes:

$$x_i x_j, \quad \sqrt{2c} x_i, \quad \text{and } c$$

Again, we never compute these features explicitly, yet the SVM operates as if it were learning in this extended space.

Generalization: Kernels and Feature Maps

In general, any kernel function $K(x, z)$ that satisfies Mercer's condition corresponds to an inner product in some feature space. Common kernels include:

- **Polynomial:** $K(x, z) = (x^\top z + c)^d$
- **RBF (Gaussian):** $K(x, z) = \exp(-\gamma \|x - z\|^2)$
- **Sigmoid:** $K(x, z) = \tanh\big(\kappa x^\top z + \theta\big)$

Why This Matters

Thanks to the dual form, the kernel trick allows SVMs to learn highly nonlinear decision boundaries while avoiding the computational cost of explicitly working in high dimensions.

Mercer's Theorem and Valid Kernels

Not every function $K(x, z)$ can be used as a kernel. We need a way to check whether a given K actually corresponds to an inner product in some feature space.

Mercer's Theorem (Informal)

A function $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ is a valid kernel if and only if, for any finite set of inputs $\{x^{(1)}, \ldots, x^{(m)}\}$, the **kernel matrix** $K_{ij} = K(x^{(i)}, x^{(j)})$ is symmetric and positive semi-definite.

Why does this matter? Because it gives us a simple way to verify whether a function is a valid kernel—without needing to find the feature mapping ϕ.

Intuition: - The kernel matrix acts like a generalized dot product matrix. - If it's symmetric and all its eigenvalues are non-negative, then it behaves like an inner product in some space—even if we never know what that space is!

Applications: - For image recognition, SVMs with polynomial or RBF kernels perform remarkably well even when the input is just raw pixel data (e.g., 16x16 digits). - For string data (e.g., proteins), defining $\phi(x)$ to count k-length substrings can lead to extremely high-dimensional feature vectors—yet we can compute $K(x, z) = \langle \phi(x), \phi(z) \rangle$ efficiently using string matching algorithms.

Broader Impact

The kernel trick works beyond SVMs. Any algorithm that depends only on inner products—like the perceptron or PCA—can be "kernelized" by replacing $\langle x, z \rangle$ with $K(x, z)$.

8 Soft Margin and Regularization

Real-world data often isn't perfectly separable. Even one outlier can distort the decision boundary and shrink the margin drastically.

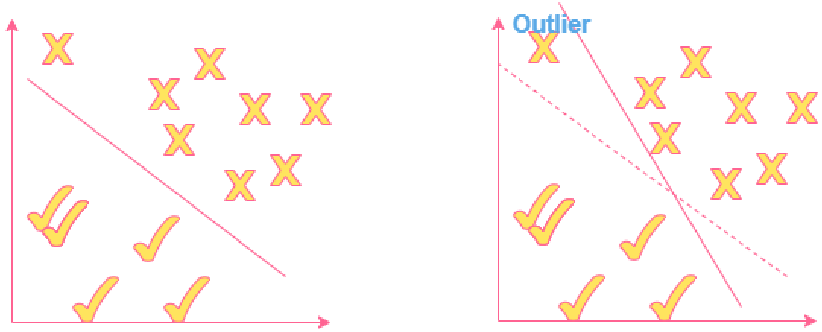

Figure 2: Left: ideal margin without outliers. Right: a single outlier shifts the boundary and reduces margin.

To make SVMs more robust, we introduce **slack variables** $\xi_i \geq 0$ that allow some margin violations. This leads to the **soft margin SVM**:

$$\min_{w,b,\xi} \quad \frac{1}{2}\|w\|^2 + C \sum \xi_i \quad \text{s.t. } y^{(i)}(w^\top x^{(i)} + b) \geq 1 - \xi_i$$

- The term ξ_i measures how much point $x^{(i)}$ violates the margin. - The parameter C controls how harshly violations are penalized.

In the dual form, this adds a new constraint:

$$0 \leq \alpha_i \leq C$$

This simple change lets SVMs tolerate overlap and noise while still focusing on finding a large-margin separator.

9 Hinge Loss and Empirical Risk Minimization

The SVM objective can be seen as minimizing the **hinge loss**:

$$\ell_{\text{hinge}}(y, f(x)) = \max(0, 1 - yf(x))$$

Hinge loss is a convex upper bound on 0–1 loss and leads to a tractable optimization problem.

10 Computational Considerations

SVM training involves solving a quadratic programming (QP) problem. Several algorithms exist:

- **SMO (Sequential Minimal Optimization)**: breaks the QP into 2D subproblems
- **Kernel cache + decomposition** for large datasets
- `scikit-learn`'s `SVC` uses LIBSVM underneath

11 Solution Part 2: SMO Algorithm Instead of Quadratic Solvers

To solve the SVM dual optimization problem efficiently, we avoid using general-purpose quadratic programming solvers and use a specialized method: **Sequential Minimal Optimization (SMO)**.

6.1 Coordinate Ascent

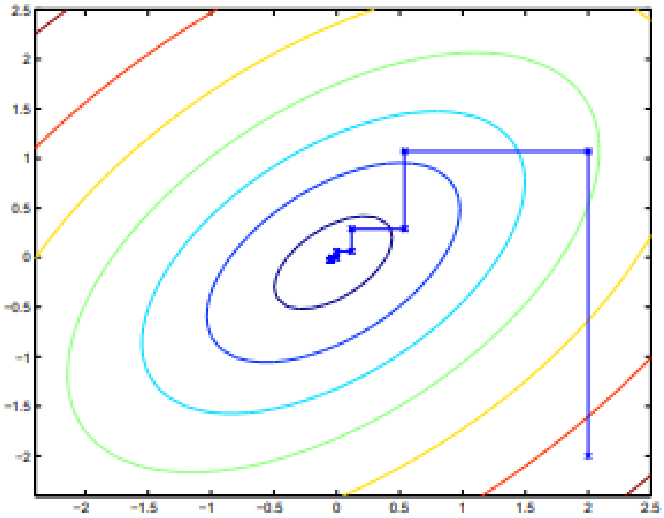

Figure 3: Coordinate ascent: each step follows a direction aligned with one axis.

And here is an example of coordinate ascent in action: Notice that in Coordinate ascent, each step only changes one variable while holding the others fixed.

If our goal is just to solve an unconstrained optimization problem:

$$\max_{\alpha} \quad W(\alpha_1, \ldots, \alpha_m)$$

The W here is just some function of the parameters α's. To solve this optimization, the idea is that we only choose one parameter, let's say $\hat{\alpha}_i$, and hold all variables α's except α_i. So we can only optimize W with respect to just the parameter $\hat{\alpha}_i$.

The coordinate ascent algorithm:

Coordinate Ascent Algorithm

```
Loop until convergence:
For $i = 1, \ldots, m$:
$\alpha_i \leftarrow \arg\max_{\hat{\alpha}_i} W(\alpha_1, \ldots, \alpha_{i-1}, \hat{\alpha}_i, \alpha_{i+1}, \ldots, \alpha_m)$
```

6.2 SMO

Our dual optimization problem is:

$$\max_{\alpha} \quad W(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} y^{(i)} y^{(j)} \alpha_i \alpha_j x^{(i)\top} x^{(j)}$$

subject to:

$$0 \leq \alpha_i \leq C, \quad \text{for } i = 1, \ldots, m \quad \text{and} \quad \sum_{i=1}^{m} \alpha_i y^{(i)} = 0$$

Now suppose we try to update just a single parameter, say α_1. From the equality constraint, we get:

$$\alpha_1 = -y^{(1)} \sum_{i=2}^{m} \alpha_i y^{(i)}$$

This means α_1 is determined by the rest, so we cannot freely update it — we're stuck.

Key Insight

To satisfy the equality constraint, we must update **two** parameters at once. This leads to the core idea of the SMO algorithm.

Basic SMO Loop

SMO High-Level Procedure

```
Repeat until convergence:
Select two parameters $\alpha_i$ and $\alpha_j$ (with $j \neq i$)
Optimize $W(\alpha)$ with respect to $\alpha_i$ and $\alpha_j$,
while holding all other parameters fixed.
```

This reduces the large constrained QP into a 2D subproblem, which we can solve analytically while respecting:

- the box constraints $0 \leq \alpha_i, \alpha_j \leq C$, - and the equality constraint $\alpha_i y^{(i)} + \alpha_j y^{(j)} =$ constant.

For more details, refer to:

- Platt, J. (1998). *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines.*

6.3 SMO Pseudocode

The following pseudocode outlines a practical SMO implementation that alternates between choosing pairs of α_i, α_j and updating them efficiently.

Main SMO Loop

```
numChanged $\leftarrow 0$
examineAll $\leftarrow$ True
while (numChanged > 0 or examineAll is True):
if examineAll:
for all $i$ in training set:
numChanged $+=$ examineExample($i$)
else:
for all $i$ where $0 < \alpha_i < C$:
numChanged $+=$ examineExample($i$)
if examineAll:  examineAll $\leftarrow$ False
else if (numChanged == 0):  examineAll $\leftarrow$ True
```

Function: examineExample(i)

```
Get $y_i$, $\alpha_i$, $E_i$
$r_i = y_i E_i$
If  $(r_i < -\texttt{tol} \wedge \alpha_i < C)$ or $(r_i > \texttt{tol} \wedge \alpha_i > 0)$:
If more than 1 non-bound $\alpha$:
Choose $j = \arg\max_j |E_i - E_j|$
If takeStep(i, j):  return 1
Loop over random non-bound $j$, try takeStep(i, j)
Loop over all $j$, try takeStep(i, j)
return 0
```

Function: `takeStep(i, j)`

```
If  $i = j$: return False
Get $y_i, y_j, \alpha_i, \alpha_j, E_i, E_j$
$s = y_i y_j$
If  $y_i \neq y_j$:    $L = \max(0, \alpha_j - \alpha_i)$, $H = \min(C, \alpha_j - \alpha_i + C)$
Else:    $L = \max(0, \alpha_i + \alpha_j - C)$, $H = \min(C, \alpha_i + \alpha_j)$
Compute $\eta = K_{ii} + K_{jj} - 2K_{ij}$
If  $\eta \leq 0$: return False
        $\alpha_j^{\text{new}} = \alpha_j + \frac{y_j(E_i - E_j)}{\eta}$   clip to $[L, H]$
If  change in $\alpha_j$ is too small: return False
        $\alpha_i^{\text{new}} = \alpha_i + s(\alpha_j - \alpha_j^{\text{new}})$
Update bias $b$, return True
```

Compute thresholds:

$$b_1 = b - E_i - y_i(\alpha_i^{\text{new}} - \alpha_i)K(x_i, x_i) - y_j(\alpha_j^{\text{new}} - \alpha_j)K(x_i, x_j)$$

$$b_2 = b - E_j - y_i(\alpha_i^{\text{new}} - \alpha_i)K(x_i, x_j) - y_j(\alpha_j^{\text{new}} - \alpha_j)K(x_j, x_j)$$

Update bias b based on which α's are within bounds:

$$\text{If } 0 < \alpha_i^{\text{new}} < C : \quad b \leftarrow b_1$$
$$\text{Else if } 0 < \alpha_j^{\text{new}} < C : \quad b \leftarrow b_2$$
$$\text{Else: } b \leftarrow \frac{b_1 + b_2}{2}$$

12 Exercises

To reinforce your understanding of Support Vector Machines, attempt the following exercises. They are divided into two levels: conceptual derivation, and hands-on numerical intuition.

Level 1: Conceptual Understanding and Derivations

1. **Margin Geometry:** Prove that the geometric margin is given by $\frac{2}{\|w\|}$.
2. **Dual Formulation:** Derive the dual of the soft-margin SVM using Lagrangian multipliers and KKT conditions.
3. **Kernels:** Show that $K(x,z) = (x^\top z)^2$ corresponds to a feature mapping $\phi(x) \in \mathbb{R}^9$. List the components of $\phi(x)$ explicitly for $x \in \mathbb{R}^3$.
4. **Hinge Loss:** Explain why the hinge loss is a convex upper bound of the 0–1 loss.
5. **KKT Conditions:** For a trained SVM, explain what it means if $0 < \alpha_i < C$, and how this relates to support vectors.

Level 2: Numerical Application and Visualization

6. **Hand Calculation of a Linear SVM:** Given a small 2D dataset (3–4 points), compute w, b, and the margin by hand for the hard-margin SVM.
7. **Compare Margins:** Generate synthetic 2D datasets and train both hard and soft margin SVMs. Plot the decision boundaries and margins.
8. **Hyperplane Equation:** Given an SVM solution w and b, write the hyperplane equation and verify which points are on the margin, inside it, or correctly classified.
9. **RBF Kernel on XOR:** Train an SVM with an RBF kernel on the XOR problem. Plot the resulting non-linear decision boundary.

10. **Grid Search for C:** Use a cross-validation grid search to study the effect of different values of C on training and test error.
11. **SMO in Practice:** Implement a basic version of the SMO algorithm on a toy dataset. Compare results with ‘sklearn.SVC‘.

References:

- Platt, John (1998), *Sequential Minimal Optimization: A Fast Algorithm for Training Support Vector Machines*. Microsoft Research. https://www.microsoft.com/.../sequential-minimal-optimization
- CS229 Notes: `http://cs229.stanford.edu/materials/smo.pdf`
- Lucien East (2022), *Implement SVM with SMO from Scratch*. `https://lucien-east.github.io/2022/07/30/Implement-SVM-with-SMO-from-scratch`
- An Introduction to Support Vector Machine (SVM) and SMO Algorithm. `https://www.codeproject.com/Articles/1267445/An-Introduction-to-Support-Vector-Machine-SVM-and`