CPE 490 590: Machine Learning for Engineering Applications

11 Support Vector Machine

Rahul Bhadani
Electrical & Computer Engineering, The University of Alabama in Huntsville

Outline

Announcement

Homework 3

- Due: March 9, 2025
- Advanced Regression, Logistics Regression, and Neural Network

Constraint Optimization with Inequality

Inequality Constraints

$$\max / \min f(x)$$
$$\text{subject to}$$
$$g(x) \geq 0$$

Without the constraint, $\frac{df(x^*)}{dx} = 0$.

When the inequality constraint is added in, either the solution could occur when $g(x^*) = 0$, or $g(x^*) > 0$.

Inequality Constraints with Lagrange Multiplier

We could still write the Lagrangian:

$$\mathcal{L}(x, \lambda) = f(x) + \lambda g(x)$$

$$\frac{\partial \mathcal{L}}{\partial x}(x^*, \lambda^*) = \frac{\partial f}{\partial x}(x^*) + \frac{\partial g(x^*)'}{\partial x}\lambda^* = 0$$

$$\frac{\partial \mathcal{L}}{\partial \lambda}(x^*, \lambda^*) = g(x^*) \geq 0$$

$$\Rightarrow \lambda^* \geq 0, \quad \lambda^* \odot \frac{\partial \mathcal{L}}{\partial \lambda}(x^*, \lambda^*) = \lambda^* \odot g(x^*) = 0$$

where $\odot$ is the element-wise product.

These conditions are called **Karush-Kuhn-Tucker Conditions** (KKT).

Karush-Kuhn-Tucker Conditions

- If we have several equality and inequality constraints, in the following form

$$\arg\min_x f(x), \text{ s.t. } g_i(x) = 0, h_j(x) \leq 0$$

The necessary and sufficient conditions (also known as the KKT conditions) for

$$\frac{\partial L(x^*, \lambda^*, \beta^*)}{\partial x} = 0, \quad \frac{\partial L(x^*, \lambda^*, \beta^*)}{\partial \beta} = 0, \quad g_i(x^*) = 0, \quad h_j(x^*) \leq 0,$$

$$\sum_{j=1}^{J} \beta_j^* h_j(x^*) = 0, \quad \lambda_i, \beta_j \geq 0$$

The last condition is sometimes written in the equivalent form: $\beta_j^* h_j(x^*) = 0$.

The Generalized Lagrangian

The generalized Lagrangian function to minimize $f(x)$ subject to the equality constraints on $g_i(x)$ and inequality constraints on $h_j(x)$ is given by:

$$L(x, \lambda, \beta) = f(x) + \sum_{i=1}^{I} \lambda_i g_i(x) + \sum_{j=1}^{J} \beta_j h_j(x)$$

Example 1

$$\min f(\mathbf{x}) = x_1^2 + x_2^2$$

subject to

$$x_1 + x_2 = 1$$
$$x_2 \geq \alpha$$

Example 1

Class Work

Example 1

Class Work

Example 1

Class Work

Example 2

$$\min f(\mathbf{x}) = (x_1 - 2)^2 + (x_2 - 1)^2$$

subject to

$$h_1(\mathbf{x}) = x_1^2 - x_2 \leq 0$$
$$h_2(\mathbf{x}) = x_1 + x_2 - 2 \leq 0$$

Example 2

Class Work

Example 2

Class Work

Example 2

Class Work

Example 2

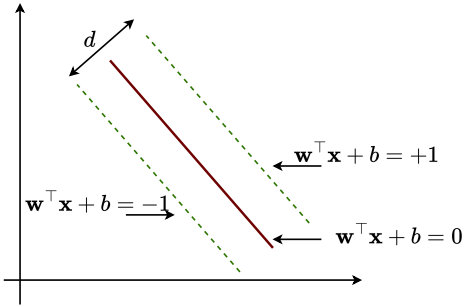
Class Work

Example 2

Class Work

Support Vector Machine

Motivating SVM

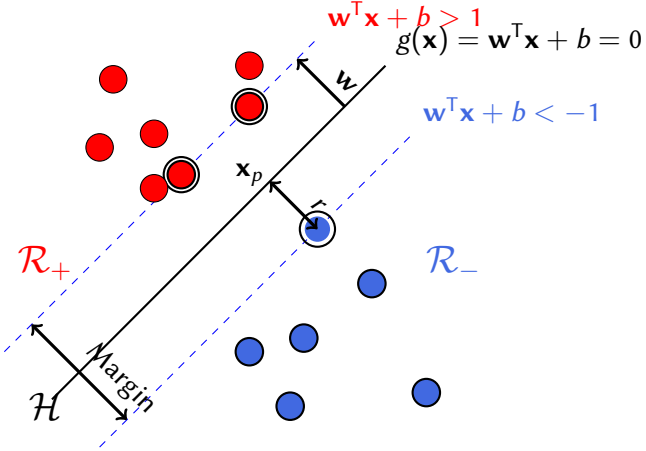


A linear support vector machine (SVM) aims to find a decision plane $\mathbf{w}^\top \mathbf{x} + b = 0$ that maximizes the margin of separation.

Assume that all data points satisfy the constraints:

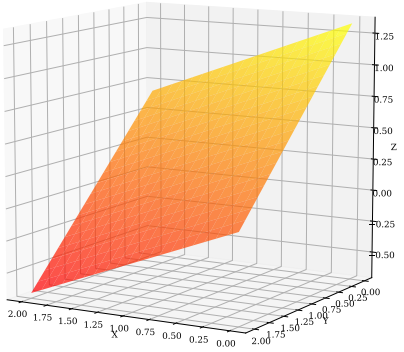
- $\mathbf{w}^\top \mathbf{x} + b \geq +1$ where $y_i = +1$
- $\mathbf{w}^\top \mathbf{x} + b \leq -1$ where $y_i = -1$

The SVM and Linear Classification



Equation for a Hyperplane

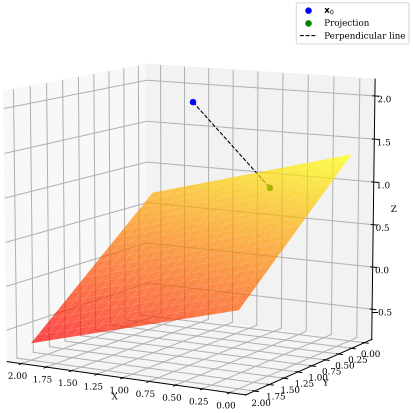
$$\mathbf{w}^\top \mathbf{x} = \beta$$



Euclidean Projection of a Point on a Plane

We project a point $\mathbf{x}_0$ onto the hyperplane $\mathbf{w}^\top \mathbf{x} = \beta$.
The projection point $\mathbf{x}^*$ on the plane is give by

$$\mathbf{x}^* = \mathbf{x}_0 + (\beta - \mathbf{w}^\top \mathbf{x}_0) \frac{\mathbf{w}}{||\mathbf{w}||^2}$$



Euclidean Projection as an Optimization Problem

One way to come up with the above equation is to solve

$$\min \frac{1}{2}||\mathbf{x} - \mathbf{x}_0||^2$$

subject to

$$\mathbf{w}^\top \mathbf{x} = \beta$$

Distance between two Hyperplanes

Consider two Hyperplanes :
$\mathcal{H}_1 : \mathbf{w}^\top \mathbf{x} = b_1$
$\mathcal{H}_2 : \mathbf{w}^\top \mathbf{x} = b_2$
The distance between two hyperplanes is computed by projecting any point $\mathbf{x}_1$ that is already on one plane $\mathcal{H}_1$ onto $\mathcal{H}_2$.

$$\mathbf{x}_2 = \mathbf{x}_1 + (b_2 - \mathbf{w}^\top \mathbf{x}_0) \frac{\mathbf{w}}{||\mathbf{w}||^2}$$

$$\mathbf{x}_2 = \mathbf{x}_1 + (b_2 - b_1) \frac{\mathbf{w}}{||\mathbf{w}||^2}$$

Distance between two Hyperplanes

Hence, the distance between two hyperplanes is given by

$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \left|\left|(b_2 - b_1)\frac{\mathbf{w}}{||\mathbf{w}||^2}\right|\right|$$

$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \frac{|b_2 - b_1|}{||\mathbf{w}||}$$

Formulating SVM Problem: Hyperplane Distance

In SVM problem, if we have

- $\mathbf{w}^\top \mathbf{x} + b = +1$
- $\mathbf{w}^\top \mathbf{x} + b = -1$

Let $b_1 = 1 - b$ and $b_2 = -1 - b$.

Then,

$$d = ||\mathbf{x}_2 - \mathbf{x}_1|| = \frac{|b_2 - b_1|}{||\mathbf{w}||}$$
$$= \frac{|-1 - b - 1 + b|}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

Formulating SVM Problem: Hyperplane Distance

Hence, the distance between two hyperplanes is given by We want to maximize d, which is equivalent to minimizing $\frac{||\mathbf{w}||}{2}$.

However $||\mathbf{w}||$ is not convex, so instead we minimize $\frac{||\mathbf{w}||^2}{2}$ which is convex.

Final Optimization Problem for SVM

The conversation on constrained optimization was all about minimization tasks and our SVM formulation is a maximization problem. We can convert our maximization task to a minimization one by

$$\mathbf{w}^* = \arg\min \frac{1}{2}\|\mathbf{w}\|_2^2$$
$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 \quad \% \; 1 - y_i(\mathbf{w}^T\mathbf{x}_i + b) \leq 0$$

Lagrangian for SVM

The Lagrangian function for SVM is given by

$$\mathcal{L}(\mathbf{w}, \alpha) = \frac{1}{2}\|\mathbf{w}\|_2^2 - \sum_{i=1}^{n} \alpha_i \left[y_i(\mathbf{w}^T \mathbf{x}_i + b) - 1 \right]$$

Deriving the Support Vector Machine

Our goal is to find the $\mathbf{w}$ and b that minimize this function. Therefore, we must compute the derivative of L w.r.t. $\mathbf{w}$ then set it equal to zero. The result of this derivative is given by

$$\frac{\partial L(\alpha)}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \quad \rightarrow \quad \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$$

Deriving the Support Vector Machine

Note that by setting the gradient equal to zero that we were able to find an expression for $\mathbf{w}$! Now the derivative of L w.r.t. b can be be compute then set to zero, which gives us

$$\frac{\partial L(\alpha)}{\partial b} = \sum_{i=1}^{n} \alpha_i y_i \quad \rightarrow \quad \sum_{i=1}^{n} \alpha_i y_i = 0$$

Observation: By taking the derivative w.r.t. b, we ended up getting a constraint!

Deriving the Support Vector Machine

We can simplify L by substituting in to L what we know about $\mathbf{w}$ and the sum of $\alpha^T\mathbf{y}$.

$$\mathcal{L}(\alpha) =$$

Deriving the Support Vector Machine

We can simplify L by substituting in to L what we know about $\mathbf{w}$ and the sum of $\alpha^T\mathbf{y}$.

$$\mathcal{L}(\alpha) = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n}\alpha_i y_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i$$

Deriving the Support Vector Machine

We can simplify L by substituting in to L what we know about $\mathbf{w}$ and the sum of $\alpha^T\mathbf{y}$.

$$\mathcal{L}(\alpha) = \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n}\alpha_i y_i \mathbf{w}^T\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i$$

$$= \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^T\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i$$

Deriving the Support Vector Machine

We can simplify L by substituting in to L what we know about $\mathbf{w}$ and the sum of $\alpha^\mathsf{T}\mathbf{y}$.

$$\begin{aligned}
\mathcal{L}(\alpha) &= \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j - \sum_{i=1}^{n}\alpha_i y_i \mathbf{w}^\mathsf{T}\mathbf{x}_i - b\sum_{i=1}^{n}\alpha_i y_i + \sum_{i=1}^{n}\alpha_i \\
&= \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j - \sum_{i=1}^{n}\sum_{j=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j + \sum_{i=1}^{n}\alpha_i \\
&= \sum_{i=1}^{n}\alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n}\alpha_i\alpha_j y_i y_j \mathbf{x}_i^\mathsf{T}\mathbf{x}_j
\end{aligned}$$

Summary of the SVM so far

- The expression below is known as the *dual form* of the constrained optimization task, which is maximized over α. Note that $\mathbf{x}_i^\top \mathbf{x}_j$ is a measure of similarity between two vectors, so we replace it with a function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \text{ s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, \alpha_i \geq 0$$

Summary of the SVM so far

- The expression below is known as the *dual form* of the constrained optimization task, which is maximized over α. Note that $\mathbf{x}_i^\mathsf{T}\mathbf{x}_j$ is a measure of similarity between two vectors, so we replace it with a function $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2}\sum_{j=1}^{n}\sum_{i=1}^{n} \alpha_i\alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \text{ s.t. } \sum_{i=1}^{n} y_i\alpha_i = 0, \alpha_i \geq 0$$

- Let $\Phi(\mathbf{x})$ be a high-dimensional nonlinear representation of $\mathbf{x}$.

$$\mathbf{w}^\mathsf{T}\Phi(\mathbf{x}) + b = \left(\sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i)\right)^\mathsf{T} \Phi(\mathbf{x}) + b = \sum_{i=1}^{n} \alpha_i y_i \Phi(\mathbf{x}_i)^\mathsf{T}\Phi(\mathbf{x}) + b$$

where $\kappa(\mathbf{x}_i, \mathbf{x}_j) = \Phi(\mathbf{x}_i)^\mathsf{T}\Phi(\mathbf{x}_j)$. Every time the data show up in the expression they are dot products.

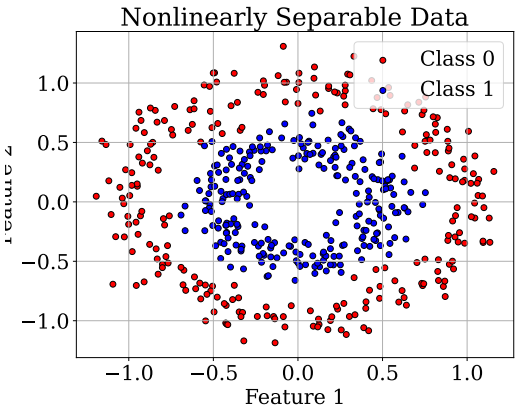
Nonlinear Support Vector Machine and Kernel

Nonlinear Separable Data

Most real-world problems exhibit data that are not linearly separable.

Nonlinear Separable Data

How do we create a separation for ?



Kernel for Nonlinear Separation

Solution is: transform the original variable(s). The function ϕ used to transform the original variables (or features) is called **Kernel**.

What are kernels?

In layman's terms:

What are kernels?

In layman's terms: A kernel is a measure of similarity between two patterns $\mathbf{x}$ and $\mathbf{x}'$

What are kernels?

In layman's terms: A kernel is a measure of similarity between two patterns $\mathbf{x}$ and $\mathbf{x}'$

- Consider a similarity measure of the form:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$(\mathbf{x}, \mathbf{x}') \mapsto \kappa(\mathbf{x}, \mathbf{x}')$$

What are kernels?

In layman's terms: A kernel is a measure of similarity between two patterns $\mathbf{x}$ and $\mathbf{x}'$

- Consider a similarity measure of the form:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$(\mathbf{x}, \mathbf{x}') \mapsto \kappa(\mathbf{x}, \mathbf{x}')$$

- $\kappa(\mathbf{x}, \mathbf{x}')$ returns a real valued quantity measuring the similarity between $\mathbf{x}$ and $\mathbf{x}'$

What are kernels?

In layman's terms: A kernel is a measure of similarity between two patterns $\mathbf{x}$ and $\mathbf{x}'$

- Consider a similarity measure of the form:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$(\mathbf{x}, \mathbf{x}') \mapsto \kappa(\mathbf{x}, \mathbf{x}')$$

- $\kappa(\mathbf{x}, \mathbf{x}')$ returns a real valued quantity measuring the similarity between $\mathbf{x}$ and $\mathbf{x}'$
- One simple measure of similarity is the *canonical dot product*
  - computes the cosine of the angle between two vectors, provided they are normalized to length 1
  - dot product of two vectors form a *pre-Hilbert space*

What are kernels?

In layman's terms: A kernel is a measure of similarity between two patterns $\mathbf{x}$ and $\mathbf{x}'$

- Consider a similarity measure of the form:

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$$
$$(\mathbf{x}, \mathbf{x}') \mapsto \kappa(\mathbf{x}, \mathbf{x}')$$

  - $\kappa(\mathbf{x}, \mathbf{x}')$ returns a real valued quantity measuring the similarity between $\mathbf{x}$ and $\mathbf{x}'$
  - One simple measure of similarity is the *canonical dot product*
    - computes the cosine of the angle between two vectors, provided they are normalized to length 1
    - dot product of two vectors form a *pre-Hilbert space*

- Kernels **represent** patterns in some dot product space $\mathcal{H}$

$$\Phi : \mathcal{X} \to \mathcal{H}$$

Feature Space: $\mathcal{X} \quad \rightarrow \quad \mathcal{H}$

A few notes about our mapping into $\mathcal{H}$ via Φ

1. Mapping lets us define a similarity measure from the dot product in $\mathcal{H}$

$$k(\mathbf{x}, \mathbf{x}') := \Phi(\mathbf{x})^{\mathsf{T}} \Phi(\mathbf{x}')$$

Feature Space: $\mathcal{X} \quad \rightarrow \quad \mathcal{H}$

A few notes about our mapping into $\mathcal{H}$ via Φ

1. Mapping lets us define a similarity measure from the dot product in $\mathcal{H}$

   $$k(\mathbf{x}, \mathbf{x}') := \Phi(\mathbf{x})^{\mathsf{T}} \Phi(\mathbf{x}')$$

2. Patterns are dealt with geometrically; efficient learning algorithms may be applied using linear algebra

Feature Space: $\mathcal{X} \rightarrow \mathcal{H}$

A few notes about our mapping into $\mathcal{H}$ via Φ

1. Mapping lets us define a similarity measure from the dot product in $\mathcal{H}$

   $$k(\mathbf{x}, \mathbf{x}') := \Phi(\mathbf{x})^{\mathsf{T}} \Phi(\mathbf{x}')$$

2. Patterns are dealt with geometrically; efficient learning algorithms may be applied using linear algebra
3. The selection of $\Phi(\cdot)$ leads to a large selection of similarity measures

A simple kernel example

Consider the following dot product:

$$\begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix}^\mathsf{T} \begin{bmatrix} x_1^2 \\ \sqrt{2}x_1x_2 \\ x_2^2 \end{bmatrix} = x_1^2x_1^2 + 2x_1^2x_2^2 + x_2^2x_2^2$$

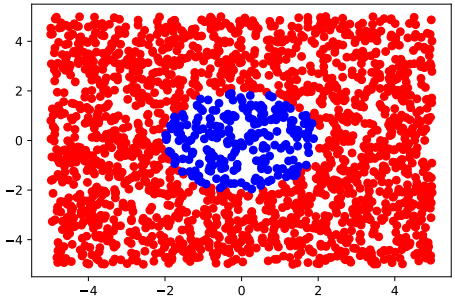
$$= (x_1^2 + x_2^2)^2$$

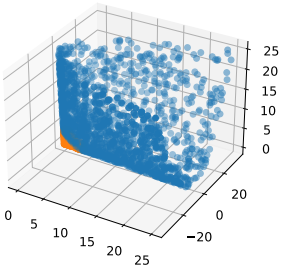
$$= \left[\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}^\mathsf{T} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \right]^2$$

$$= (\mathbf{x}^\mathsf{T}\mathbf{x})^2$$

$$\Phi(\mathbf{x})^\mathsf{T}\Phi(\mathbf{x}) = \kappa(\mathbf{x}, \mathbf{x})$$

A simple kernel example





Gaussian kernels

The Gaussian kernel is defined as

$$k(\mathbf{x}, \mathbf{x}') = \exp\{(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)\}$$

The term in the exponent $(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$ is a scaler value computed in the vector space of the data. Recall the dot product for two arbitrary vectors is given by

$$\mathbf{x}^T\mathbf{x}' = \sum_{i=1}^{d} x_i x_i'$$

All we need to show the calculation of the Gaussian kernel has an infinite sum.

$$\Phi(\mathbf{x})^\top\Phi(\mathbf{x}') = k(\mathbf{x}, \mathbf{x}') = \exp\{(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)\} = \sum_{n=0}^{\infty} \frac{(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)^n}{n!}$$

The Kernel Trick

- The "trick" in each of the kernel examples is that we never needed to calculate $\Phi(\mathbf{x})$. Instead, we applied a nonlinear function to a dot product and this was equivalent to the dot product in a high dimensional space.
  - *This is known as the kernel trick!*
  - Kernels allow us to solve nonlinear classification problems by taking advantage of Cover's Theorem without the overhead of finding the high dimensional space.

Other popular kernel choices

Radial Basis Kernel

$$\kappa(\mathbf{x}, \mathbf{z}) = \exp\{(-\gamma\|\mathbf{x} - \mathbf{z}\|^2)\}$$

Polynomial Kernel

$$\kappa(\mathbf{x}, \mathbf{z}) = (\alpha\mathbf{x}^T\mathbf{z} + \beta)^p$$

Hyperbolic Tangent Kernel

$$\kappa(\mathbf{x}, \mathbf{z}) = \tanh(\alpha\mathbf{x}^T\mathbf{z} + \beta)$$

Fuzzy Separation: Soft Constraints

Soft Margin for SVM

There may be the cases where some data patterns might not be separable by a linear hyperlane due to outliers, noise, or weak nonlinearity. In such a case, we may afford some misclassification if we want to use linear separation.

Slack Variable

We introduce slack variable such that

$$y_i(\mathbf{w}^\top \mathbf{x}_i + b) \geq 1 - \varepsilon_i$$

Optimization Problem with Fuzzy Separation

A New Formulation

We can modify the objective of maximizing the margin by introducing *slack variables* to meet the constraints

$$\arg\min \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\varepsilon_i,$$

$$\text{s.t. } y_i(\mathbf{w}^T\mathbf{x}_i + b) \geqslant 1 - \varepsilon_i, \quad \varepsilon_i \geqslant 0$$

where ε_i is nonnegative if the constraint cannot be met.

Deriving the Soft Margin SVM

The Lagrangian function is given by

$$L(\mathbf{w}, \alpha, \mu) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\mu_i\xi_i - \sum_{i=1}^{n}\alpha_i\left[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right]$$

where α_i and μ_i are Lagrange multipliers.

Deriving the Soft Margin SVM

The Lagrangian function is given by

$$L(\mathbf{w}, \alpha, \mu) = \frac{1}{2}\|\mathbf{w}\|_2^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\mu_i\xi_i - \sum_{i=1}^{n}\alpha_i\left[y_i(\mathbf{w}^T\mathbf{x}_i + b) - 1 + \xi_i\right]$$

where α_i and μ_i are Lagrange multipliers.

Our approach is the same as it was with the hard margin SVM. We have the Lagrangian function and we're going to final the dual form by finding

$$\frac{\partial L}{\partial \mathbf{w}}, \quad \frac{\partial L}{\partial b}, \quad \frac{\partial L}{\partial \xi}$$

Deriving the Soft Margin SVM

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \quad \% \text{ same result as before}$$

Substitute these findings back into and remember that $\mu_i \xi_i = 0$ from the KKT conditions.

Deriving the Soft Margin SVM

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \quad \% \text{ same result as before}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \longrightarrow C = \alpha_i + \mu_i \quad \% \ \alpha_i \text{ is less than } C$$

Substitute these findings back into and remember that $\mu_i \xi_i = 0$ from the KKT conditions.

Deriving the Soft Margin SVM

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i = 0 \longrightarrow \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \quad \% \text{ same result as before}$$

$$\frac{\partial L}{\partial \xi_i} = C - \alpha_i - \mu_i = 0 \longrightarrow C = \alpha_i + \mu_i \quad \% \ \alpha_i \text{ is less than } C$$

$$\frac{\partial L}{\partial b} = -\sum_{i=1}^{n} \alpha_i y_i = 0 \longrightarrow \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \% \text{ same result as before}$$

Substitute these findings back into and remember that $\mu_i \xi_i = 0$ from the KKT conditions.

Deriving the Soft Margin SVM

$$L(\alpha) =$$

Deriving the Soft Margin SVM

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \mu_i \xi_i$$

Deriving the Soft Margin SVM

$$
\begin{aligned}
L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \mu_i \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \xi_i (\alpha_i + \mu_i)
\end{aligned}
$$

Deriving the Soft Margin SVM

$$
\begin{aligned}
L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \mu_i \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \xi_i (\alpha_i + \mu_i) \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - C \sum_{i=1}^{n} \xi_i
\end{aligned}
$$

Deriving the Soft Margin SVM

$$
\begin{aligned}
L(\alpha) &= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i \xi_i - \sum_{i=1}^{n} \mu_i \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \xi_i (\alpha_i + \mu_i) \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j + C \sum_{i=1}^{n} \xi_i - C \sum_{i=1}^{n} \xi_i \\
&= \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^\top \mathbf{x}_j \quad \% \text{ same result as the hard margin}
\end{aligned}
$$

The Soft Margin SVM

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \text{ s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, 0 \leq \alpha_i \leq C$$

- The objective function we end up with the soft margin SVM is nearly identical to the hard margin. The only different is that the Lagrange multiplier, α_i, is now bounded by C.

The Soft Margin SVM

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \text{ s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, 0 \leq \alpha_i \leq C$$

- The objective function we end up with the soft margin SVM is nearly identical to the hard margin. The only different is that the Lagrange multiplier, α_i, is now bounded by C.
- The optimization problem might look difficult; however, this is a *quadratic program* and there are optimizers available to solve this problem. See `CVXOPT` or `PyCVX` in Python.

The Soft Margin SVM

$$\arg\max_{\alpha} \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{j=1}^{n} \sum_{i=1}^{n} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j), \text{ s.t. } \sum_{i=1}^{n} y_i \alpha_i = 0, 0 \leq \alpha_i \leq C$$

- The objective function we end up with the soft margin SVM is nearly identical to the hard margin. The only different is that the Lagrange multiplier, α_i, is now bounded by C.
- The optimization problem might look difficult; however, this is a *quadratic program* and there are optimizers available to solve this problem. See `CVXOPT` or `PyCVX` in Python.
- Many of the α_i's will be zero after solving the QP. The data samples that correspond to non-zero values of α_i are known as the *support vectors*.

References

Christopher Bishop (2007)

Pattern Recognition and Machine Learning

New York, NY: Springer 1st edition.

Vladimir Vapnik (2000)

The Nature of Statistical Learning

New York, NY: Springer 1st edition.

Christopher Burges (1998)

A Tutorial on Support Vector Machines for Pattern Recognition

Data Mining and Knowledge Discovery, pp. 121–167.

Code Availability and Python Notebook

Code used for this chapter is available at
https://github.com/rahulbhadani/CPE490_590_Sp2025/blob/master/Code/Chapter_11_Support_Vector_Machine.ipynb

The End