Lecture 10: Kernel Methods and Support Vector Machines

Tao LIN

SoE, Westlake University

November 18, 2025

WESTLAKE UNIVERSITY | SCHOOL OF ENGINEERING

Table of Contents

A Brief History

- Support Vector Machines are a set of powerful supervised learning methods based on the concept of maximizing the margin.

A Brief History

- Support Vector Machines are a set of powerful supervised learning methods based on the concept of maximizing the margin.
- The idea was developed by **Vladimir Vapnik** and colleagues.

A Brief History

- Support Vector Machines are a set of powerful supervised learning methods based on the concept of maximizing the margin.
- The idea was developed by **Vladimir Vapnik** and colleagues.
- **Key Papers:**
  - **1992 (Linearly Separable):** Boser, Guyon, and Vapnik introduce the optimal margin classifier.
    - Solution is a linear combination of “supporting patterns” (Support Vectors).
  - **1995 (Non-Separable):** Cortes and Vapnik extend the idea to non-separable data (the “Soft Margin”).
    - Introduces non-linear mapping to a high-dimension feature space.

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.
- **Least Squares:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.
- **Least Squares:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$
- **Logistic Regression:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w}$

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.
- **Least Squares:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$
- **Logistic Regression:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w}$
- It's often more convenient to use labels $y_n \in \{-1, 1\}$.

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.
- **Least Squares:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$
- **Logistic Regression:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w}$
- It's often more convenient to use labels $y_n \in \{-1, 1\}$.
- We use the mapping: $y_n = 2\tilde{y}_n - 1 \leftrightarrow \tilde{y}_n = \frac{1}{2}(y_n + 1)$.

Motivation: Reframing Classification Losses

- Let's reconsider binary classification with data $(\mathbf{x}_n, \tilde{y}_n)$, where $\tilde{y}_n \in \{0, 1\}$.
- **Least Squares:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \mathbf{w})^2$
- **Logistic Regression:** $\min_{\mathbf{w}} \frac{1}{N} \sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w}$
- It's often more convenient to use labels $y_n \in \{-1, 1\}$.
- We use the mapping: $y_n = 2\tilde{y}_n - 1 \leftrightarrow \tilde{y}_n = \frac{1}{2}(y_n + 1)$.
- Let's see how our loss functions change with this new y_n.

Motivation: Deriving the MSE Loss

- For Least Squares, let's assume bias e_0 is component 0. Define $\tilde{\mathbf{w}} = \frac{1}{2}(\mathbf{w} + e_0)$ (this is a one-to-one mapping).

Motivation: Deriving the MSE Loss

- For Least Squares, let's assume bias e_0 is component 0. Define $\tilde{\mathbf{w}} = \frac{1}{2}(\mathbf{w} + e_0)$ (this is a one-to-one mapping).
- The derivation becomes:

$$
\begin{aligned}
4\sum_{n=1}^{N}(\tilde{y}_n - \mathbf{x}_n^\top \tilde{\mathbf{w}})^2 &= 4\sum_{n=1}^{N}(\frac{1}{2}(y_n + 1) - \frac{1}{2}\mathbf{x}_n^\top(\mathbf{w} + e_0))^2 \\
&= \sum_{n=1}^{N}((y_n + 1) - \mathbf{x}_n^\top(\mathbf{w} + e_0))^2 = \sum_{n=1}^{N}(y_n - \mathbf{x}_n^\top \mathbf{w})^2 \\
&\overset{(a)}{=} \sum_{n=1}^{N}(1 - y_n \mathbf{x}_n^\top \mathbf{w})^2
\end{aligned}
$$

Motivation: Deriving the MSE Loss

- For Least Squares, let's assume bias e_0 is component 0. Define $\tilde{\mathbf{w}} = \frac{1}{2}(\mathbf{w} + e_0)$ (this is a one-to-one mapping).
- The derivation becomes:

$$
\begin{aligned}
4 \textstyle\sum_{n=1}^{N} (\tilde{y}_n - \mathbf{x}_n^\top \tilde{\mathbf{w}})^2 &= 4 \sum_{n=1}^{N} \left(\frac{1}{2}(y_n + 1) - \frac{1}{2}\mathbf{x}_n^\top (\mathbf{w} + e_0)\right)^2 \\
&= \sum_{n=1}^{N} \left((y_n + 1) - \mathbf{x}_n^\top (\mathbf{w} + e_0)\right)^2 = \sum_{n=1}^{N} (y_n - \mathbf{x}_n^\top \mathbf{w})^2 \\
&\overset{(a)}{=} \sum_{n=1}^{N} (1 - y_n \mathbf{x}_n^\top \mathbf{w})^2
\end{aligned}
$$

- In step (a), we used $y_n^2 = 1$ (since $y_n \in \{\pm 1\}$).

Motivation: Deriving the MSE Loss

- For Least Squares, let's assume bias e_0 is component 0. Define $\tilde{\mathbf{w}} = \frac{1}{2}(\mathbf{w} + e_0)$ (this is a one-to-one mapping).

- The derivation becomes:

$$
\begin{aligned}
4\textstyle\sum_{n=1}^{N}(\tilde{y}_n - \mathbf{x}_n^\top \tilde{\mathbf{w}})^2 &= 4\sum_{n=1}^{N}\left(\frac{1}{2}(y_n + 1) - \frac{1}{2}\mathbf{x}_n^\top(\mathbf{w} + e_0)\right)^2 \\
&= \sum_{n=1}^{N}((y_n + 1) - \mathbf{x}_n^\top(\mathbf{w} + e_0))^2 = \sum_{n=1}^{N}(y_n - \mathbf{x}_n^\top \mathbf{w})^2 \\
&\overset{(a)}{=} \sum_{n=1}^{N}(1 - y_n \mathbf{x}_n^\top \mathbf{w})^2
\end{aligned}
$$

- In step (a), we used $y_n^2 = 1$ (since $y_n \in \{\pm 1\}$).

- This gives the MSE loss in margin form:

$$
\text{MSE}(z, y) = (1 - yz)^2 \quad \text{where } z = \mathbf{x}_n^\top \mathbf{w}
$$

Motivation: Deriving the Logistic Loss

- For Logistic Regression, we can also rewrite the loss:

$$\sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w} = \sum_{n=1}^{N} \log(1 + e^{-y_n \mathbf{x}_n^\top \mathbf{w}})$$

Motivation: Deriving the Logistic Loss

- For Logistic Regression, we can also rewrite the loss:

$$\sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w} = \sum_{n=1}^{N} \log(1 + e^{-y_n \mathbf{x}_n^\top \mathbf{w}})$$

- This is easily seen by checking the two cases:
  - $\tilde{y}_n = 1 \leftrightarrow y_n = 1$:

$$\begin{aligned} \log(1 + e^{\mathbf{w}^\top \mathbf{x}}) - \mathbf{w}^\top \mathbf{x} &= \log(e^{\mathbf{w}^\top \mathbf{x}}(e^{-\mathbf{w}^\top \mathbf{x}} + 1)) - \mathbf{w}^\top \mathbf{x} && (1) \\ &= \log(e^{\mathbf{w}^\top \mathbf{x}}) + \log(1 + e^{-\mathbf{w}^\top \mathbf{x}}) - \mathbf{w}^\top \mathbf{x} && (2) \\ &= \log(1 + e^{-\mathbf{w}^\top \mathbf{x}}) && (3) \end{aligned}$$

  - $\tilde{y}_n = 0 \leftrightarrow y_n = -1$:

$$\log(1 + e^{\mathbf{w}^\top \mathbf{x}}) - 0 = \log(1 + e^{-(-1)\mathbf{w}^\top \mathbf{x}}) \quad (4)$$

Motivation: Deriving the Logistic Loss

- For Logistic Regression, we can also rewrite the loss:

$$\sum_{n=1}^{N} \log(1 + e^{\mathbf{x}_n^\top \mathbf{w}}) - \tilde{y}_n \mathbf{x}_n^\top \mathbf{w} = \sum_{n=1}^{N} \log(1 + e^{-y_n \mathbf{x}_n^\top \mathbf{w}})$$

- This is easily seen by checking the two cases:
  - $\tilde{y}_n = 1 \leftrightarrow y_n = 1$:

$$\log(1 + e^{\mathbf{w}^\top \mathbf{x}}) - \mathbf{w}^\top \mathbf{x} = \log(e^{\mathbf{w}^\top \mathbf{x}}(e^{-\mathbf{w}^\top \mathbf{x}} + 1)) - \mathbf{w}^\top \mathbf{x} \tag{1}$$

$$= \log(e^{\mathbf{w}^\top \mathbf{x}}) + \log(1 + e^{-\mathbf{w}^\top \mathbf{x}}) - \mathbf{w}^\top \mathbf{x} \tag{2}$$

$$= \log(1 + e^{-\mathbf{w}^\top \mathbf{x}}) \tag{3}$$

  - $\tilde{y}_n = 0 \leftrightarrow y_n = -1$:

$$\log(1 + e^{\mathbf{w}^\top \mathbf{x}}) - 0 = \log(1 + e^{-(-1)\mathbf{w}^\top \mathbf{x}}) \tag{4}$$

- This gives the Logistic loss in margin form:

$$\text{LogisticLoss}(z, y) = \log(1 + \exp(-yz))$$

Motivation: The Hinge Loss (SVM)

- Both MSE and Logistic Loss are functions of the functional margin yz.

Motivation: The Hinge Loss (SVM)

- Both MSE and Logistic Loss are functions of the functional margin yz.
- Support Vector Machines (SVMs) are formed by using a third loss, the Hinge Loss:

$$\text{Hinge}(z, y) = [1 - yz]_+ = \max\{0, 1 - yz\}$$

Motivation: The Hinge Loss (SVM)

- Both MSE and Logistic Loss are functions of the functional margin yz.
- Support Vector Machines (SVMs) are formed by using a third loss, the Hinge Loss:

$$\text{Hinge}(z, y) = [1 - yz]_+ = \max\{0, 1 - yz\}$$

- ...and adding a regularization term.

Motivation: The Hinge Loss (SVM)

- Both MSE and Logistic Loss are functions of the functional margin yz.
- Support Vector Machines (SVMs) are formed by using a third loss, the Hinge Loss:

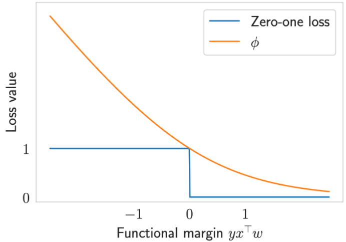
$$\text{Hinge}(z, y) = [1 - yz]_+ = \max\{0, 1 - yz\}$$

- ...and adding a regularization term.
- This leads directly to the SVM optimization problem we will study.

The 0-1 Loss and Convex Relaxations

- The "ideal" loss for classification is the Zero-one loss:

$$L_{0-1}(\eta) = \mathbf{1}_{\eta<0} \quad \text{where } \eta = y\mathbf{x}^\top \mathbf{w}$$

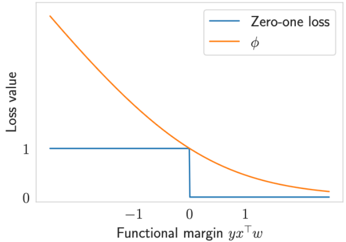


The 0-1 Loss and Convex Relaxations

- The "ideal" loss for classification is the Zero-one loss:

$$L_{0-1}(\eta) = \mathbf{1}_{\eta<0} \quad \text{where } \eta = y\mathbf{x}^\top \mathbf{w}$$

- This corresponds to Empirical Risk Minimization (ERM).



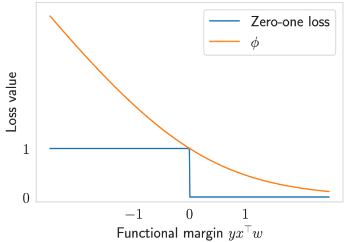
The 0-1 Loss and Convex Relaxations

- The "ideal" loss for classification is the Zero-one loss:

$$L_{0-1}(\eta) = \mathbf{1}_{\eta<0} \quad \text{where } \eta = y\mathbf{x}^\top \mathbf{w}$$

- This corresponds to Empirical Risk Minimization (ERM).

- **Problem:** The 0-1 loss is **not convex** and **not continuous**. It is computationally hard to optimize.



The 0-1 Loss and Convex Relaxations

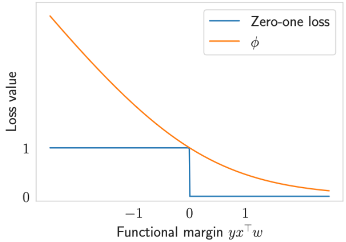
- The “ideal” loss for classification is the Zero-one loss:

$$L_{0-1}(\eta) = \mathbf{1}_{\eta<0} \quad \text{where } \eta = y\mathbf{x}^\top \mathbf{w}$$

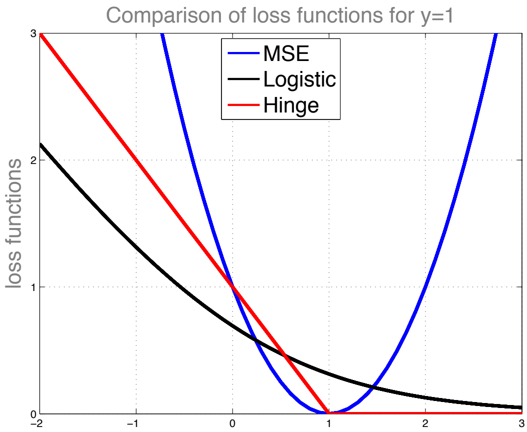
- This corresponds to Empirical Risk Minimization (ERM).

- **Problem:** The 0-1 loss is **not convex** and **not continuous**. It is computationally hard to optimize.

- **Solution:** We replace the 0-1 loss with a convex surrogate (an upper bound) that is easier to minimize.

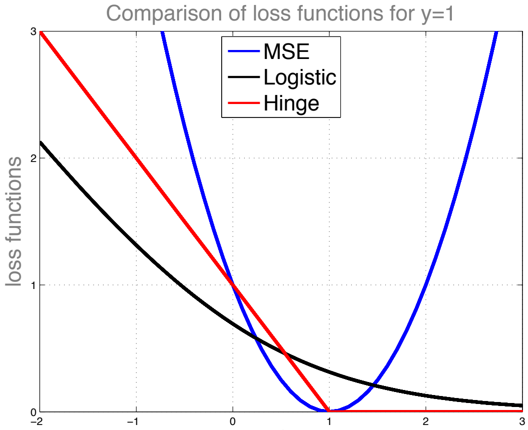


Comparison of Loss Functions



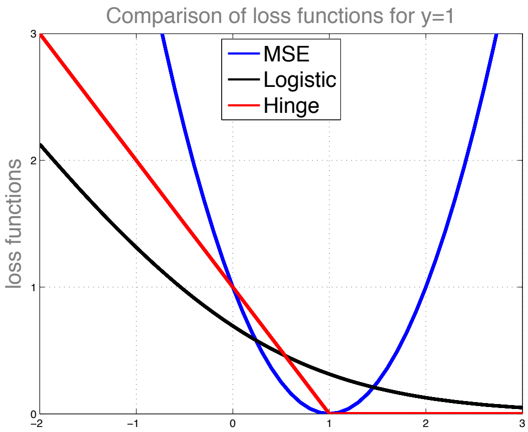
- **MSE:** Penalizes all points, even correctly classified ones. Symmetric.

Comparison of Loss Functions



- **MSE:** Penalizes all points, even correctly classified ones. Symmetric.
- **Logistic:** Always incurs a cost, but it's asymmetric.

Comparison of Loss Functions



- **MSE:** Penalizes all points, even correctly classified ones. Symmetric.
- **Logistic:** Always incurs a cost, but it's asymmetric.
- **Hinge:** No cost if a point is "sufficiently far" on the correct side (i.e., $y_n \mathbf{x}_n^\top \mathbf{w} \geq 1$). Creates a margin.

Support Vector Machine Optimization Problem

SVMs solve the following optimization problem:

$$\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+}_{\text{Hinge Loss (Empirical Risk)}} + \underbrace{\frac{\lambda}{2}\left\|\mathbf{w}\right\|^2}_{\text{Regularization}}$$

Support Vector Machine Optimization Problem

SVMs solve the following optimization problem:

$$\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+}_{\text{Hinge Loss (Empirical Risk)}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{Regularization}}$$

- The **Hinge Loss** pushes for correct classification, penalizing points in the margin ($yz < 1$).
- The **Regularization** ($\|w\|^2$) pushes for a simpler model.

Support Vector Machine Optimization Problem

SVMs solve the following optimization problem:

$$\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+}_{\text{Hinge Loss (Empirical Risk)}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{Regularization}}$$

- The **Hinge Loss** pushes for correct classification, penalizing points in the margin ($yz < 1$).
- The **Regularization** ($\|\mathbf{w}\|^2$) pushes for a simpler model.
- **Geometric Meaning:** Minimizing $\|\mathbf{w}\|^2$ is equivalent to **maximizing the margin**.

Support Vector Machine Optimization Problem

SVMs solve the following optimization problem:

$$\min_{\mathbf{w}} \underbrace{\frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+}_{\text{Hinge Loss (Empirical Risk)}} + \underbrace{\frac{\lambda}{2}\|\mathbf{w}\|^2}_{\text{Regularization}}$$

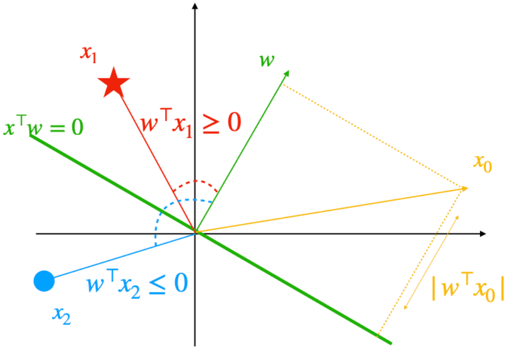
- The **Hinge Loss** pushes for correct classification, penalizing points in the margin ($yz < 1$).
- The **Regularization** ($\|\mathbf{w}\|^2$) pushes for a simpler model.
- **Geometric Meaning:** Minimizing $\|\mathbf{w}\|^2$ is equivalent to **maximizing the margin**.
- λ is a hyperparameter that controls the trade-off:
  - **Small λ:** “Harder” margin. Focus on maximizing margin. Low bias, high variance.
  - **Large λ:** “Softer” margin. Focus on minimizing classification errors. High bias, low variance.

Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.

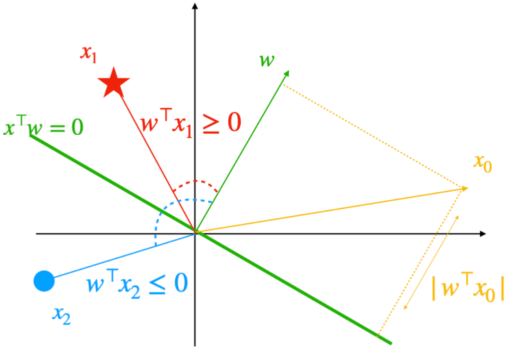
Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.
- The prediction is $g(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$.



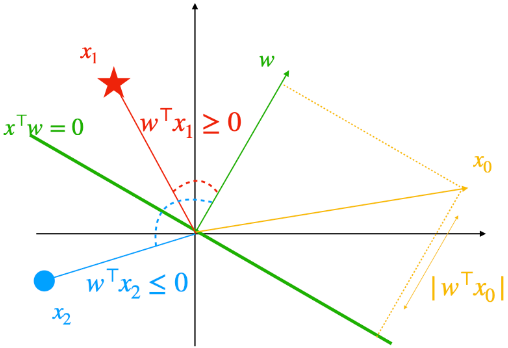
Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.
- The prediction is $g(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$.
- The geometric margin for a point $\mathbf{x}_0$ is its Euclidean distance to the hyperplane H.



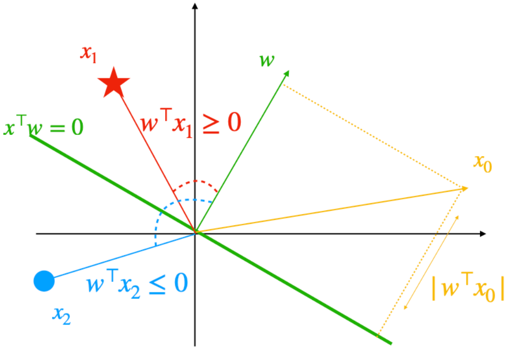
Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.
- The prediction is $g(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$.
- The geometric margin for a point $\mathbf{x}_0$ is its Euclidean distance to the hyperplane H.
- **Claim:** The distance from $\mathbf{x}_0$ to H is $\frac{|\mathbf{x}_0^\top \mathbf{w}|}{\|\mathbf{w}\|}$.
  - (This is found by solving $\min_{\mathbf{x}} \|\mathbf{x}_0 - \mathbf{x}\|$ s.t. $\mathbf{x}^\top \mathbf{w} = 0$).



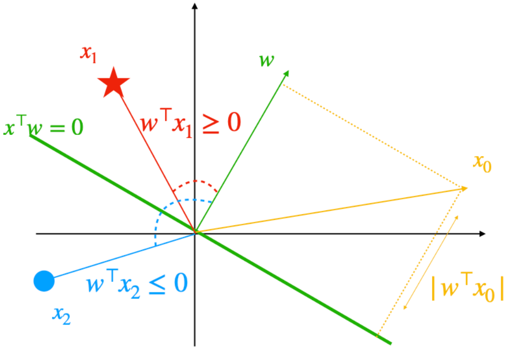
Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.
- The prediction is $g(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$.
- The geometric margin for a point $\mathbf{x}_0$ is its Euclidean distance to the hyperplane H.
- **Claim:** The distance from $\mathbf{x}_0$ to H is $\frac{|\mathbf{x}_0^\top \mathbf{w}|}{\|\mathbf{w}\|}$.
  - (This is found by solving $\min_{\mathbf{x}} \|\mathbf{x}_0 - \mathbf{x}\|$ s.t. $\mathbf{x}^\top \mathbf{w} = 0$).
- The functional margin for a point $(\mathbf{x}_n, y_n)$ is $y_n(\mathbf{x}_n^\top \mathbf{w})$.
  - $g(\mathbf{x}_n) = y_n \iff$ functional margin is positive.

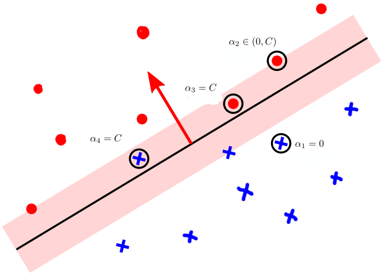


Geometric Preliminaries: Margins

- A linear classifier is defined by a hyperplane: $H = \{\mathbf{x} : \mathbf{x}^\top \mathbf{w} = 0\}$.
- The prediction is $g(\mathbf{x}) = \text{sign}(\mathbf{x}^\top \mathbf{w})$.
- The geometric margin for a point $\mathbf{x}_0$ is its Euclidean distance to the hyperplane H.
- **Claim:** The distance from $\mathbf{x}_0$ to H is $\frac{|\mathbf{x}_0^\top \mathbf{w}|}{\|\mathbf{w}\|}$.
    - (This is found by solving $\min_{\mathbf{x}} \|\mathbf{x}_0 - \mathbf{x}\|$ s.t. $\mathbf{x}^\top \mathbf{w} = 0$).
- The functional margin for a point $(\mathbf{x}_n, y_n)$ is $y_n(\mathbf{x}_n^\top \mathbf{w})$.
    - $g(\mathbf{x}_n) = y_n \iff$ functional margin is positive.
- We can rescale $\mathbf{w}$ without changing the hyperplane, but it changes the functional margin. The geometric margin is invariant.

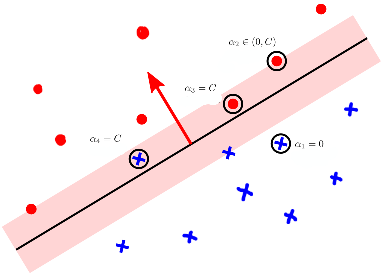


Geometric Interpretation: The Margin



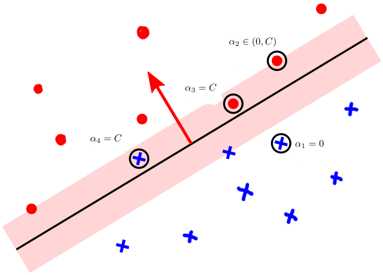
- The separating hyperplane is $\mathbf{x}^\top \mathbf{w} = 0$.

Geometric Interpretation: The Margin



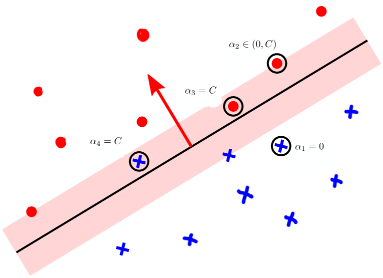
- The separating hyperplane is $\mathbf{x}^\top \mathbf{w} = 0$.
- The "margin" is the "street" defined by the two hyperplanes $\mathbf{x}^\top \mathbf{w} = 1$ and $\mathbf{x}^\top \mathbf{w} = -1$.

Geometric Interpretation: The Margin



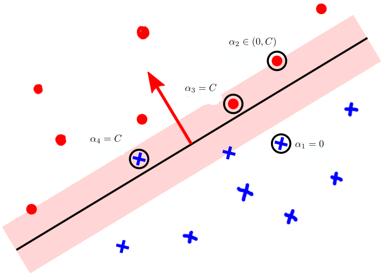
- The separating hyperplane is $\mathbf{x}^\top \mathbf{w} = 0$.
- The "margin" is the "street" defined by the two hyperplanes $\mathbf{x}^\top \mathbf{w} = 1$ and $\mathbf{x}^\top \mathbf{w} = -1$.
- The Hinge Loss penalizes any point n that is on the wrong side of its margin boundary (i.e., $y_n \mathbf{x}_n^\top \mathbf{w} < 1$).

Geometric Interpretation: The Margin



- The separating hyperplane is $\mathbf{x}^\top \mathbf{w} = 0$.
- The "margin" is the "street" defined by the two hyperplanes $\mathbf{x}^\top \mathbf{w} = 1$ and $\mathbf{x}^\top \mathbf{w} = -1$.
- The Hinge Loss penalizes any point n that is on the wrong side of its margin boundary (i.e., $y_n \mathbf{x}_n^\top \mathbf{w} < 1$).
- The width of this margin is $\frac{2}{\|\mathbf{w}\|}$. The margin does not only depend on the direction of the vector $\mathbf{w}$ but also its norm.

Geometric Interpretation: The Margin



- The separating hyperplane is $\mathbf{x}^\top \mathbf{w} = 0$.
- The “margin” is the “street” defined by the two hyperplanes $\mathbf{x}^\top \mathbf{w} = 1$ and $\mathbf{x}^\top \mathbf{w} = -1$.
- The Hinge Loss penalizes any point n that is on the wrong side of its margin boundary (i.e., $y_n \mathbf{x}_n^\top \mathbf{w} < 1$).
- The width of this margin is $\frac{2}{\|\mathbf{w}\|}$. The margin does not only depend on the direction of the vector $\mathbf{w}$ but also its norm.
- **Maximizing the margin** $\left(\frac{2}{\|\mathbf{w}\|}\right)$ is the same as **minimizing** $\|\mathbf{w}\|^2$.

The Hard-Margin Formulation

- Let's first assume the data is **linearly separable**.

The Hard-Margin Formulation

- Let's first assume the data is **linearly separable**.
- The goal is to find the max-margin separating hyperplane.

(I) Maximize Geometric Margin Maximize the geometric margin $\frac{M}{\|\mathbf{w}\|}$, while ensuring all points are correctly classified: $\max_{\mathbf{w}, \|\mathbf{w}\|=1} \min_n \left|\mathbf{x}_n^\top \mathbf{w}\right|$ s.t. $y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 0, \forall n$.

The Hard-Margin Formulation

- Let's first assume the data is **linearly separable**.
- The goal is to find the max-margin separating hyperplane.
- This problem can be formulated in (at least) three equivalent ways:

(I) Maximize Geometric Margin Maximize the geometric margin $\frac{M}{\|\mathbf{w}\|}$, while ensuring all points are correctly classified: $\max_{\mathbf{w},\|\mathbf{w}\|=1} \min_n \left|\mathbf{x}_n^\top \mathbf{w}\right|$ s.t. $y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 0, \forall n$.

(II) Maximize Functional Margin Fix $\|\mathbf{w}\| = 1$ and maximize the functional margin M:

$$\max_{M \in \mathbb{R}, \mathbf{w}, \|\mathbf{w}\|=1} M \quad \text{s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq M, \forall n$$

The Hard-Margin Formulation

- Let's first assume the data is **linearly separable**.
- The goal is to find the max-margin separating hyperplane.
- This problem can be formulated in (at least) three equivalent ways:

(I) Maximize Geometric Margin Maximize the geometric margin $\frac{M}{\|\mathbf{w}\|}$, while ensuring all points are correctly classified: $\max_{\mathbf{w}, \|\mathbf{w}\|=1} \min_n \left|\mathbf{x}_n^\top \mathbf{w}\right|$ s.t. $y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 0, \forall n$.

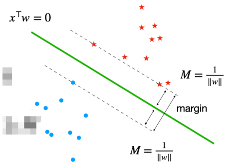
(II) Maximize Functional Margin Fix $\|\mathbf{w}\| = 1$ and maximize the functional margin M:

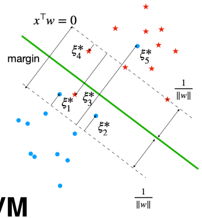
$$\max_{M \in \mathbb{R}, \mathbf{w}, \|\mathbf{w}\|=1} M \quad \text{s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq M, \forall n$$

(III) Minimize $\|\mathbf{w}\|^2$ (Standard Form) Rescale $\mathbf{w}$ so the functional margin for the closest points is 1. Then minimize $\|\mathbf{w}\|^2$ (which maximizes geometric margin $1/\|\mathbf{w}\|$).

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \quad \text{s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 1, \forall n$$

From Hard-Margin to Soft-Margin SVM





Hard-Margin SVM

- Assumes data is **linearly separable**.

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \text{ s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 1, \forall n$$

- Fails if data is not separable.

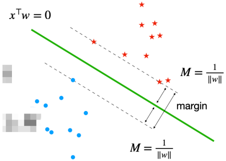
Soft-Margin SVM

- Allows for violations of the margin.
- Introduce positive “slack” variables $\xi_n \geq 0$.

$$\min_{\mathbf{w},\xi} \frac{\lambda}{2}\|\mathbf{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N}\xi_n \text{ s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 1 - \xi_n, \forall n$$

$$\xi_n \geq 0, \forall n$$

From Hard-Margin to Soft-Margin SVM

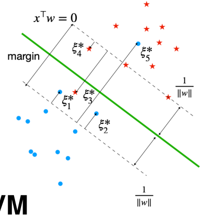


Hard-Margin SVM

- Assumes data is **linearly separable**.

$$\min_{\mathbf{w}} \frac{1}{2} \|\mathbf{w}\|^2 \text{ s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 1, \forall n$$

- Fails if data is not separable.



Soft-Margin SVM

- Allows for violations of the margin.
- Introduce positive "slack" variables $\xi_n \geq 0$.

$$\min_{\mathbf{w},\xi} \frac{\lambda}{2} \|\mathbf{w}\|^2 + \frac{1}{N}\sum_{n=1}^{N} \xi_n \text{ s.t. } y_n(\mathbf{x}_n^\top \mathbf{w}) \geq 1 - \xi_n, \forall n$$

$$\xi_n \geq 0, \forall n$$

The two formulations are equivalent. The optimal ξ_n is exactly the Hinge Loss:

$$\xi_n^* = \max(0, 1 - y_n \mathbf{x}_n^\top \mathbf{w}) = [1 - y_n \mathbf{x}_n^\top \mathbf{w}]_+$$

Optimization (Primal)

$$\min_{\mathbf{w}} \; \frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+ + \tfrac{\lambda}{2}\left\|\mathbf{w}\right\|^2$$

- Is this function convex? (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).

Optimization (Primal)

$$\min_{\mathbf{w}} \ \frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+ + \frac{\lambda}{2}\left\|\mathbf{w}\right\|^2$$

- Is this function convex? **Yes.** (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).

Optimization (Primal)

$$\min_{\mathbf{w}} \; \frac{1}{N} \sum_{n=1}^{N} \left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- Is this function convex? **Yes.** (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).
- Is it differentiable? (The Hinge loss has a "kink" at $1 - y_n \mathbf{x}_n^\top \mathbf{w} = 0$).

Optimization (Primal)

$$\min_{\mathbf{w}} \; \frac{1}{N} \sum_{n=1}^{N} \left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+ + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

- Is this function convex? **Yes.** (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).
- Is it differentiable? **No.** (The Hinge loss has a “kink” at $1 - y_n \mathbf{x}_n^\top \mathbf{w} = 0$).

Optimization (Primal)

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{n=1}^{N}\left[1 - y_n \mathbf{x}_n^\top \mathbf{w}\right]_+ + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- Is this function convex? **Yes.** (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).
- Is it differentiable? **No.** (The Hinge loss has a "kink" at $1 - y_n \mathbf{x}_n^\top \mathbf{w} = 0$).
- Because it is convex, we can optimize it using subgradient methods, like Stochastic Subgradient Descent (SGD).

Optimization (Primal)

$$\min_{\mathbf{w}} \frac{1}{N}\sum_{n=1}^{N}\left[1-y_n\mathbf{x}_n^\top\mathbf{w}\right]_+ + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

- Is this function convex? **Yes.** (Sum of convex functions: Hinge loss is convex, $\|\mathbf{w}\|^2$ is convex).
- Is it differentiable? **No.** (The Hinge loss has a “kink” at $1-y_n\mathbf{x}_n^\top\mathbf{w}=0$).
- Because it is convex, we can optimize it using subgradient methods, like Stochastic Subgradient Descent (SGD).
- But this can be slow. An alternative, and very important, approach is to use duality.

Duality: The Big Picture

- We rewrite our minimization problem (Primal) $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ into equivalent "saddle-point" problem.

Duality: The Big Picture

- We rewrite our minimization problem (Primal) $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ into equivalent "saddle-point" problem.
- We introduce an auxiliary function $G(\mathbf{w}, \boldsymbol{\alpha})$ and dual variables $\boldsymbol{\alpha}$:

$$\mathcal{L}(\mathbf{w}) = \max_{\boldsymbol{\alpha}} \; G(\mathbf{w}, \boldsymbol{\alpha})$$

Duality: The Big Picture

- We rewrite our minimization problem (Primal) $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ into equivalent “saddle-point” problem.
- We introduce an auxiliary function $G(\mathbf{w}, \boldsymbol{\alpha})$ and dual variables $\boldsymbol{\alpha}$:

$$\mathcal{L}(\mathbf{w}) = \max_{\boldsymbol{\alpha}} \; G(\mathbf{w}, \boldsymbol{\alpha})$$

- **Primal Problem:**

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}} G(\mathbf{w}, \boldsymbol{\alpha})$$

Duality: The Big Picture

- We rewrite our minimization problem (Primal) $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ into equivalent "saddle-point" problem.
- We introduce an auxiliary function $G(\mathbf{w}, \boldsymbol{\alpha})$ and dual variables $\boldsymbol{\alpha}$:

$$\mathcal{L}(\mathbf{w}) = \max_{\boldsymbol{\alpha}} \; G(\mathbf{w}, \boldsymbol{\alpha})$$

- **Primal Problem:**

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}} G(\mathbf{w}, \boldsymbol{\alpha})$$

- **Dual Problem:** (We swap the min and max)

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} G(\mathbf{w}, \boldsymbol{\alpha})$$

Duality: The Big Picture

- We rewrite our minimization problem (Primal) $\min_{\mathbf{w}} \mathcal{L}(\mathbf{w})$ into equivalent "saddle-point" problem.
- We introduce an auxiliary function $G(\mathbf{w}, \boldsymbol{\alpha})$ and dual variables $\boldsymbol{\alpha}$:

$$\mathcal{L}(\mathbf{w}) = \max_{\boldsymbol{\alpha}} \; G(\mathbf{w}, \boldsymbol{\alpha})$$

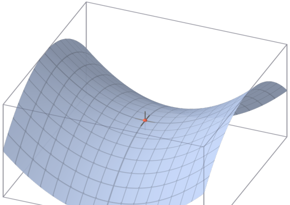
- **Primal Problem:**

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha}} G(\mathbf{w}, \boldsymbol{\alpha})$$

- **Dual Problem:** (We swap the min and max)

$$\max_{\boldsymbol{\alpha}} \min_{\boldsymbol{w}} G(\mathbf{w}, \boldsymbol{\alpha})$$

- For SVM, the domains are convex and G is convex in $\mathbf{w}$ and concave in $\boldsymbol{\alpha}$, so strong duality holds:



Duality: Finding $G(\mathbf{w}, \boldsymbol{\alpha})$ for SVM

- We use a key trick for the Hinge loss:

$$[z]_+ = \max_{\alpha \in [0,1]} \alpha z$$

Duality: Finding $G(\mathbf{w}, \boldsymbol{\alpha})$ for SVM

- We use a key trick for the Hinge loss:

$$[z]_+ = \max_{\alpha \in [0,1]} \alpha z$$

- Applying this to our Hinge loss for each data point:

$$[1 - y_n \mathbf{x}_n^\top \mathbf{w}]_+ = \max_{\alpha_n \in [0,1]} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w})$$

Duality: Finding $G(\mathbf{w}, \boldsymbol{\alpha})$ for SVM

- We use a key trick for the Hinge loss:

$$[z]_+ = \max_{\alpha \in [0,1]} \alpha z$$

- Applying this to our Hinge loss for each data point:

$$[1 - y_n \mathbf{x}_n^\top \mathbf{w}]_+ = \max_{\alpha_n \in [0,1]} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w})$$

- We can now rewrite the *entire* SVM problem:

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha} \in [0,1]^N} \underbrace{\frac{1}{N} \sum_{n=1}^{N} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2}_{G(\boldsymbol{w}, \boldsymbol{\alpha})}$$

Duality: Finding $G(\mathbf{w}, \boldsymbol{\alpha})$ for SVM

- We use a key trick for the Hinge loss:

$$[z]_+ = \max_{\alpha \in [0,1]} \alpha z$$

- Applying this to our Hinge loss for each data point:

$$[1 - y_n \mathbf{x}_n^\top \mathbf{w}]_+ = \max_{\alpha_n \in [0,1]} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w})$$

- We can now rewrite the *entire* SVM problem:

$$\min_{\boldsymbol{w}} \max_{\boldsymbol{\alpha} \in [0,1]^N} \underbrace{\frac{1}{N} \sum_{n=1}^{N} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w}) + \frac{\lambda}{2} \|\mathbf{w}\|^2}_{G(\boldsymbol{w}, \boldsymbol{\alpha})}$$

- This $G(\mathbf{w}, \boldsymbol{\alpha})$ is convex in $\mathbf{w}$ and concave (linear) in $\boldsymbol{\alpha}$.

Duality: Solving the Dual (Step 1)

We want to solve: $\max_{\boldsymbol{\alpha} \in [0,1]^N} \min_{\boldsymbol{w}} \; G(\mathbf{w}, \boldsymbol{\alpha})$

- **Step 1: Minimize w.r.t.** $\mathbf{w}$ (for a fixed $\boldsymbol{\alpha}$)

Duality: Solving the Dual (Step 1)

We want to solve: $\max_{\boldsymbol{\alpha}\in[0,1]^N} \min_{\boldsymbol{w}} \; G(\mathbf{w}, \boldsymbol{\alpha})$

- **Step 1: Minimize w.r.t. $\mathbf{w}$** (for a fixed $\boldsymbol{\alpha}$)
- We find the $\mathbf{w}$ that minimizes G by taking the gradient and setting it to zero:

$$\nabla_{\boldsymbol{w}} G(\mathbf{w}, \boldsymbol{\alpha}) = \nabla_{\boldsymbol{w}} \left(\frac{1}{N} \sum_{n=1}^{N} \alpha_n - \frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^\top \mathbf{w} + \tfrac{\lambda}{2} \|\mathbf{w}\|^2 \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n + \lambda \mathbf{w} \overset{\text{set}}{=} 0$$

Duality: Solving the Dual (Step 1)

We want to solve: $\max_{\boldsymbol{\alpha} \in [0,1]^N} \min_{\boldsymbol{w}} \; G(\mathbf{w}, \boldsymbol{\alpha})$

- **Step 1: Minimize w.r.t. $\mathbf{w}$** (for a fixed $\boldsymbol{\alpha}$)
- We find the $\mathbf{w}$ that minimizes G by taking the gradient and setting it to zero:

$$\nabla_{\boldsymbol{w}} G(\mathbf{w}, \boldsymbol{\alpha}) = \nabla_{\boldsymbol{w}} \left(\frac{1}{N} \sum_{n=1}^{N} \alpha_n - \frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^\top \mathbf{w} + \frac{\lambda}{2} \|\mathbf{w}\|^2 \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n + \lambda \mathbf{w} \overset{\text{set}}{=} 0$$

- This gives us the optimal $\mathbf{w}$ *as a function of* $\boldsymbol{\alpha}$:

$$\mathbf{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = \frac{1}{\lambda N} \mathbf{X}^\top \mathbf{Y} \boldsymbol{\alpha}$$

Duality: Solving the Dual (Step 1)

We want to solve: $\max_{\boldsymbol{\alpha}\in[0,1]^N} \min_{\boldsymbol{w}} \; G(\mathbf{w}, \boldsymbol{\alpha})$

- **Step 1: Minimize w.r.t. $\mathbf{w}$** (for a fixed $\boldsymbol{\alpha}$)
- We find the $\mathbf{w}$ that minimizes G by taking the gradient and setting it to zero:

$$\nabla_{\boldsymbol{w}} G(\mathbf{w}, \boldsymbol{\alpha}) = \nabla_{\boldsymbol{w}} \left(\frac{1}{N} \sum_{n=1}^{N} \alpha_n - \frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n^\top \mathbf{w} + \tfrac{\lambda}{2} \|\mathbf{w}\|^2 \right)$$

$$= -\frac{1}{N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n + \lambda \mathbf{w} \overset{\text{set}}{=} 0$$

- This gives us the optimal $\mathbf{w}$ *as a function of* $\boldsymbol{\alpha}$:

$$\mathbf{w}(\boldsymbol{\alpha}) = \tfrac{1}{\lambda N} \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n = \tfrac{1}{\lambda N} \mathbf{X}^\top \mathbf{Y} \boldsymbol{\alpha}$$

- **Crucial Insight:** The optimal weight vector $\mathbf{w}$ is a linear combination of the data points $\mathbf{x}_n$.

Duality: Solving the Dual (Step 2)

- **Step 2: Plug $\mathbf{w}(\alpha)$ back into $G(\mathbf{w}, \alpha)$.**

Duality: Solving the Dual (Step 2)

- **Step 2: Plug $\mathbf{w}(\boldsymbol{\alpha})$ back into $G(\mathbf{w}, \boldsymbol{\alpha})$.**
- We substitute $\mathbf{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda N} \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j$ into G:

$$G(\mathbf{w}(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = \frac{1}{N} \sum_{n=1}^{N} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w}(\boldsymbol{\alpha})) + \tfrac{\lambda}{2} \|\mathbf{w}(\boldsymbol{\alpha})\|^2$$

$$= \tfrac{1}{N} \sum_n \alpha_n - \tfrac{1}{N} \sum_n \alpha_n y_n \mathbf{x}_n^\top \left(\tfrac{1}{\lambda N} \sum_j \alpha_j y_j \mathbf{x}_j\right) + \tfrac{\lambda}{2} \left\| \tfrac{1}{\lambda N} \sum_n \alpha_n y_n \mathbf{x}_n \right\|^2$$

Duality: Solving the Dual (Step 2)

- **Step 2: Plug $\mathbf{w}(\boldsymbol{\alpha})$ back into $G(\mathbf{w}, \boldsymbol{\alpha})$.**
- We substitute $\mathbf{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda N} \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j$ into G:

$$G(\mathbf{w}(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = \frac{1}{N} \sum_{n=1}^{N} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w}(\boldsymbol{\alpha})) + \frac{\lambda}{2} \|\mathbf{w}(\boldsymbol{\alpha})\|^2$$

$$= \frac{1}{N} \sum_n \alpha_n - \frac{1}{N} \sum_n \alpha_n y_n \mathbf{x}_n^\top \left(\frac{1}{\lambda N} \sum_j \alpha_j y_j \mathbf{x}_j\right) + \frac{\lambda}{2} \left\| \frac{1}{\lambda N} \sum_n \alpha_n y_n \mathbf{x}_n \right\|^2$$

- Let's simplify the terms. Note $\sum_n \sum_j \alpha_n \alpha_j y_n y_j (\mathbf{x}_n^\top \mathbf{x}_j) = \boldsymbol{\alpha}^\top \mathbf{YKY}\boldsymbol{\alpha}$ where $\mathbf{K} = \mathbf{XX}^\top$.

$$= \frac{1}{N} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{\lambda N^2} (\boldsymbol{\alpha}^\top \mathbf{YXX}^\top \mathbf{Y}\boldsymbol{\alpha}) + \frac{\lambda}{2\lambda^2 N^2} (\boldsymbol{\alpha}^\top \mathbf{YXX}^\top \mathbf{Y}\boldsymbol{\alpha})$$
$$= \frac{1}{N} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda N^2} \boldsymbol{\alpha}^\top \mathbf{YKY}\boldsymbol{\alpha}$$

Duality: Solving the Dual (Step 2)

- **Step 2: Plug $\mathbf{w}(\boldsymbol{\alpha})$ back into $G(\mathbf{w}, \boldsymbol{\alpha})$.**
- We substitute $\mathbf{w}(\boldsymbol{\alpha}) = \frac{1}{\lambda N} \sum_{j=1}^{N} \alpha_j y_j \mathbf{x}_j$ into G:

$$G(\mathbf{w}(\boldsymbol{\alpha}), \boldsymbol{\alpha}) = \frac{1}{N} \sum_{n=1}^{N} \alpha_n (1 - y_n \mathbf{x}_n^\top \mathbf{w}(\boldsymbol{\alpha})) + \tfrac{\lambda}{2} \|\mathbf{w}(\boldsymbol{\alpha})\|^2$$

$$= \tfrac{1}{N} \sum_n \alpha_n - \tfrac{1}{N} \sum_n \alpha_n y_n \mathbf{x}_n^\top (\tfrac{1}{\lambda N} \sum_j \alpha_j y_j \mathbf{x}_j) + \tfrac{\lambda}{2} \left\| \tfrac{1}{\lambda N} \sum_n \alpha_n y_n \mathbf{x}_n \right\|^2$$

- Let's simplify the terms. Note $\sum_n \sum_j \alpha_n \alpha_j y_n y_j (\mathbf{x}_n^\top \mathbf{x}_j) = \boldsymbol{\alpha}^\top \mathbf{YKY} \boldsymbol{\alpha}$ where $\mathbf{K} = \mathbf{XX}^\top$.

$$= \tfrac{1}{N} \boldsymbol{\alpha}^\top \mathbf{1} - \tfrac{1}{\lambda N^2} (\boldsymbol{\alpha}^\top \mathbf{YXX}^\top \mathbf{Y} \boldsymbol{\alpha}) + \tfrac{\lambda}{2\lambda^2 N^2} (\boldsymbol{\alpha}^\top \mathbf{YXX}^\top \mathbf{Y} \boldsymbol{\alpha})$$
$$= \tfrac{1}{N} \boldsymbol{\alpha}^\top \mathbf{1} - \tfrac{1}{2\lambda N^2} \boldsymbol{\alpha}^\top \mathbf{YKY} \boldsymbol{\alpha}$$

- This gives the final Dual Problem:

$$\max_{\boldsymbol{\alpha} \in [0,1]^N} \ \tfrac{1}{N} \boldsymbol{\alpha}^\top \mathbf{1} - \tfrac{1}{2\lambda N^2} \boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha}$$

where $\mathbf{Q} = \mathbf{YKY} = \mathrm{diag}(\mathbf{y}) \mathbf{XX}^\top \mathrm{diag}(\mathbf{y})$

Duality: Why Bother?

The dual problem is often better:

$$\max_{\boldsymbol{\alpha} \in [0,1]^N} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda N^2} \boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} \quad \text{where } \mathbf{Q} = \mathbf{Y}(\mathbf{X}\mathbf{X}^\top)\mathbf{Y}$$

- **1. It's a constrained quadratic problem.** We can solve it efficiently with Coordinate Ascent.

Duality: Why Bother?

The dual problem is often better:

$$\max_{\boldsymbol{\alpha}\in[0,1]^N} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda N^2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha} \quad \text{where } \mathbf{Q} = \mathbf{Y}(\mathbf{X}\mathbf{X}^\top)\mathbf{Y}$$

- **1. It's a constrained quadratic problem.** We can solve it efficiently with Coordinate Ascent.
- **2. The data only appears as $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.**
  - This $N \times N$ matrix is called the Kernel.
  - The problem complexity now depends on N (samples), not D (features).
  - **This is a huge win when $D \gg N$ (e.g., text, genomics).**
  - This property is what enables the Kernel Trick.

Duality: Why Bother?

The dual problem is often better:

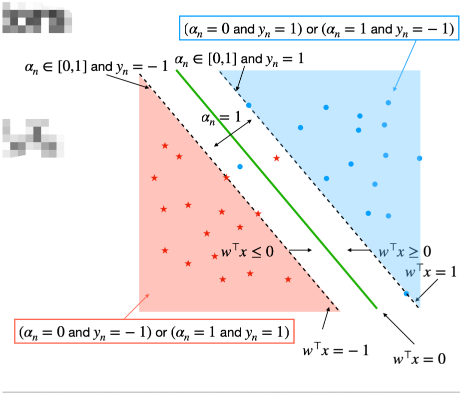
$$\max_{\boldsymbol{\alpha} \in [0,1]^N} \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda N^2} \boldsymbol{\alpha}^\top \mathbf{Q} \boldsymbol{\alpha} \quad \text{where } \mathbf{Q} = \mathbf{Y}(\mathbf{X}\mathbf{X}^\top)\mathbf{Y}$$

- **1. It's a constrained quadratic problem.** We can solve it efficiently with Coordinate Ascent.
- **2. The data only appears as $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.**
  - This $N \times N$ matrix is called the Kernel.
  - The problem complexity now depends on N (samples), not D (features).
  - **This is a huge win when $D \gg N$ (e.g., text, genomics).**
  - This property is what enables the Kernel Trick.
- **3. The solution α is sparse.**
  - Most α_n will be 0.
  - The $\mathbf{w}$ vector ($\mathbf{w} = \frac{1}{\lambda N} \sum \alpha_n y_n \mathbf{x}_n$) only depends on the points $\mathbf{x}_n$ where $\alpha_n > 0$.
  - These points are the Support Vectors.

Interpretation: Support Vectors

From the KKT conditions (the optimality conditions for the dual):

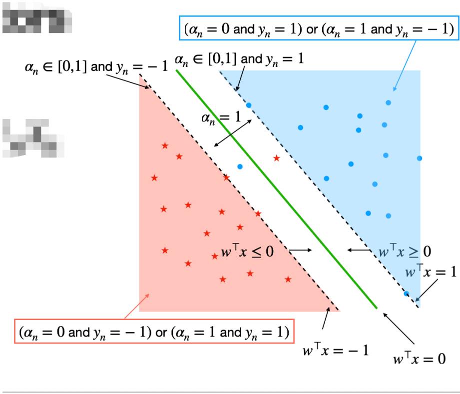
- **Case 1: Point is outside margin.** $(\alpha_n = 0)$
  - $y_n \mathbf{x}_n^\top \mathbf{w} > 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} < 0$).
  - The point is correctly classified with room to spare.
  - The Hinge Loss is 0.
  - This point has **no influence** on the decision boundary.



Interpretation: Support Vectors

From the KKT conditions (the optimality conditions for the dual):

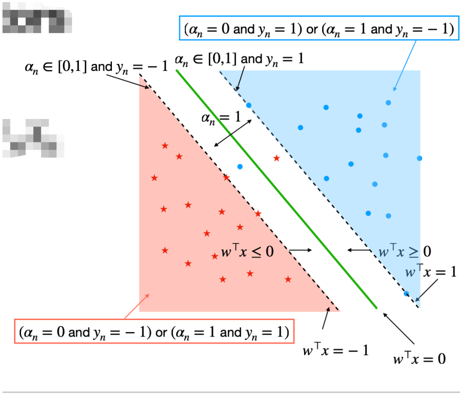
- **Case 1: Point is outside margin.** $(\alpha_n = 0)$
  - $y_n \mathbf{x}_n^\top \mathbf{w} > 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} < 0$).
  - The point is correctly classified with room to spare.
  - The Hinge Loss is 0.
  - This point has **no influence** on the decision boundary.
- **Case 2: Point is on the margin.** $(\alpha_n \in (0, 1])$
  - $y_n \mathbf{x}_n^\top \mathbf{w} = 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} = 0$).
  - This is an Essential Support Vector. It lies exactly on the margin and holds the "street" in place.



Interpretation: Support Vectors

From the KKT conditions (the optimality conditions for the dual):

- **Case 1: Point is outside margin.** ($\alpha_n = 0$)
  - $y_n \mathbf{x}_n^\top \mathbf{w} > 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} < 0$).
  - The point is correctly classified with room to spare.
  - The Hinge Loss is 0.
  - This point has **no influence** on the decision boundary.
- **Case 2: Point is on the margin.** ($\alpha_n \in (0, 1]$)
  - $y_n \mathbf{x}_n^\top \mathbf{w} = 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} = 0$).
  - This is an Essential Support Vector. It lies exactly on the margin and holds the "street" in place.
- **Case 3: Point is inside margin or misclassified.** ($\alpha_n = 1$)
  - $y_n \mathbf{x}_n^\top \mathbf{w} < 1$ (or $1 - y_n \mathbf{x}_n^\top \mathbf{w} > 0$).
  - This is a Bound Support Vector. It is either inside the margin or on the wrong side of the hyperplane.



A Deeper Look: KKT Conditions

- KKT (Karush-Kuhn-Tucker) conditions are the formal rules that an optimal solution to a constrained problem (like SVM) must satisfy.

A Deeper Look: KKT Conditions

- KKT (Karush-Kuhn-Tucker) conditions are the formal rules that an optimal solution to a constrained problem (like SVM) must satisfy.
- They provide the direct link between the primal problem (finding $\mathbf{w}$) and the dual problem (finding $\boldsymbol{\alpha}$).

A Deeper Look: KKT Conditions

- KKT (Karush-Kuhn-Tucker) conditions are the formal rules that an optimal solution to a constrained problem (like SVM) must satisfy.
- They provide the direct link between the primal problem (finding $\mathbf{w}$) and the dual problem (finding α).
- The most important rule for interpretation is Complementary Slackness.

A Deeper Look: KKT Conditions

- KKT (Karush-Kuhn-Tucker) conditions are the formal rules that an optimal solution to a constrained problem (like SVM) must satisfy.
- They provide the direct link between the primal problem (finding $\mathbf{w}$) and the dual problem (finding $\boldsymbol{\alpha}$).
- The most important rule for interpretation is Complementary Slackness.
- This rule connects the dual variable α_n to the primal constraint for point n. In simple terms, it means:

Either the constraint is active (the point is on/inside the margin),
or the dual variable α_n for that constraint must be zero.

A Deeper Look: KKT Conditions

- KKT (Karush-Kuhn-Tucker) conditions are the formal rules that an optimal solution to a constrained problem (like SVM) must satisfy.
- They provide the direct link between the primal problem (finding $\mathbf{w}$) and the dual problem (finding $\boldsymbol{\alpha}$).
- The most important rule for interpretation is Complementary Slackness.
- This rule connects the dual variable α_n to the primal constraint for point n. In simple terms, it means:

Either the constraint is active (the point is on/inside the margin),
or the dual variable α_n for that constraint must be zero.

- This is the formal reason for the three cases:
  - **Case 1: Point is outside margin.**
    - $y_n \mathbf{x}_n^\top \mathbf{w} > 1$. The constraint is *inactive* (slack).
    - KKT conditions force $\implies \alpha_n = 0$.
  - **Case 2 & 3: Point is on or inside margin.**
    - $y_n \mathbf{x}_n^\top \mathbf{w} \leq 1$. The constraint is *active* (tight).
    - KKT conditions allow $\implies \alpha_n > 0$.
    - These are the Support Vectors.

Optimization: Coordinate Ascent

Goal: Maximize the dual $g(\boldsymbol{\alpha}) = \boldsymbol{\alpha}^\top \mathbf{1} - \frac{1}{2\lambda N^2}\boldsymbol{\alpha}^\top \mathbf{Q}\boldsymbol{\alpha}$ subject to $\alpha_n \in [0, 1]$.

Idea: Update one coordinate at a time, keeping others fixed.

initialize $\boldsymbol{\alpha}^{(0)} \in \mathbb{R}^N$ (e.g., all zeros)
for $t = 0$ to maxIter **do**
    sample a coordinate n randomly from $1 \dots N$.
    optimize g w.r.t. that single coordinate α_n (which is just a 1D quadratic problem).
$$u^\star \leftarrow \arg\max_{u \in \mathbb{R}} g\left(\alpha_1^{(t)}, \dots, \alpha_{n-1}^{(t)}, u, \alpha_{n+1}^{(t)}, \dots, \alpha_N^{(t)}\right)$$
    clip the result to respect the “box” constraint:
$$u^\star \leftarrow \min(1, \max(0, u^\star))$$
    update $\alpha_n^{(t+1)} \leftarrow u^\star$

Issues with SVM

- There is no obvious probabilistic interpretation (unlike Logistic Regression).
  - The output is a “score” $\mathbf{x}^\top \mathbf{w}$, not a probability $p(y = 1|\mathbf{x})$.
  - (Though this can be mitigated with “Platt Scaling”).

Issues with SVM

- There is no obvious probabilistic interpretation (unlike Logistic Regression).
  - The output is a “score” $\mathbf{x}^\top \mathbf{w}$, not a probability $p(y = 1|\mathbf{x})$.
  - (Though this can be mitigated with “Platt Scaling”).
- Extension to multi-class classification is non-trivial. Common approaches:
  - **One-vs-Rest (OvR):** Train K binary classifiers (class k vs. all others).
  - **One-vs-One (OvO):** Train $\frac{K(K-1)}{2}$ binary classifiers (class i vs. class j).

Table of Contents

Motivation: Why Kernelize?

- We just saw that the SVM dual depends only on inner products $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.

Motivation: Why Kernelize?

- We just saw that the SVM dual depends only on inner products $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This $N \times N$ matrix is called a kernel matrix.

Motivation: Why Kernelize?

- We just saw that the SVM dual depends only on inner products $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This $N \times N$ matrix is called a kernel matrix.
- Let's see if this property holds for other algorithms, like Ridge Regression.

Motivation: Why Kernelize?

- We just saw that the SVM dual depends only on inner products $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This $N \times N$ matrix is called a kernel matrix.
- Let's see if this property holds for other algorithms, like Ridge Regression.
- This will lead us to a powerful general idea: the Kernel Trick.

Motivation: Why Kernelize?

- We just saw that the SVM dual depends only on inner products $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This $N \times N$ matrix is called a kernel matrix.
- Let's see if this property holds for other algorithms, like Ridge Regression.
- This will lead us to a powerful general idea: the Kernel Trick.
- This trick allows us to use highly complex, non-linear, and even infinite-dimensional feature maps *without* paying the computational cost.

Alternative Formulation of Ridge Regression

Recall the **primal** problem for Ridge Regression:

$$\min_{\mathbf{w}} \quad \frac{1}{2N} \left\| \mathbf{y} - \mathbf{X}\mathbf{w} \right\|^2 + \frac{\lambda}{2} \left\| \mathbf{w} \right\|^2$$

Alternative Formulation of Ridge Regression

Recall the **primal** problem for Ridge Regression:

$$\min_{\mathbf{w}} \quad \frac{1}{2N}\left\|\mathbf{y} - \mathbf{X}\mathbf{w}\right\|^2 + \frac{\lambda}{2}\left\|\mathbf{w}\right\|^2$$

The solution $\mathbf{w}^\star \in \mathbb{R}^D$ is:

$$\mathbf{w}^\star = \frac{1}{N}\left(\frac{1}{N}\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_D\right)^{-1}\mathbf{X}^\top\mathbf{y}$$

Alternative Formulation of Ridge Regression

Recall the **primal** problem for Ridge Regression:

$$\min_{\mathbf{w}} \quad \frac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

The solution $\mathbf{w}^\star \in \mathbb{R}^D$ is:

$$\mathbf{w}^\star = \frac{1}{N}(\frac{1}{N}\mathbf{X}^\top\mathbf{X} + \lambda \mathbf{I}_D)^{-1}\mathbf{X}^\top\mathbf{y}$$

- The main computation is inverting a $D \times D$ matrix.

Alternative Formulation of Ridge Regression

Recall the **primal** problem for Ridge Regression:

$$\min_{\mathbf{w}} \quad \frac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \frac{\lambda}{2} \|\mathbf{w}\|^2$$

The solution $\mathbf{w}^\star \in \mathbb{R}^D$ is:

$$\mathbf{w}^\star = \frac{1}{N}(\frac{1}{N}\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^\top\mathbf{y}$$

- The main computation is inverting a $D \times D$ matrix.
- This costs $O(D^3 + ND^2)$. This is great if $D \ll N$.

Alternative Formulation of Ridge Regression

Recall the **primal** problem for Ridge Regression:

$$\min_{\mathbf{w}} \quad \frac{1}{2N}\|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

The solution $\mathbf{w}^\star \in \mathbb{R}^D$ is:

$$\mathbf{w}^\star = \frac{1}{N}(\frac{1}{N}\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^\top\mathbf{y}$$

- The main computation is inverting a $D \times D$ matrix.
- This costs $O(D^3 + ND^2)$. This is great if $D \ll N$.
- But what if $D \gg N$? (e.g., 100 samples, 1,000,000 features)

Alternative Formulation (Dual)

We claim the solution can be written alternatively as:

$$\mathbf{w}^\star = \frac{1}{N}\mathbf{X}^\top \left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N\right)^{-1}\mathbf{y} \tag{5}$$

Alternative Formulation (Dual)

We claim the solution can be written alternatively as:

$$\mathbf{w}^\star = \frac{1}{N}\mathbf{X}^\top\left(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}_N\right)^{-1}\mathbf{y} \tag{5}$$

- The main computation is now inverting an $N \times N$ matrix.

Alternative Formulation (Dual)

We claim the solution can be written alternatively as:

$$\mathbf{w}^{\star} = \frac{1}{N}\mathbf{X}^{\top}\left(\frac{1}{N}\mathbf{X}\mathbf{X}^{\top} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{y} \tag{5}$$

- The main computation is now inverting an $N \times N$ matrix.
- This costs $O(N^3 + DN^2)$.

Alternative Formulation (Dual)

We claim the solution can be written alternatively as:

$$\mathbf{w}^\star = \tfrac{1}{N}\mathbf{X}^\top(\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}_N)^{-1}\mathbf{y} \tag{5}$$

- The main computation is now inverting an $N \times N$ matrix.
- This costs $O(N^3 + DN^2)$.
- This is much better if $N \ll D$!

Alternative Formulation (Dual)

We claim the solution can be written alternatively as:

$$\mathbf{w}^\star = \tfrac{1}{N}\mathbf{X}^\top(\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda\mathbf{I}_N)^{-1}\mathbf{y} \tag{5}$$

- The main computation is now inverting an $N \times N$ matrix.
- This costs $O(N^3 + DN^2)$.
- This is much better if $N \ll D$!

This relies on the “push-through” matrix identity:

$$(\mathbf{PQ} + \mathbf{I}_N)^{-1}\mathbf{P} = \mathbf{P}(\mathbf{QP} + \mathbf{I}_M)^{-1}$$

(Here, $\mathbf{P} = \mathbf{X}^\top$ and $\mathbf{Q} = \frac{1}{\lambda N}\mathbf{X}$)

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the “push-through” identity).

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the “push-through” identity).
- Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ and $\mathbf{Q} \in \mathbb{R}^{n \times m}$.

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the “push-through” identity).
- Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ and $\mathbf{Q} \in \mathbb{R}^{n \times m}$.
- We have: $\mathbf{P}(\mathbf{Q}\mathbf{P} + \mathbf{I}_n) = \mathbf{P}\mathbf{Q}\mathbf{P} + \mathbf{P} = (\mathbf{P}\mathbf{Q} + \mathbf{I}_m)\mathbf{P}$.

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the “push-through” identity).
- Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ and $\mathbf{Q} \in \mathbb{R}^{n \times m}$.
- We have: $\mathbf{P}(\mathbf{QP} + \mathbf{I}_n) = \mathbf{PQP} + \mathbf{P} = (\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}$.
- Assuming the inverses exist, we can multiply by $(\mathbf{PQ} + \mathbf{I}_m)^{-1}$ on the left and $(\mathbf{QP} + \mathbf{I}_n)^{-1}$ on the right:

$$
\begin{aligned}
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)(\mathbf{QP} + \mathbf{I}_n)^{-1} &= (\mathbf{PQ} + \mathbf{I}_m)^{-1}(\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1} \\
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P} &= \mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1}
\end{aligned}
$$

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the "push-through" identity).
- Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ and $\mathbf{Q} \in \mathbb{R}^{n \times m}$.
- We have: $\mathbf{P}(\mathbf{QP} + \mathbf{I}_n) = \mathbf{PQP} + \mathbf{P} = (\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}$.
- Assuming the inverses exist, we can multiply by $(\mathbf{PQ} + \mathbf{I}_m)^{-1}$ on the left and $(\mathbf{QP} + \mathbf{I}_n)^{-1}$ on the right:

$$
\begin{aligned}
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)(\mathbf{QP} + \mathbf{I}_n)^{-1} &= (\mathbf{PQ} + \mathbf{I}_m)^{-1}(\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1} \\
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P} &= \mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1}
\end{aligned}
$$

- We get our result by setting $\mathbf{P} = \mathbf{X}^\top$ and $\mathbf{Q} = \frac{1}{\lambda N}\mathbf{X}$.

Proof of Alternative Formulation

- This relies on a common matrix identity (sometimes called the “push-through” identity).
- Let $\mathbf{P} \in \mathbb{R}^{m \times n}$ and $\mathbf{Q} \in \mathbb{R}^{n \times m}$.
- We have: $\mathbf{P}(\mathbf{QP} + \mathbf{I}_n) = \mathbf{PQP} + \mathbf{P} = (\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}$.
- Assuming the inverses exist, we can multiply by $(\mathbf{PQ} + \mathbf{I}_m)^{-1}$ on the left and $(\mathbf{QP} + \mathbf{I}_n)^{-1}$ on the right:

$$
\begin{aligned}
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)(\mathbf{QP} + \mathbf{I}_n)^{-1} &= (\mathbf{PQ} + \mathbf{I}_m)^{-1}(\mathbf{PQ} + \mathbf{I}_m)\mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1} \\
(\mathbf{PQ} + \mathbf{I}_m)^{-1}\mathbf{P} &= \mathbf{P}(\mathbf{QP} + \mathbf{I}_n)^{-1}
\end{aligned}
$$

- We get our result by setting $\mathbf{P} = \mathbf{X}^\top$ and $\mathbf{Q} = \frac{1}{\lambda N}\mathbf{X}$.
- This switches a $D \times D$ inverse for an $N \times N$ inverse.

The Representer Theorem

The alternative form $\mathbf{w}^{\star} = \mathbf{X}^{\top}\boldsymbol{\alpha}^{\star}$ is not a coincidence.

$$\text{Define } \boldsymbol{\alpha}^{\star} = \frac{1}{N}\left(\frac{1}{N}\mathbf{X}\mathbf{X}^{\top} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{y}. \quad \text{Then } \mathbf{w}^{\star} = \mathbf{X}^{\top}\boldsymbol{\alpha}^{\star}.$$

The Representer Theorem

The alternative form $\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star$ is not a coincidence.

Define $\boldsymbol{\alpha}^\star = \frac{1}{N}(\frac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$. Then $\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star$.

This shows $\mathbf{w}^\star$ lies in the space spanned by the feature vectors (the rows of $\mathbf{X}$).

The Representer Theorem

The alternative form $\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star$ is not a coincidence.

$$\text{Define } \boldsymbol{\alpha}^\star = \tfrac{1}{N}\left(\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N\right)^{-1}\mathbf{y}. \quad \text{Then } \mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star.$$

This shows $\mathbf{w}^\star$ lies in the space spanned by the feature vectors (the rows of $\mathbf{X}$).

The Representer Theorem (General) For any loss function $\mathcal{L}_n$, the solution $\mathbf{w}^\star$ to:

$$\min_{\mathbf{w}} \; \tfrac{1}{N}\textstyle\sum_{n=1}^{N} \mathcal{L}_n(\mathbf{x}_n^\top \mathbf{w}, y_n) + \tfrac{\lambda}{2}\left\|\mathbf{w}\right\|^2$$

can be written as a linear combination of the data points:

$$\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star = \textstyle\sum_{n=1}^{N} \alpha_n^\star \mathbf{x}_n$$

The Representer Theorem

The alternative form $\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star$ is not a coincidence.

$$\text{Define } \boldsymbol{\alpha}^\star = \tfrac{1}{N}(\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)^{-1}\mathbf{y}. \quad \text{Then } \mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star.$$

This shows $\mathbf{w}^\star$ lies in the space spanned by the feature vectors (the rows of $\mathbf{X}$).

The Representer Theorem (General) For any loss function $\mathcal{L}_n$, the solution $\mathbf{w}^\star$ to:

$$\min_{\mathbf{w}} \; \tfrac{1}{N}\sum_{n=1}^N \mathcal{L}_n(\mathbf{x}_n^\top \mathbf{w}, y_n) + \tfrac{\lambda}{2}\|\mathbf{w}\|^2$$

can be written as a linear combination of the data points:

$$\mathbf{w}^\star = \mathbf{X}^\top \boldsymbol{\alpha}^\star = \sum_{n=1}^N \alpha_n^\star \mathbf{x}_n$$

Proof (Intuition): Decompose $\mathbf{w} = \mathbf{w}_\parallel + \mathbf{w}_\perp$, where $\mathbf{w}_\parallel$ is in the span of $\mathbf{X}$ and $\mathbf{w}_\perp$ is orthogonal to it.

- The loss term $\mathcal{L}_n$ only depends on $\mathbf{x}_n^\top \mathbf{w} = \mathbf{x}_n^\top \mathbf{w}_\parallel$.
- The regularization term $\|\mathbf{w}\|^2 = \|\mathbf{w}_\parallel\|^2 + \|\mathbf{w}_\perp\|^2$.
- To minimize the cost, we must set $\mathbf{w}_\perp = 0$. Thus, $\mathbf{w}^\star = \mathbf{w}_\parallel^\star \in \text{span}(\mathbf{X})$.

Representer Theorem (Formal Proof)

- Let $\mathbf{w}^*$ be any optimal solution.

Representer Theorem (Formal Proof)

- Let $\mathbf{w}^*$ be any optimal solution.
- We can always decompose $\mathbf{w}^*$ into two orthogonal components:

$$\mathbf{w}^* = \underbrace{\sum_{n=1}^{N} \alpha_n \mathbf{x}_n}_{\mathbf{w}_{\parallel} \in \text{span}(\mathbf{X})} + \underbrace{\mathbf{u}}_{\mathbf{w}_{\perp} \perp \text{span}(\mathbf{X})}$$

where $\mathbf{u}^\top \mathbf{x}_n = 0$ for all n.

Representer Theorem (Formal Proof)

- Let $\mathbf{w}^*$ be any optimal solution.
- We can always decompose $\mathbf{w}^*$ into two orthogonal components:

$$\mathbf{w}^* = \underbrace{\sum_{n=1}^{N} \alpha_n \mathbf{x}_n}_{\mathbf{w}_\parallel \in \text{span}(\mathbf{X})} + \underbrace{\mathbf{u}}_{\mathbf{w}_\perp \perp \text{span}(\mathbf{X})}$$

  where $\mathbf{u}^\top \mathbf{x}_n = 0$ for all n.
- **Part 1: The Loss Term.** The loss only depends on $\mathbf{w}_\parallel$:

$$\mathbf{x}_n^\top \mathbf{w}^* = \mathbf{x}_n^\top (\mathbf{w}_\parallel + \mathbf{u}) = \mathbf{x}_n^\top \mathbf{w}_\parallel + \mathbf{x}_n^\top \mathbf{u} = \mathbf{x}_n^\top \mathbf{w}_\parallel$$

  So, $l(\mathbf{x}_n^\top \mathbf{w}^*, y_n) = l(\mathbf{x}_n^\top \mathbf{w}_\parallel, y_n)$.

Representer Theorem (Formal Proof)

- Let $\mathbf{w}^*$ be any optimal solution.
- We can always decompose $\mathbf{w}^*$ into two orthogonal components:

$$\mathbf{w}^* = \underbrace{\sum_{n=1}^{N} \alpha_n \mathbf{x}_n}_{\mathbf{w}_{\parallel} \in \text{span}(\mathbf{X})} + \underbrace{\mathbf{u}}_{\mathbf{w}_{\perp} \perp \text{span}(\mathbf{X})}$$

  where $\mathbf{u}^\top \mathbf{x}_n = 0$ for all n.
- **Part 1: The Loss Term.** The loss only depends on $\mathbf{w}_{\parallel}$:

$$\mathbf{x}_n^\top \mathbf{w}^* = \mathbf{x}_n^\top (\mathbf{w}_{\parallel} + \mathbf{u}) = \mathbf{x}_n^\top \mathbf{w}_{\parallel} + \mathbf{x}_n^\top \mathbf{u} = \mathbf{x}_n^\top \mathbf{w}_{\parallel}$$

  So, $l(\mathbf{x}_n^\top \mathbf{w}^*, y_n) = l(\mathbf{x}_n^\top \mathbf{w}_{\parallel}, y_n)$.
- **Part 2: The Regularization Term.** By Pythagorean theorem (since $\mathbf{w}_{\parallel} \perp \mathbf{u}$):

$$\|\mathbf{w}^*\|^2 = \left\|\mathbf{w}_{\parallel} + \mathbf{u}\right\|^2 = \left\|\mathbf{w}_{\parallel}\right\|^2 + \|\mathbf{u}\|^2$$

  This means $\left\|\mathbf{w}_{\parallel}\right\|^2 \leq \|\mathbf{w}^*\|^2$.

Representer Theorem (Formal Proof)

- Let $\mathbf{w}^*$ be any optimal solution.
- We can always decompose $\mathbf{w}^*$ into two orthogonal components:

$$\mathbf{w}^* = \underbrace{\sum_{n=1}^{N} \alpha_n \mathbf{x}_n}_{\mathbf{w}_{||} \in \text{span}(\mathbf{X})} + \underbrace{\mathbf{u}}_{\mathbf{w}_{\perp} \perp \text{span}(\mathbf{X})}$$

  where $\mathbf{u}^\top \mathbf{x}_n = 0$ for all n.
- **Part 1: The Loss Term.** The loss only depends on $\mathbf{w}_{||}$:

$$\mathbf{x}_n^\top \mathbf{w}^* = \mathbf{x}_n^\top (\mathbf{w}_{||} + \mathbf{u}) = \mathbf{x}_n^\top \mathbf{w}_{||} + \mathbf{x}_n^\top \mathbf{u} = \mathbf{x}_n^\top \mathbf{w}_{||}$$

  So, $l(\mathbf{x}_n^\top \mathbf{w}^*, y_n) = l(\mathbf{x}_n^\top \mathbf{w}_{||}, y_n)$.
- **Part 2: The Regularization Term.** By Pythagorean theorem (since $\mathbf{w}_{||} \perp \mathbf{u}$):

$$\|\mathbf{w}^*\|^2 = \left\|\mathbf{w}_{||} + \mathbf{u}\right\|^2 = \left\|\mathbf{w}_{||}\right\|^2 + \|\mathbf{u}\|^2$$

  This means $\left\|\mathbf{w}_{||}\right\|^2 \leq \|\mathbf{w}^*\|^2$.
- **Conclusion:** The total cost for $\mathbf{w}_{||}$ is always less than or equal to the cost for $\mathbf{w}^*$.

Kernelized Ridge Regression

- The Representer Theorem lets us write an equivalent optimization problem in terms of $\boldsymbol{\alpha} \in \mathbb{R}^N$ instead of $\mathbf{w} \in \mathbb{R}^D$.

Kernelized Ridge Regression

- The Representer Theorem lets us write an equivalent optimization problem in terms of $\boldsymbol{\alpha} \in \mathbb{R}^N$ instead of $\mathbf{w} \in \mathbb{R}^D$.
- **Primal Problem (in $\mathbf{w}$):**

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \ \frac{1}{2N} \left\|\mathbf{y} - \mathbf{X}\mathbf{w}\right\|^2 + \frac{\lambda}{2} \left\|\mathbf{w}\right\|^2$$

Kernelized Ridge Regression

- The Representer Theorem lets us write an equivalent optimization problem in terms of $\boldsymbol{\alpha} \in \mathbb{R}^N$ instead of $\mathbf{w} \in \mathbb{R}^D$.
- **Primal Problem (in $\mathbf{w}$):**

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \; \tfrac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \tfrac{\lambda}{2} \|\mathbf{w}\|^2$$

- **Dual Problem (in α):**

$$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}} \; \frac{1}{2}\boldsymbol{\alpha}^\top (\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)\boldsymbol{\alpha} - \tfrac{1}{N}\boldsymbol{\alpha}^\top \mathbf{y}$$

Kernelized Ridge Regression

- The Representer Theorem lets us write an equivalent optimization problem in terms of $\boldsymbol{\alpha} \in \mathbb{R}^N$ instead of $\mathbf{w} \in \mathbb{R}^D$.
- **Primal Problem (in $\mathbf{w}$):**

$$\mathbf{w}^\star = \arg\min_{\mathbf{w}} \ \tfrac{1}{2N} \|\mathbf{y} - \mathbf{X}\mathbf{w}\|^2 + \tfrac{\lambda}{2} \|\mathbf{w}\|^2$$

- **Dual Problem (in $\boldsymbol{\alpha}$):**

$$\boldsymbol{\alpha}^\star = \arg\min_{\boldsymbol{\alpha}} \ \frac{1}{2}\boldsymbol{\alpha}^\top (\tfrac{1}{N}\mathbf{X}\mathbf{X}^\top + \lambda \mathbf{I}_N)\boldsymbol{\alpha} - \tfrac{1}{N}\boldsymbol{\alpha}^\top \mathbf{y}$$

- The dual problem only depends on the data through the kernel matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.

The Kernel Matrix

- The dual formulations for SVM and Ridge Regression depend only on the $N \times N$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.

The Kernel Matrix

- The dual formulations for SVM and Ridge Regression depend only on the $N \times N$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This is called the Kernel Matrix or Gram Matrix.

The Kernel Matrix

- The dual formulations for SVM and Ridge Regression depend only on the $N \times N$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This is called the Kernel Matrix or Gram Matrix.
- Each element $\mathbf{K}_{ij}$ is the inner product between two data points:

$$\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$$

The Kernel Matrix

- The dual formulations for SVM and Ridge Regression depend only on the $N \times N$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This is called the Kernel Matrix or Gram Matrix.
- Each element $\mathbf{K}_{ij}$ is the inner product between two data points:

$$\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$$

- $\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \begin{pmatrix} \mathbf{x}_1^\top \mathbf{x}_1 & \mathbf{x}_1^\top \mathbf{x}_2 & \cdots & \mathbf{x}_1^\top \mathbf{x}_N \\ \mathbf{x}_2^\top \mathbf{x}_1 & \mathbf{x}_2^\top \mathbf{x}_2 & \cdots & \mathbf{x}_2^\top \mathbf{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_N^\top \mathbf{x}_1 & \mathbf{x}_N^\top \mathbf{x}_2 & \cdots & \mathbf{x}_N^\top \mathbf{x}_N \end{pmatrix}$

The Kernel Matrix

- The dual formulations for SVM and Ridge Regression depend only on the $N \times N$ matrix $\mathbf{K} = \mathbf{X}\mathbf{X}^\top$.
- This is called the Kernel Matrix or Gram Matrix.
- Each element $\mathbf{K}_{ij}$ is the inner product between two data points:

$$\mathbf{K}_{ij} = \mathbf{x}_i^\top \mathbf{x}_j$$

- $\mathbf{K} = \mathbf{X}\mathbf{X}^\top = \begin{pmatrix} \mathbf{x}_1^\top \mathbf{x}_1 & \mathbf{x}_1^\top \mathbf{x}_2 & \cdots & \mathbf{x}_1^\top \mathbf{x}_N \\ \mathbf{x}_2^\top \mathbf{x}_1 & \mathbf{x}_2^\top \mathbf{x}_2 & \cdots & \mathbf{x}_2^\top \mathbf{x}_N \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{x}_N^\top \mathbf{x}_1 & \mathbf{x}_N^\top \mathbf{x}_2 & \cdots & \mathbf{x}_N^\top \mathbf{x}_N \end{pmatrix}$
- This matrix is symmetric ($\mathbf{K} = \mathbf{K}^\top$) and positive semi-definite ($\mathbf{K} \geq 0$).

The Kernel Trick: Feature Maps

- What if our features $\mathbf{x}$ aren't good enough? We can augment them with a feature map $\phi(\mathbf{x})$.

The Kernel Trick: Feature Maps

- What if our features $\mathbf{x}$ aren't good enough? We can augment them with a feature map $\phi(\mathbf{x})$.
- Example: $\mathbf{x} = [x_1, x_2] \rightarrow \phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$

The Kernel Trick: Feature Maps

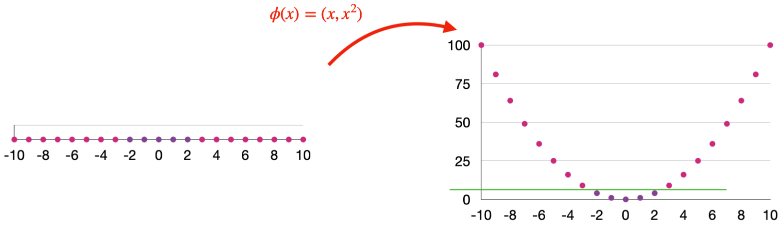
- What if our features $\mathbf{x}$ aren't good enough? We can augment them with a feature map $\phi(\mathbf{x})$.
- Example: $\mathbf{x} = [x_1, x_2] \rightarrow \phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1 x_2]$
- Our algorithms now run in this new, higher-dimensional feature space.
  - $\mathbf{w}$ is now in this new space.
  - The kernel matrix becomes $\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$.

The Kernel Trick: Feature Maps

- What if our features $\mathbf{x}$ aren't good enough? We can augment them with a feature map $\phi(\mathbf{x})$.
- Example: $\mathbf{x} = [x_1, x_2] \to \phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1 x_2]$
- Our algorithms now run in this new, higher-dimensional feature space.
  - $\mathbf{w}$ is now in this new space.
  - The kernel matrix becomes $\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$.
- **Problem:** If $\phi(\mathbf{x})$ is very high-dimensional (or infinite!), computing $\phi(\mathbf{x})$ for all data and then $\mathbf{K}$ is extremely expensive or impossible.

The Kernel Trick: Feature Maps

- What if our features $\mathbf{x}$ aren't good enough? We can augment them with a feature map $\phi(\mathbf{x})$.
- Example: $\mathbf{x} = [x_1, x_2] \rightarrow \phi(\mathbf{x}) = [x_1^2, x_2^2, \sqrt{2}x_1x_2]$
- Our algorithms now run in this new, higher-dimensional feature space.
    - $\mathbf{w}$ is now in this new space.
    - The kernel matrix becomes $\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$.
- **Problem:** If $\phi(\mathbf{x})$ is very high-dimensional (or infinite!), computing $\phi(\mathbf{x})$ for all data and then $\mathbf{K}$ is extremely expensive or impossible.
- **Example:** Data not linearly separable in 1D.



Kernel Matrix with Feature Maps

- When we use a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{\tilde{D}}$, our data matrix $\mathbf{X}$ becomes $\mathbf{\Phi}$.

Kernel Matrix with Feature Maps

- When we use a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{\tilde{D}}$, our data matrix $\mathbf{X}$ becomes $\mathbf{\Phi}$.
- The kernel matrix is now computed in the new feature space:

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

Kernel Matrix with Feature Maps

- When we use a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{\tilde{D}}$, our data matrix $\mathbf{X}$ becomes $\mathbf{\Phi}$.
- The kernel matrix is now computed in the new feature space:

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- This is $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$.

Kernel Matrix with Feature Maps

- When we use a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{\tilde{D}}$, our data matrix $\mathbf{X}$ becomes $\mathbf{\Phi}$.
- The kernel matrix is now computed in the new feature space:

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- This is $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$.
- **The Problem:** If the new dimension $\tilde{D}$ is very large (e.g., $D 1 \tilde{D}$), computing *one* inner product $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ costs $O(\tilde{D})$.

Kernel Matrix with Feature Maps

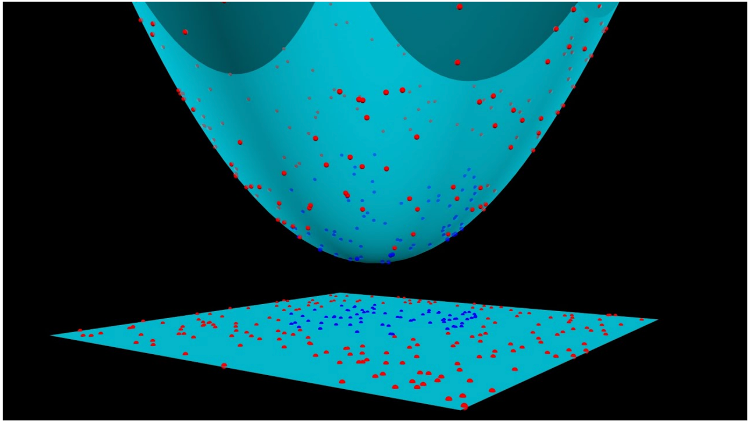
- When we use a feature map $\phi : \mathbb{R}^D \to \mathbb{R}^{\tilde{D}}$, our data matrix $\mathbf{X}$ becomes $\mathbf{\Phi}$.
- The kernel matrix is now computed in the new feature space:

$$\mathbf{K}_{ij} = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- This is $\mathbf{K} = \mathbf{\Phi}\mathbf{\Phi}^\top$.
- **The Problem:** If the new dimension $\tilde{D}$ is very large (e.g., $D1\tilde{D}$), computing *one* inner product $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$ costs $O(\tilde{D})$.
- This is too expensive, or impossible if $\tilde{D} = \infty$.

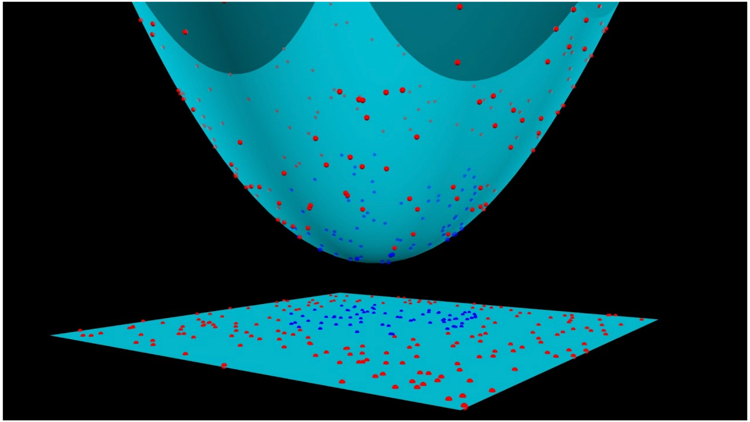
Usefulness of Feature Spaces

Data that is not linearly separable in 2D...



Usefulness of Feature Spaces

Data that is not linearly separable in 2D...



...can become linearly separable when mapped to a higher dimension (3D).

The Kernel Trick: The "Shortcut"

The Kernel Trick: If we can find a kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ that computes the inner product in the feature space *directly*, we never need to build $\phi(\mathbf{x})$.

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

The Kernel Trick: The “Shortcut”

The Kernel Trick: If we can find a kernel function $\kappa(\mathbf{x}, \mathbf{x}')$ that computes the inner product in the feature space *directly*, we never need to build $\phi(\mathbf{x})$.

$$\kappa(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$$

- We only need to compute the $N \times N$ kernel matrix $\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j)$.
- All “kernelized” algorithms (like SVM dual, Ridge dual) only need $\mathbf{K}$.
- We can work in an incredibly high-dimensional space, as long as we have an efficient $\kappa(\cdot, \cdot)$.

Example 1: Polynomial Kernel

- **Kernel Function:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$

Example 1: Polynomial Kernel

- **Kernel Function:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$
- Let $\mathbf{x} \in \mathbb{R}^3$. $\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + x_3 x_3')^2$

Example 1: Polynomial Kernel

- **Kernel Function:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$
- Let $\mathbf{x} \in \mathbb{R}^3$. $\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + x_3 x_3')^2$
- **Implicit Feature Map $\phi(\mathbf{x}) \in \mathbb{R}^6$:**

$$\phi(\mathbf{x})^\top = \left[x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3\right]$$

Example 1: Polynomial Kernel

- **Kernel Function:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$
- Let $\mathbf{x} \in \mathbb{R}^3$. $\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + x_3 x_3')^2$
- **Implicit Feature Map $\phi(\mathbf{x}) \in \mathbb{R}^6$:**

$$\phi(\mathbf{x})^\top = \left[x_1^2, x_2^2, x_3^2, \sqrt{2}x_1 x_2, \sqrt{2}x_1 x_3, \sqrt{2}x_2 x_3\right]$$

- **Cost:**
  - Computing $\kappa(\mathbf{x}, \mathbf{x}')$: 1 inner product ($O(D)$), 1 multiplication.
  - Computing $\phi(\mathbf{x})$ and then the dot product: $O(D^2)$ operations.

Example 1: Polynomial Kernel

- **Kernel Function:** $\kappa(\mathbf{x}, \mathbf{x}') = (\mathbf{x}^\top \mathbf{x}')^2$
- Let $\mathbf{x} \in \mathbb{R}^3$. $\kappa(\mathbf{x}, \mathbf{x}') = (x_1 x_1' + x_2 x_2' + x_3 x_3')^2$
- **Implicit Feature Map** $\phi(\mathbf{x}) \in \mathbb{R}^6$**:**

$$\phi(\mathbf{x})^\top = \left[x_1^2, x_2^2, x_3^2, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, \sqrt{2}x_2x_3\right]$$

- **Cost:**
  - Computing $\kappa(\mathbf{x}, \mathbf{x}')$: 1 inner product ($O(D)$), 1 multiplication.
  - Computing $\phi(\mathbf{x})$ and then the dot product: $O(D^2)$ operations.
- The kernel function is a huge computational shortcut!

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

- This is a very popular and powerful kernel. (Note: $\gamma = 1$ in your notes, e.g., $\exp(-(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')))$

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

- This is a very popular and powerful kernel. (Note: $\gamma = 1$ in your notes, e.g., $\exp(-(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')))$
- **Implicit Feature Map $\phi(\mathbf{x})$:**
  - The corresponding feature space is **infinite-dimensional**!

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

- This is a very popular and powerful kernel. (Note: $\gamma = 1$ in your notes, e.g., $\exp(-(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')))$
- **Implicit Feature Map $\phi(\mathbf{x})$:**
  - The corresponding feature space is **infinite-dimensional**!
- We can see this by expanding the simple 1D case ($x \in \mathbb{R}$, $\gamma = 1$):

$$\kappa(x, x') = e^{-(x-x')^2} = e^{-x^2} e^{-(x')^2} e^{2xx'}$$
$$= e^{-x^2} e^{-(x')^2} \sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}$$

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

- This is a very popular and powerful kernel. (Note: $\gamma = 1$ in your notes, e.g., $\exp(-(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')))$
- **Implicit Feature Map $\phi(\mathbf{x})$:**
  - The corresponding feature space is **infinite-dimensional**!
- We can see this by expanding the simple 1D case ($x \in \mathbb{R}$, $\gamma = 1$):

$$\begin{aligned} \kappa(x, x') &= e^{-(x-x')^2} = e^{-x^2} e^{-(x')^2} e^{2xx'} \\ &= e^{-x^2} e^{-(x')^2} \sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!} \end{aligned}$$

- This is an inner product between two infinite vectors:

$$\phi(x)^\top = e^{-x^2}\left[\ldots, \sqrt{\frac{2^k}{k!}} x^k, \ldots\right]$$

Example 2: RBF (Gaussian) Kernel

- **Kernel Function:**

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

- This is a very popular and powerful kernel. (Note: $\gamma = 1$ in your notes, e.g., $\exp(-(\mathbf{x} - \mathbf{x}')^\top(\mathbf{x} - \mathbf{x}')))$
- **Implicit Feature Map $\phi(\mathbf{x})$:**
  - The corresponding feature space is **infinite-dimensional**!
- We can see this by expanding the simple 1D case ($x \in \mathbb{R}$, $\gamma = 1$):

$$\begin{aligned}\kappa(x, x') &= e^{-(x-x')^2} = e^{-x^2} e^{-(x')^2} e^{2xx'} \\ &= e^{-x^2} e^{-(x')^2} \sum_{k=0}^{\infty} \frac{2^k (x)^k (x')^k}{k!}\end{aligned}$$

- This is an inner product between two infinite vectors:

$$\phi(x)^\top = e^{-x^2}\left[\ldots, \sqrt{\frac{2^k}{k!}} x^k, \ldots\right]$$

- The Kernel Trick lets us compute this impossible-to-build inner product.

Building New Kernels from Old

If κ_1 and κ_2 are valid kernels, so are:

- **Positive Scaling:** $\kappa(\mathbf{x}, \mathbf{x}') = c\,\kappa_1(\mathbf{x}, \mathbf{x}')$ for $c \geq 0$.

Building New Kernels from Old

If κ_1 and κ_2 are valid kernels, so are:

- **Positive Scaling:** $\kappa(\mathbf{x}, \mathbf{x}') = c\,\kappa_1(\mathbf{x}, \mathbf{x}')$ for $c \geq 0$.
- **Sum:** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$.
  - Proof: $\phi(\cdot) = [\phi_1(\cdot), \phi_2(\cdot)]$

Building New Kernels from Old

If κ_1 and κ_2 are valid kernels, so are:

- **Positive Scaling:** $\kappa(\mathbf{x}, \mathbf{x}') = c\,\kappa_1(\mathbf{x}, \mathbf{x}')$ for $c \geq 0$.
- **Sum:** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$.
  - Proof: $\phi(\cdot) = [\phi_1(\cdot), \phi_2(\cdot)]$
- **Product:** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$.
  - Proof: $\phi(\cdot)$ has entries $(\phi_1(\cdot))_i(\phi_2(\cdot))_j$

Building New Kernels from Old

If κ_1 and κ_2 are valid kernels, so are:

- **Positive Scaling:** $\kappa(\mathbf{x}, \mathbf{x}') = c\,\kappa_1(\mathbf{x}, \mathbf{x}')$ for $c \geq 0$.
- **Sum:** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}') + \kappa_2(\mathbf{x}, \mathbf{x}')$.
  - Proof: $\phi(\cdot) = [\phi_1(\cdot), \phi_2(\cdot)]$
- **Product:** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$.
  - Proof: $\phi(\cdot)$ has entries $(\phi_1(\cdot))_i(\phi_2(\cdot))_j$
- **Function mapping:** $\kappa(\mathbf{x}, \mathbf{x}') = f(\mathbf{x})f(\mathbf{x}')$ for any real-valued f.

Proofs for Building Kernels

- **Proof (Sum):** $\kappa(\mathbf{x}, \mathbf{x}') = \alpha\kappa_1(\mathbf{x}, \mathbf{x}') + \beta\kappa_2(\mathbf{x}, \mathbf{x}')$
  - We can define a new feature map $\phi(\mathbf{x})$ by concatenating the scaled original feature maps:
$$\boldsymbol{\phi}(\mathbf{x}) = \begin{bmatrix} \sqrt{\alpha}\boldsymbol{\phi}_1(\mathbf{x}) \\ \sqrt{\beta}\boldsymbol{\phi}_2(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{D_1+D_2}$$
  - The inner product is then:
$$\boldsymbol{\phi}(\mathbf{x})^\top \boldsymbol{\phi}(\mathbf{x}') = \alpha\boldsymbol{\phi}_1(\mathbf{x})^\top \boldsymbol{\phi}_1(\mathbf{x}') + \beta\boldsymbol{\phi}_2(\mathbf{x})^\top \boldsymbol{\phi}_2(\mathbf{x}')$$

Proofs for Building Kernels

- **Proof (Sum):** $\kappa(\mathbf{x}, \mathbf{x}') = \alpha\kappa_1(\mathbf{x}, \mathbf{x}') + \beta\kappa_2(\mathbf{x}, \mathbf{x}')$
  - We can define a new feature map $\phi(\mathbf{x})$ by concatenating the scaled original feature maps:
$$\phi(\mathbf{x}) = \begin{bmatrix} \sqrt{\alpha}\phi_1(\mathbf{x}) \\ \sqrt{\beta}\phi_2(\mathbf{x}) \end{bmatrix} \in \mathbb{R}^{D_1+D_2}$$
  - The inner product is then:
$$\phi(\mathbf{x})^\top\phi(\mathbf{x}') = \alpha\phi_1(\mathbf{x})^\top\phi_1(\mathbf{x}') + \beta\phi_2(\mathbf{x})^\top\phi_2(\mathbf{x}')$$
- **Proof (Product):** $\kappa(\mathbf{x}, \mathbf{x}') = \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')$
  - We define a new feature map $\phi(\mathbf{x})$ as the tensor (outer) product of the original features.
  - The new feature vector contains all possible products $(\phi_1(\mathbf{x}))_i(\phi_2(\mathbf{x}))_j$.
  - The inner product becomes:
$$\begin{aligned}
\phi(\mathbf{x})^\top\phi(\mathbf{x}') &= \sum_{i,j}(\phi_1(\mathbf{x}))_i(\phi_2(\mathbf{x}))_j(\phi_1(\mathbf{x}'))_i(\phi_2(\mathbf{x}'))_j \\
&= \left(\sum_i(\phi_1(\mathbf{x}))_i(\phi_1(\mathbf{x}'))_i\right)\left(\sum_j(\phi_2(\mathbf{x}))_j(\phi_2(\mathbf{x}'))_j\right) \\
&= \kappa_1(\mathbf{x}, \mathbf{x}')\kappa_2(\mathbf{x}, \mathbf{x}')
\end{aligned}$$

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.
- A new data point $\mathbf{x}$ is classified by $y = \mathbf{w}^\star \cdot \phi(\mathbf{x})$.

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.
- A new data point $\mathbf{x}$ is classified by $y = \mathbf{w}^\star \cdot \phi(\mathbf{x})$.
- We can't compute $\mathbf{w}^\star$ or $\phi(\mathbf{x})$ if they are infinite!

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.
- A new data point $\mathbf{x}$ is classified by $y = \mathbf{w}^\star \cdot \phi(\mathbf{x})$.
- We can't compute $\mathbf{w}^\star$ or $\phi(\mathbf{x})$ if they are infinite!
- But we can use the Representer Theorem:

$$\mathbf{w}^\star = \sum_{n=1}^{N} \alpha_n^\star \phi(\mathbf{x}_n) \qquad \text{(using } \phi \text{ in } \mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha}\text{)}$$

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.
- A new data point $\mathbf{x}$ is classified by $y = \mathbf{w}^\star \cdot \phi(\mathbf{x})$.
- We can't compute $\mathbf{w}^\star$ or $\phi(\mathbf{x})$ if they are infinite!
- But we can use the Representer Theorem:

$$\mathbf{w}^\star = \sum_{n=1}^{N} \alpha_n^\star \phi(\mathbf{x}_n) \qquad \text{(using } \phi \text{ in } \mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha})$$

- Now, the prediction becomes:

$$\begin{aligned} y &= \left(\sum_{n=1}^{N} \alpha_n^\star \boldsymbol{\phi}(\mathbf{x}_n) \right)^\top \boldsymbol{\phi}(\mathbf{x}) \\ &= \sum_{n=1}^{N} \alpha_n^\star \left(\boldsymbol{\phi}(\mathbf{x}_n)^\top \boldsymbol{\phi}(\mathbf{x}) \right) \\ &= \sum_{n=1}^{N} \alpha_n^\star \kappa(\mathbf{x}_n, \mathbf{x}) \end{aligned}$$

Prediction with Kernels

- This whole enterprise is useless if we can't make predictions on new data.
- A new data point $\mathbf{x}$ is classified by $y = \mathbf{w}^\star \cdot \phi(\mathbf{x})$.
- We can't compute $\mathbf{w}^\star$ or $\phi(\mathbf{x})$ if they are infinite!
- But we can use the Representer Theorem:

$$\mathbf{w}^\star = \sum_{n=1}^{N} \alpha_n^\star \phi(\mathbf{x}_n) \qquad \text{(using } \phi \text{ in } \mathbf{w} = \mathbf{X}^\top \boldsymbol{\alpha}\text{)}$$

- Now, the prediction becomes:

$$\begin{aligned} y &= \left(\sum_{n=1}^{N} \alpha_n^\star \phi(\mathbf{x}_n) \right)^\top \phi(\mathbf{x}) \\ &= \sum_{n=1}^{N} \alpha_n^\star \left(\phi(\mathbf{x}_n)^\top \phi(\mathbf{x}) \right) \\ &= \sum_{n=1}^{N} \alpha_n^\star \kappa(\mathbf{x}_n, \mathbf{x}) \end{aligned}$$

Properties of Kernels: Mercer's Condition

How do we know if a function $\kappa(\mathbf{x}, \mathbf{x}')$ is a valid kernel?
(i.e., corresponds to an inner product in *some* feature space?)

Properties of Kernels: Mercer's Condition

How do we know if a function $\kappa(\mathbf{x}, \mathbf{x}')$ is a valid kernel?
(i.e., corresponds to an inner product in *some* feature space?)

Mercer's Condition: A function κ is a valid kernel if and only if:

Properties of Kernels: Mercer's Condition

How do we know if a function $\kappa(\mathbf{x}, \mathbf{x}')$ is a valid kernel?
(i.e., corresponds to an inner product in *some* feature space?)

Mercer's Condition: A function κ is a valid kernel if and only if:

1. It is **symmetric**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$$

Properties of Kernels: Mercer's Condition

How do we know if a function $\kappa(\mathbf{x}, \mathbf{x}')$ is a valid kernel?
(i.e., corresponds to an inner product in *some* feature space?)

Mercer's Condition: A function κ is a valid kernel if and only if:

1. It is **symmetric**:

$$\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$$

2. The kernel matrix $\mathbf{K}$ is **positive semi-definite** for *any* set of points $\{\mathbf{x}_n\}_{n=1}^N$.

$$\mathbf{K}_{ij} = \kappa(\mathbf{x}_i, \mathbf{x}_j) \Longrightarrow \mathbf{c}^\top \mathbf{K} \mathbf{c} \geq 0 \text{ for all } \mathbf{c} \in \mathbb{R}^N$$

Recap: Key Ideas

- Many algorithms (SVM, Ridge Regression, PCA) can be "kernelized" because their solutions can be written to depend only on inner products, $\mathbf{X}\mathbf{X}^\top$.

Recap: Key Ideas

- Many algorithms (SVM, Ridge Regression, PCA) can be "kernelized" because their solutions can be written to depend only on inner products, $\mathbf{X}\mathbf{X}^\top$.
- This allows us to generalize a linear model by replacing the simple inner product $\mathbf{x}^\top\mathbf{x}'$ with a non-linear kernel function $\kappa(\mathbf{x}, \mathbf{x}')$.

Recap: Key Ideas

- Many algorithms (SVM, Ridge Regression, PCA) can be "kernelized" because their solutions can be written to depend only on inner products, $\mathbf{X}\mathbf{X}^\top$.
- This allows us to generalize a linear model by replacing the simple inner product $\mathbf{x}^\top\mathbf{x}'$ with a non-linear kernel function $\kappa(\mathbf{x}, \mathbf{x}')$.
- This creates a non-linear model in the original input space, but one that is still linear (and convex) in a new, high-dimensional feature space.

Recap: Key Ideas

- Many algorithms (SVM, Ridge Regression, PCA) can be "kernelized" because their solutions can be written to depend only on inner products, $\mathbf{X}\mathbf{X}^\top$.
- This allows us to generalize a linear model by replacing the simple inner product $\mathbf{x}^\top\mathbf{x}'$ with a non-linear kernel function $\kappa(\mathbf{x}, \mathbf{x}')$.
- This creates a non-linear model in the original input space, but one that is still linear (and convex) in a new, high-dimensional feature space.
- The Kernel Trick is that we can do this *without ever computing* the feature maps $\phi(\mathbf{x})$, which can be very high-dim or even infinite-dim.

Recap: Key Ideas

- Many algorithms (SVM, Ridge Regression, PCA) can be "kernelized" because their solutions can be written to depend only on inner products, $\mathbf{X}\mathbf{X}^\top$.
- This allows us to generalize a linear model by replacing the simple inner product $\mathbf{x}^\top\mathbf{x}'$ with a non-linear kernel function $\kappa(\mathbf{x}, \mathbf{x}')$.
- This creates a non-linear model in the original input space, but one that is still linear (and convex) in a new, high-dimensional feature space.
- The Kernel Trick is that we can do this *without ever computing* the feature maps $\phi(\mathbf{x})$, which can be very high-dim or even infinite-dim.
- To make a new prediction, we just need to compute the kernel (similarity) between the new point $\mathbf{x}$ and all the training points $\mathbf{x}_n$.