SVM (Contd), Multiclass and One-Class SVM

Piyush Rai

Introduction to Machine Learning (CS771A)

September 4, 2018
Recap: Hyperplane-based Classification

- Basic idea: Learn to separate by a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$

\[ \hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b) \]

- Predict the label of a test input $\mathbf{x}_*$ as: $\hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b)$
Recap: Hyperplane-based Classification

- Basic idea: Learn to separate by a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$

![Hyperplane diagram](image)

- Predict the label of a test input $\mathbf{x}_*$ as: $\hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b)$

- The hyperplane may be “implied” by the model, or learned directly
Recap: Hyperplane-based Classification

- Basic idea: Learn to separate by a hyperplane $w^\top x + b = 0$

- Predict the label of a test input $x_*$ as: $\hat{y}_* = \text{sign}(w^\top x_* + b)$

- The hyperplane may be “implied” by the model, or learned directly
  - Implied: Prototype-based classification, nearest neighbors, generative classification, etc.
Recap: Hyperplane-based Classification

- Basic idea: Learn to separate by a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$

- Predict the label of a test input $\mathbf{x}_*$ as: $\hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b)$

- The hyperplane may be “implied” by the model, or learned directly
  - Implied: Prototype-based classification, nearest neighbors, generative classification, etc.
  - Directly learned: Logistic regression, Perceptron, Support Vector Machine, etc.
Basic idea: Learn to separate by a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$

Predict the label of a test input $\mathbf{x}_*$ as: $\hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b)$

The hyperplane may be “implied” by the model, or learned directly
- Implied: Prototype-based classification, nearest neighbors, generative classification, etc.
- Directly learned: Logistic regression, Perceptron, Support Vector Machine, etc.

The “direct” approach defines a model with parameters $\mathbf{w}$ (and optionally $b$) and learns them by minimizing a suitable loss function (and doesn’t model $\mathbf{x}$, i.e., purely discriminative)
Recap: Hyperplane-based Classification

- Basic idea: Learn to separate by a hyperplane $\mathbf{w}^\top \mathbf{x} + b = 0$

- Predict the label of a test input $\mathbf{x}_*$ as: $\hat{y}_* = \text{sign}(\mathbf{w}^\top \mathbf{x}_* + b)$

- The hyperplane may be “implied” by the model, or learned directly
  - Implied: Prototype-based classification, nearest neighbors, generative classification, etc.
  - Directly learned: Logistic regression, Perceptron, Support Vector Machine, etc.

- The “direct” approach defines a model with parameters $\mathbf{w}$ (and optionally $b$) and learns them by minimizing a suitable loss function (and doesn’t model $\mathbf{x}$, i.e., purely discriminative)

- The hyperplane need not be linear (e.g., can be made nonlinear using kernel methods - next class)
Recap: Hyperplanes and Margin

\[ \mathbf{w}^T \mathbf{x}_n + b = 0 \]
Recap: Hyperplanes and Margin

Class +1

$w^T x_n + b = 1$

$w^T x_n + b = -1$

$w^T x_n + b \geq 1$ for $y_n = +1$

$w^T x_n + b \leq -1$ for $y_n = -1$

Class -1
Recap: Hyperplanes and Margin

Class +1

\[ w^\top x_n + b = 1 \]

Class -1

\[ w^\top x_n + b = -1 \]

\[ w^\top x_n + b \geq 1 \text{ for } y_n = +1 \]
\[ w^\top x_n + b \leq -1 \text{ for } y_n = -1 \]

\[ y_n (w^\top x_n + b) \geq 1 \]
Recap: Hyperplanes and Margin

\[ w^T x_n + b = 1 \]

\[ w^T x_n + b = -1 \]

\[ w^T x_n + b \geq 1 \text{ for } y_n = +1 \]
\[ w^T x_n + b \leq -1 \text{ for } y_n = -1 \]

\[ y_n (w^T x_n + b) \geq 1 \]

\[ \gamma = \min_{1 \leq n \leq N} \frac{|w^T x_n + b|}{||w||} = \frac{1}{||w||} \]

(Margin on each side)
Recap: Maximum-Margin Hyperplane

\[
(\hat{w}, \hat{b}) = \arg \max_{w, b} \frac{2}{\|w\|}, \quad \text{s.t.} \quad y_n(w^T x_n + b) \geq 1
\]
Recap: Maximum-Margin Hyperplane

\[
(\hat{w}, \hat{b}) = \arg \max_{w, b} \frac{2}{\|w\|}, \quad \text{s.t.} \quad y_n(w^\top x_n + b) \geq 1
\]

Total margin \((2\gamma)\)

\[
(\hat{w}, \hat{b}) = \arg \min_{w, b} \frac{\|w\|^2}{2}, \quad \text{s.t.} \quad y_n(w^\top x_n + b) \geq 1
\]

(equivalent to)
Recap: Maximum-Margin Hyperplane

\[(\hat{w}, \hat{b}) = \arg \max_{w,b} \frac{2}{\|w\|} , \quad \text{s.t.} \quad y_n(w^\top x_n + b) \geq 1\]

(hard-margin SVM)

\[(\hat{w}, \hat{b}) = \arg \min_{w,b} \frac{\|w\|^2}{2} , \quad \text{s.t.} \quad y_n(w^\top x_n + b) \geq 1\]

Hard-margin SVM

("hard" = want all points to satisfy the margin constraint)
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n(w^T x_n + b) \geq 1$.
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n (w^T x_n + b) \geq 1$
- Let’s allow every point $x_n$ to “slack the constraint” by a distance $\xi_n \geq 0$
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n(w^T x_n + b) \geq 1$
- Let's allow every point $x_n$ to "slack the constraint" by a distance $\xi_n \geq 0$

New Objective: Maximize the margin while keeping the sum of slacks $\sum_{n=1}^{N} \xi_n$ small

Note: Can also think of the sum of slacks as the total training error
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n(w^\top x_n + b) \geq 1$
- Let’s allow every point $x_n$ to “slack the constraint” by a distance $\xi_n \geq 0$

\[ w^\top x_n + b \geq 1 - \xi_n \text{ for } y_n = +1 \]
\[ w^\top x_n + b \leq -1 + \xi_n \text{ for } y_n = -1 \]
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint \( y_n(w^T x_n + b) \geq 1 \)
- Let’s allow every point \( x_n \) to “slack the constraint” by a distance \( \xi_n \geq 0 \)

\[
\begin{align*}
\text{Class } +1 & : w^T x_n + b \geq 1 - \xi_n & \text{for } y_n = +1 \\
\text{Class } -1 & : w^T x_n + b \leq -1 + \xi_n & \text{for } y_n = -1 \\
y_n(w^T x_n + b) \geq 1 - \xi_n, & \quad \xi_n \geq 0
\end{align*}
\]
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n (w^T x_n + b) \geq 1$
- Let’s allow every point $x_n$ to “slack the constraint” by a distance $\xi_n \geq 0$

Points with $\xi_n \geq 0$ will be either in the margin region or totally on the wrong side
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n(w^\top x_n + b) \geq 1$
- Let’s allow every point $x_n$ to “slack the constraint” by a distance $\xi_n \geq 0$

![Diagram](image)

- Points with $\xi_n \geq 0$ will be either in the margin region or totally on the wrong side
- **New Objective:** Maximize the margin while keeping the sum of slacks $\sum_{n=1}^{N} \xi_n$ small

Note: Can also think of the sum of slacks as the total training error
Recap: Maximum-Margin Hyperplane with Slacks

- Still want a max-margin hyperplane but want to relax the hard constraint $y_n(w^\top x_n + b) \geq 1$
- Let’s allow every point $x_n$ to “slack the constraint” by a distance $\xi_n \geq 0$

Points with $\xi_n \geq 0$ will be either in the margin region or totally on the wrong side

New Objective: Maximize the margin while keeping the sum of slacks $\sum_{n=1}^{N} \xi_n$ small

Note: Can also think of the sum of slacks as the total training error
Recap: Maximum-Margin Hyperplane with Slacks

\[
(\hat{w}, \hat{b}, \xi) = \arg \min_{w, b} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n
\]

Maximize the margin

s.t. \( y_n (w^\top x_n + b) \geq 1 - \xi_n \) Slack-relaxed constraints

Minimize the sum of slacks (don't have too many violations)

Hyperparameter to balance the two

\( \xi_n \geq 0 \)

This formulation is known as the "soft-margin" SVM

Very small \( C \): Large margin but also large training error. :-(

Very large \( C \): Small training error but also small margin. :-(

\( C \) controls the trade-off between large margin and small training error
Recap: Maximum-Margin Hyperplane with Slacks

\[
(\hat{w}, \hat{b}, \xi) = \arg\min_{w, b} \frac{\|w\|^2}{2} + C \sum_{n=1}^{N} \xi_n
\]

subject to:

\[y_n(w^T x_n + b) \geq 1 - \xi_n\] (Slack-relaxed constraints)

\[\xi_n \geq 0\]

This formulation is known as the “soft-margin” SVM.
Recap: Maximum-Margin Hyperplane with Slacks

\[
(w, b, \xi) = \arg \min_{w,b} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n
\]

\[
s.t. \quad y_n(w^T x_n + b) \geq 1 - \xi_n \quad \xi_n \geq 0
\]

- This formulation is known as the “soft-margin” SVM
- Very small $C$: Large margin but also large training error. :-(
Recap: Maximum-Margin Hyperplane with Slacks

This formulation is known as the “soft-margin” SVM

- Very small $C$: Large margin but also large training error. :-(
- Very large $C$: Small training error but also small margin. :-(
Recap: Maximum-Margin Hyperplane with Slacks

This formulation is known as the “soft-margin” SVM

- Very small $C$: Large margin but also large training error. :-(
- Very large $C$: Small training error but also small margin. :-(
- $C$ controls the trade-off between large margin and small training error
Objective for the hard-margin SVM (unknowns are $w$ and $b$)

$$\arg\min_{w, b} \frac{||w||^2}{2}$$

subject to $y_n(w^T x_n + b) \geq 1$, $n = 1, \ldots, N$

Objective for the soft-margin SVM (unknowns are $w$, $b$, and $\{\xi_n\}_{n=1}^N$)

$$\arg\min_{w, b, \xi} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

subject to $y_n(w^T x_n + b) \geq 1 - \xi_n$, $\xi_n \geq 0$, $n = 1, \ldots, N$

In either case, we have to solve a constrained, convex optimization problem.
Solving SVM Objectives
Solving Hard-Margin SVM

- The hard-margin SVM optimization problem is:

\[
\begin{align*}
\text{arg min}_{w,b} & \frac{||w||^2}{2} \\
\text{subject to} & \quad 1 - y_n(w^T x_n + b) \leq 0, \quad n = 1, \ldots, N
\end{align*}
\]

- A constrained optimization problem. Can solve using Lagrange's method
The hard-margin SVM optimization problem is:

\[
\begin{align*}
\text{arg min} \quad & \frac{||w||^2}{2} \\
\text{subject to} \quad & 1 - y_n(w^T x_n + b) \leq 0, \quad n = 1, \ldots, N
\end{align*}
\]

A constrained optimization problem. Can solve using Lagrange’s method

Introduce Lagrange Multipliers \(\alpha_n\) \((n = \{1, \ldots, N\})\), one for each constraint, and solve

\[
\min_{w, b} \max_{\alpha \geq 0} \mathcal{L}(w, b, \alpha) = \frac{||w||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\}
\]

Note: \(\alpha = [\alpha_1, \ldots, \alpha_N]\) is the vector of Lagrange multipliers
Solving Hard-Margin SVM

- The hard-margin SVM optimization problem is:

\[
\begin{align*}
\text{arg min}_{w, b} & \quad \frac{||w||^2}{2} \\
\text{subject to} & \quad 1 - y_n(w^T x_n + b) \leq 0, \quad n = 1, \ldots, N
\end{align*}
\]

- A constrained optimization problem. Can solve using Lagrange’s method

- Introduce Lagrange Multipliers \(\alpha_n\) \((n = \{1, \ldots, N\})\), one for each constraint, and solve

\[
\begin{align*}
\min_{w, b} \quad & \max_{\alpha \geq 0} \quad \mathcal{L}(w, b, \alpha) = \frac{||w||^2}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\}
\end{align*}
\]

- Note: \(\alpha = [\alpha_1, \ldots, \alpha_N]\) is the vector of Lagrange multipliers

- Note: It is easier (and helpful; we will soon see why) to solve the dual problem: min and then max
Solving Hard-Margin SVM

- The dual problem (min then max) is

\[
\max_{\alpha \geq 0} \min_{w, b} \mathcal{L}(w, b, \alpha) = \frac{w^T w}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\}
\]
Solving Hard-Margin SVM

- The dual problem (min then max) is

$$\max_{\alpha \geq 0} \min_{w, b} L(w, b, \alpha) = \frac{w^T w}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n (w^T x_n + b)\}$$

- Take (partial) derivatives of $L$ w.r.t. $w, b$ and set them to zero

$$\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

$$\frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$
Solving Hard-Margin SVM

• The dual problem (min then max) is

$$\max_{\alpha \geq 0} \min_{w, b} \mathcal{L}(w, b, \alpha) = \frac{w^T w}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b)\}$$

• Take (partial) derivatives of $\mathcal{L}$ w.r.t. $w$, $b$ and set them to zero

$$\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n$$

$$\frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

• Important: Note the form of the solution $w$ - it is simply a weighted sum of all the training inputs $x_1, \ldots, x_N$ (and $\alpha_n$ is like the “importance” of $x_n$)
Solving Hard-Margin SVM

The dual problem (min then max) is

\[
\max_{\alpha \geq 0} \min_{w, b} \mathcal{L}(w, b, \alpha) = \frac{w^T w}{2} + \sum_{n=1}^{N} \alpha_n \{1 - y_n (w^T x_n + b)\}
\]

Take (partial) derivatives of \(\mathcal{L}\) w.r.t. \(w, b\) and set them to zero

\[
\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0
\]

Important: Note the form of the solution \(w\) - it is simply a weighted sum of all the training inputs \(x_1, \ldots, x_N\) (and \(\alpha_n\) is like the “importance” of \(x_n\))

Substituting \(w = \sum_{n=1}^{N} \alpha_n y_n x_n\) in Lagrangian, we get the dual problem as (verify)

\[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m, n=1}^{N} \alpha_m \alpha_n y_m y_n (x_m^T x_n)
\]
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as

\[
\max_{\alpha \geq 0} L_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_my_n x_m^\top x_n \), and \( 1 \) is a vector of 1s.
Can write the objective more compactly in vector/matrix form as

\[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^\top x_n \), and \( 1 \) is a vector of 1s

**Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is \( G \), which is p.s.d.). Note that our original SVM objective was also convex

\[\text{\dag If interested in more details of the solver, see: "Support Vector Machine Solvers" by Bottou and Lin}\]
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as

$$\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha$$

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n x_m^\top x_n$, and $1$ is a vector of 1s.

- **Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is $G$, which is p.s.d.). Note that our original SVM objective was also convex.

- **Important:** Inputs $x$’s only appear as inner products (helps to “kernelize”; more on this later).

\[†\] If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as

\[
\max_{\alpha \geq 0} L_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^\top x_n \), and \( 1 \) is a vector of 1s.

- **Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is \( G \), which is p.s.d.). Note that our original SVM objective was also convex.

- **Important:** Inputs \( x \)'s only appear as inner products (helps to “kernelize”; more on this later).

- Can solve† the above objective function for \( \alpha \) using various methods, e.g.,

If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin.
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as

\[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n \mathbf{x}_m^\top \mathbf{x}_n \), and \( \mathbf{1} \) is a vector of 1s

- **Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is \( G \), which is p.s.d.). Note that our original SVM objective was also convex

- **Important:** Inputs \( \mathbf{x} \)'s only appear as inner products (helps to "kernelize"; more on this later)

- Can solve† the above objective function for \( \alpha \) using various methods, e.g.,
  - Treating the objective as a **Quadratic Program** (QP) and running some off-the-shelf QP solver such as quadprog (MATLAB), CVXOPT, CPLEX, etc.

---

† If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as

\[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^\top x_n \), and \( 1 \) is a vector of 1s

- **Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is \( G \), which is p.s.d.). Note that our original SVM objective was also convex

- **Important:** Inputs \( x \)'s only appear as inner products (helps to “kernelize”; more on this later)

- Can solve\(^\dagger\) the above objective function for \( \alpha \) using various methods, e.g.,
  - Treating the objective as a Quadratic Program (QP) and running some off-the-shelf QP solver such as quadprog (MATLAB), CVXOPT, CPLEX, etc.
  - Using (projected) gradient methods (projection needed because the \( \alpha \)'s are constrained). Gradient methods will usually be much faster than QP methods.

\(^\dagger\)If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Hard-Margin SVM

- Can write the objective more compactly in vector/matrix form as
  \[
  \max_{\alpha \geq 0} \ L_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
  \]

  where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^\top x_n \), and \( 1 \) is a vector of 1s

- **Good news:** This is maximizing a concave function (or minimizing a convex function - verify that the Hessian is \( G \), which is p.s.d.). Note that our original SVM objective was also convex

- **Important:** Inputs \( x \)'s only appear as inner products (helps to “kernelize”; more on this later)

- Can solve\(^\dagger\) the above objective function for \( \alpha \) using various methods, e.g.,
  - Treating the objective as a **Quadratic Program** (QP) and running some off-the-shelf QP solver such as quadprog (MATLAB), CVXOPT, CPLEX, etc.
  - Using (projected) **gradient methods** (projection needed because the \( \alpha \)'s are constrained). Gradient methods will usually be much faster than QP methods.
  - Using **co-ordinate ascent** methods (optimize for one \( \alpha_n \) at a time); often very fast

\(^\dagger\) If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Hard-Margin SVM: The Solution

Once we have the $\alpha_n$’s, $\mathbf{w}$ and $b$ can be computed as:

\[
\mathbf{w} = \sum_{n=1}^{N} \alpha_n y_n \mathbf{x}_n \quad \text{(we already saw this)}
\]

\[
b = -\frac{1}{2} \left( \min_{n:y_n=+1} \mathbf{w}^T \mathbf{x}_n + \max_{n:y_n=-1} \mathbf{w}^T \mathbf{x}_n \right) \quad \text{(exercise)}
\]
Once we have the $\alpha_n$’s, $w$ and $b$ can be computed as:

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n$$  \text{(we already saw this)}

$$b = -\frac{1}{2} \left( \min_{n:y_n=+1} w^T x_n + \max_{n:y_n=-1} w^T x_n \right)$$ \text{(exercise)}

A nice property: Most $\alpha_n$’s in the solution will be zero (sparse solution)

Reason: Karush-Kuhn-Tucker (KKT) conditions
Hard-Margin SVM: The Solution

- Once we have the $\alpha_n$’s, $w$ and $b$ can be computed as:

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n$$  \hspace{1cm} \text{(we already saw this)}

$$b = -\frac{1}{2} \left( \min_{n:y_n=+1} w^T x_n + \max_{n:y_n=-1} w^T x_n \right)$$  \hspace{1cm} \text{(exercise)}

- A nice property: Most $\alpha_n$’s in the solution will be zero (sparse solution)
  - Reason: Karush-Kuhn-Tucker (KKT) conditions
  - For the optimal $\alpha_n$’s

$$\alpha_n \{ 1 - y_n (w^T x_n + b) \} = 0$$
Hard-Margin SVM: The Solution

Once we have the $\alpha_n$’s, $w$ and $b$ can be computed as:

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n \quad \text{(we already saw this)}$$

$$b = -\frac{1}{2} \left( \min_{n:y_n=+1} w^T x_n + \max_{n:y_n=-1} w^T x_n \right) \quad \text{(exercise)}$$

**A nice property:** Most $\alpha_n$’s in the solution will be zero (sparse solution)

- Reason: Karush-Kuhn-Tucker (KKT) conditions
- For the optimal $\alpha_n$’s
  $$\alpha_n \{1 - y_n (w^T x_n + b)\} = 0$$
- $\alpha_n$ is non-zero only if $x_n$
Hard-Margin SVM: The Solution

- Once we have the $\alpha_n$'s, $w$ and $b$ can be computed as:

  \[ w = \sum_{n=1}^{N} \alpha_n y_n x_n \]  
  \[ b = -\frac{1}{2} \left( \min_{n:y_n=+1} w^T x_n + \max_{n:y_n=-1} w^T x_n \right) \]  
  (we already saw this)  
  (exercise)

- **A nice property:** Most $\alpha_n$'s in the solution will be zero (sparse solution)

  - Reason: Karush-Kuhn-Tucker (KKT) conditions
  - For the optimal $\alpha_n$'s

    \[ \alpha_n \{1 - y_n (w^T x_n + b)\} = 0 \]

  - $\alpha_n$ is non-zero only if $x_n$ lies on one of the two margin boundaries, i.e., for which $y_n (w^T x_n + b) = 1$
Once we have the $\alpha_n$'s, $w$ and $b$ can be computed as:

\[ w = \sum_{n=1}^{N} \alpha_n y_n x_n \] (we already saw this)

\[ b = -\frac{1}{2} \left( \min_{n: y_n = +1} w^T x_n + \max_{n: y_n = -1} w^T x_n \right) \] (exercise)

**A nice property:** Most $\alpha_n$'s in the solution will be zero (sparse solution)

- Reason: Karush-Kuhn-Tucker (KKT) conditions
- For the optimal $\alpha_n$'s
  \[ \alpha_n \{1 - y_n (w^T x_n + b)\} = 0 \]

- $\alpha_n$ is non-zero only if $x_n$ lies on one of the two margin boundaries, i.e., for which $y_n (w^T x_n + b) = 1$
- These examples are called support vectors
Hard-Margin SVM: The Solution

- Once we have the $\alpha_n$’s, $w$ and $b$ can be computed as:

$$w = \sum_{n=1}^{N} \alpha_n y_n x_n \quad \text{(we already saw this)}$$

$$b = -\frac{1}{2} \left( \min_{n:y_n=+1} w^T x_n + \max_{n:y_n=-1} w^T x_n \right) \quad \text{(exercise)}$$

- **A nice property:** Most $\alpha_n$’s in the solution will be zero (sparse solution)

  - Reason: Karush-Kuhn-Tucker (KKT) conditions
  - For the optimal $\alpha_n$’s

$$\alpha_n \{1 - y_n (w^T x_n + b)\} = 0$$

  - $\alpha_n$ is non-zero only if $x_n$ lies on one of the two margin boundaries, i.e., for which $y_n (w^T x_n + b) = 1$
  - These examples are called support vectors
  - Recall the support vectors “support” the margin boundaries
Solving Soft-Margin SVM
Solving Soft-Margin SVM

- Recall the soft-margin SVM optimization problem:

\[
\min_{w, b, \xi} f(w, b, \xi) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to } 1 \leq y_n(w^T x_n + b) + \xi_n, \quad -\xi_n \leq 0 \quad n = 1, \ldots, N
\]

- Note: \( \xi = [\xi_1, \ldots, \xi_N] \) is the vector of slack variables
Recall the soft-margin SVM optimization problem:

\[
\min_{w, b, \xi} \quad f(w, b, \xi) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to} \quad 1 \leq y_n(w^T x_n + b) + \xi_n, \quad -\xi_n \leq 0 \quad n = 1, \ldots, N
\]

Note: \( \xi = [\xi_1, \ldots, \xi_N] \) is the vector of slack variables

Introduce Lagrange Multipliers \( \alpha_n, \beta_n \ (n = \{1, \ldots, N\}) \), for constraints, and solve the Lagrangian:

\[
\min_{w, b, \xi} \quad \max_{\alpha \geq 0, \beta \geq 0} \quad \mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]
Solving Soft-Margin SVM

• Recall the soft-margin SVM optimization problem:

\[
\begin{align*}
\min_{w, b, \xi} & \quad f(w, b, \xi) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to} & \quad 1 \leq y_n(w^T x_n + b) + \xi_n, \quad -\xi_n \leq 0 \quad n = 1, \ldots, N
\end{align*}
\]

Note: \( \xi = [\xi_1, \ldots, \xi_N] \) is the vector of slack variables

• Introduce Lagrange Multipliers \( \alpha_n, \beta_n \ (n = \{1, \ldots, N\}) \), for constraints, and solve the Lagrangian:

\[
\begin{align*}
\min_{w, b, \xi} & \quad \max_{\alpha \geq 0, \beta \geq 0} L(w, b, \xi, \alpha, \beta) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\end{align*}
\]

Note: The terms in red above were not present in the hard-margin SVM
Recall the soft-margin SVM optimization problem:

\[
\min_{w, b, \xi} f(w, b, \xi) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to } 1 \leq y_n(w^T x_n + b) + \xi_n, \quad -\xi_n \leq 0 \quad n = 1, \ldots, N
\]

Note: \( \xi = [\xi_1, \ldots, \xi_N] \) is the vector of slack variables

Introduce Lagrange Multipliers \( \alpha_n, \beta_n \ (n = \{1, \ldots, N\}) \), for constraints, and solve the Lagrangian:

\[
\min_{w, b, \xi} \max_{\alpha \geq 0, \beta \geq 0} L(w, b, \xi, \alpha, \beta) = \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]

Note: The terms in red above were not present in the hard-margin SVM

Two sets of dual variables \( \alpha = [\alpha_1, \ldots, \alpha_N] \) and \( \beta = [\beta_1, \ldots, \beta_N] \). We'll eliminate the primal variables \( w, b, \xi \) to get dual problem containing the dual variables (just like in the hard margin case)
Solving Soft-Margin SVM

- The Lagrangian problem to solve

\[
\min_{w, b, \xi} \max_{\alpha \geq 0, \beta \geq 0} L(w, b, \xi, \alpha, \beta) = \frac{w^T w}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n (w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]
Solving Soft-Margin SVM

The Lagrangian problem to solve

\[
\min_{w, b, \xi} \quad \max_{\alpha \geq 0, \beta \geq 0} \quad L(w, b, \xi, \alpha, \beta) = \frac{w^T w}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]

Take (partial) derivatives of \(L\) w.r.t. \(w, b, \xi_n\) and set them to zero

\[
\begin{align*}
\frac{\partial L}{\partial w} = 0 & \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n, \\
\frac{\partial L}{\partial b} = 0 & \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \\
\frac{\partial L}{\partial \xi_n} = 0 & \Rightarrow C - \alpha_n - \beta_n = 0
\end{align*}
\]

Note: Solution of \(w\) again has the same form as in the hard-margin case (weighted sum of all inputs with \(\alpha_n\) being the importance of input \(x_n\))

Note: Using \(C - \alpha_n - \beta_n = 0\) and \(\beta_n \geq 0\) \(\Rightarrow \alpha_n \leq C\) (recall that, for the hard-margin case, \(\alpha_n \geq 0\))

Substituting these in the Lagrangian \(L\) gives the Dual problem

\[
\max_{\alpha \leq C, \beta \geq 0} \quad L_D(\alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (x_m^T x_n)
\]
Solving Soft-Margin SVM

- The Lagrangian problem to solve

\[
\min_{w, b, \xi} \max_{\alpha \geq 0, \beta \geq 0} L(w, b, \xi, \alpha, \beta) = \frac{w^T w}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]

- Take (partial) derivatives of \( L \) w.r.t. \( w \), \( b \), \( \xi_n \) and set them to zero

\[
\frac{\partial L}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n, \quad \frac{\partial L}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial L}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0
\]

- Note: Solution of \( w \) again has the same form as in the hard-margin case (weighted sum of all inputs with \( \alpha_n \) being the importance of input \( x_n \))
Solving Soft-Margin SVM

- The Lagrangian problem to solve

\[
\min_{w, b, \xi} \max_{\alpha \geq 0, \beta \geq 0} \mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{w^T w}{2} + C\sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]

- Take (partial) derivatives of \(\mathcal{L}\) w.r.t. \(w, b, \xi_n\) and set them to zero

\[
\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n, \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0
\]

- Note: Solution of \(w\) again has the same form as in the hard-margin case (weighted sum of all inputs with \(\alpha_n\) being the importance of input \(x_n\))

- Note: Using \(C - \alpha_n - \beta_n = 0\) and \(\beta_n \geq 0 \Rightarrow \alpha_n \leq C\) (recall that, for the hard-margin case, \(\alpha \geq 0\))
Solving Soft-Margin SVM

• The Lagrangian problem to solve

\[
\min_{w, b, \xi, \alpha \geq 0, \beta \geq 0} \max \quad \mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{w^T w}{2} + C \sum_{n=1}^{N} \xi_n + \sum_{n=1}^{N} \alpha_n \{1 - y_n(w^T x_n + b) - \xi_n\} - \sum_{n=1}^{N} \beta_n \xi_n
\]

• Take (partial) derivatives of \( \mathcal{L} \) w.r.t. \( w, b, \xi_n \) and set them to zero

\[
\frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{n=1}^{N} \alpha_n y_n x_n, \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0
\]

• Note: Solution of \( w \) again has the same form as in the hard-margin case (weighted sum of all inputs with \( \alpha_n \) being the importance of input \( x_n \))

• Note: Using \( C - \alpha_n - \beta_n = 0 \) and \( \beta_n \geq 0 \) \( \Rightarrow \alpha_n \leq C \) (recall that, for the hard-margin case, \( \alpha \geq 0 \))

• Substituting these in the Lagrangian \( \mathcal{L} \) gives the Dual problem

\[
\max_{\alpha \leq C, \beta \geq 0} \quad \mathcal{L}_D(\alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m, n=1}^{N} \alpha_m \alpha_n y_m y_n (x_m^T x_n)
\]
Solving Soft-Margin SVM

- Interestingly, the dual variables $\beta$ don’t appear in the objective!

\[
\max_{\alpha \leq C} \langle \alpha \rangle = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n x_m^\top x_n$, and $1$ is a vector of 1s.

Like hard-margin case, solving the dual requires concave maximization (or convex minimization) and can be solved the same way as hard-margin SVM (except that $\alpha \leq C$). Can solve for $\alpha$ using quadratic programming solvers or (projected) gradient methods.

Given $\alpha$, the solution for $w, b$ has the same form as hard-margin case.

Note: $\alpha$ is again sparse. Nonzero $\alpha_n$'s correspond to the support vectors.

† If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Soft-Margin SVM

- Interestingly, the dual variables $\beta$ don’t appear in the objective!
- Just like the hard-margin case, we can write the dual more compactly as

$$
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
$$

where $G$ is an $N \times N$ matrix with $G_{mn} = y_my_nx_m^\top x_n$, and $1$ is a vector of 1s

† If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Soft-Margin SVM

- Interestingly, the dual variables $\beta$ don’t appear in the objective!
- Just like the hard-margin case, we can write the dual more compactly as

\[
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n \mathbf{x}_m^\top \mathbf{x}_n$, and $1$ is a vector of 1s.
- Like hard-margin case, solving the dual requires concave maximization (or convex minimization)

† If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Interestingly, the dual variables $\beta$ don’t appear in the objective!

Just like the hard-margin case, we can write the dual more compactly as

$$
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
$$

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n x_m^\top x_n$, and $1$ is a vector of 1s.

Like hard-margin case, solving the dual requires concave maximization (or convex minimization).

Can be solved\(^\dagger\) the same way as hard-margin SVM (except that $\alpha \leq C$)

- Can solve for $\alpha$ using QP solvers or (projected) gradient methods

\(^\dagger\) If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Soft-Margin SVM

- Interestingly, the dual variables $\beta$ don’t appear in the objective!
- Just like the hard-margin case, we can write the dual more compactly as

$$
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top \mathbf{G} \alpha
$$

where $\mathbf{G}$ is an $N \times N$ matrix with $G_{mn} = y_my_n x_m^\top x_n$, and $\mathbf{1}$ is a vector of 1s.

- Like hard-margin case, solving the dual requires concave maximization (or convex minimization)
- Can be solved† the same way as hard-margin SVM (except that $\alpha \leq C$)
  - Can solve for $\alpha$ using QP solvers or (projected) gradient methods
- Given $\alpha$, the solution for $w, b$ has the same form as hard-margin case

† If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin
Solving Soft-Margin SVM

- Interestingly, the dual variables $\beta$ don’t appear in the objective!
- Just like the hard-margin case, we can write the dual more compactly as

$$\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha$$

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n x_m^\top x_n$, and $1$ is a vector of 1s
- Like hard-margin case, solving the dual requires concave maximization (or convex minimization)
- Can be solved† the same way as hard-margin SVM (except that $\alpha \leq C$)
  - Can solve for $\alpha$ using QP solvers or (projected) gradient methods
- Given $\alpha$, the solution for $w, b$ has the same form as hard-margin case
- **Note:** $\alpha$ is again sparse. Nonzero $\alpha_n$’s correspond to the support vectors

---

*†If interested in more details of the solver, see: “Support Vector Machine Solvers” by Bottou and Lin*
Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - ones that lie on the margin boundaries $w^T x + b = -1$ and $w^T x + b = +1$

- The soft-margin SVM solution has three types of support vectors
  1. Lying on the margin boundaries $w^T x + b = -1$ and $w^T x + b = +1$ ($\xi_n = 0$)
  2. Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side
  3. Lying on the wrong side of the hyperplane ($\xi_n \geq 1$)
Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - ones that lie on the margin boundaries \( w^T x + b = -1 \) and \( w^T x + b = +1 \)

- The soft-margin SVM solution has three types of support vectors
  1. Lying on the margin boundaries \( w^T x + b = -1 \) and \( w^T x + b = +1 \) \( (\xi_n = 0) \)
  2. Lying within the margin region \( (0 < \xi_n < 1) \) but still on the correct side
  3. Lying on the wrong side of the hyperplane \( (\xi_n \geq 1) \)
Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - ones that lie on the margin boundaries \( w^T x + b = -1 \) and \( w^T x + b = +1 \)
- The soft-margin SVM solution has three types of support vectors
  1. Lying on the margin boundaries \( w^T x + b = -1 \) and \( w^T x + b = +1 \) (\( \xi_n = 0 \))
  2. Lying within the margin region (\( 0 < \xi_n < 1 \)) but still on the correct side
  3. Lying on the wrong side of the hyperplane (\( \xi_n \geq 1 \))
Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - .. ones that lie on the margin boundaries $w^T x + b = -1$ and $w^T x + b = +1$
- The soft-margin SVM solution has three types of support vectors
  1. Lying on the margin boundaries $w^T x + b = -1$ and $w^T x + b = +1$ ($\xi_n = 0$)
  2. Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side
Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - ones that lie on the margin boundaries $\mathbf{w}^T \mathbf{x} + b = -1$ and $\mathbf{w}^T \mathbf{x} + b = +1$

- The soft-margin SVM solution has three types of support vectors
  1. Lying on the margin boundaries $\mathbf{w}^T \mathbf{x} + b = -1$ and $\mathbf{w}^T \mathbf{x} + b = +1$ ($\xi_n = 0$)
  2. Lying within the margin region ($0 < \xi_n < 1$) but still on the correct side
  3. Lying on the wrong side of the hyperplane ($\xi_n \geq 1$)
Recall the final dual objectives for hard-margin and soft-margin SVM

**Hard-Margin SVM:**
\[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

**Soft-Margin SVM:**
\[
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

The dual formulation is nice due to two primary reasons:
- Allows conveniently handling the margin based constraint (via Lagrangians)
- Important: Allows learning nonlinear separators by replacing inner products (e.g., \(G_{mn} = y_m y_n x_m^\top x_n\))
  by kernelized similarities (kernelized SVMs)

However, the dual formulation can be expensive if \(N\) is large. Have to solve for \(N\) variables \(\alpha = [\alpha_1, \ldots, \alpha_N]\), and also need to store an \(N \times N\) matrix \(G\).

† See: “Support Vector Machine Solvers” by Bottou and Lin
SVMs via Dual Formulation: Some Comments

- Recall the final dual objectives for hard-margin and soft-margin SVM

\[
\text{Hard-Margin SVM: } \max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G\alpha
\]

\[
\text{Soft-Margin SVM: } \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G\alpha
\]

- The dual formulation is nice due to two primary reasons:

† See: “Support Vector Machine Solvers” by Bottou and Lin
Recall the final dual objectives for hard-margin and soft-margin SVM

Hard-Margin SVM: \[ \max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha \]

Soft-Margin SVM: \[ \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha \]

The dual formulation is nice due to two primary reasons:

- Allows conveniently handling the margin based constraint (via Lagrangians)

†See: “Support Vector Machine Solvers” by Bottou and Lin
Recall the final dual objectives for hard-margin and soft-margin SVM

\[
\text{Hard-Margin SVM: } \max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

\[
\text{Soft-Margin SVM: } \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

The dual formulation is nice due to two primary reasons:

- Allows conveniently handling the margin based constraint (via Lagrangians)
- **Important**: Allows learning nonlinear separators by replacing inner products (e.g., \( G_{mn} = y_m y_n x_m^\top x_n \)) by kernelized similarities (kernelized SVMs)

† See: “Support Vector Machine Solvers” by Bottou and Lin
SVMs via Dual Formulation: Some Comments

- Recall the final dual objectives for hard-margin and soft-margin SVM

\[
\text{Hard-Margin SVM: } \max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

\[
\text{Soft-Margin SVM: } \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

- The dual formulation is nice due to two primary reasons:
  - Allows conveniently handling the margin based constraint (via Lagrangians)
  - **Important**: Allows learning nonlinear separators by replacing inner products (e.g., \( G_{mn} = y_m y_n x_m^\top x_n \)) by kernelized similarities (kernelized SVMs)

- However, the dual formulation can be expensive if \( N \) is large. Have to solve for \( N \) variables \( \alpha = [\alpha_1, \ldots, \alpha_N] \), and also need to store an \( N \times N \) matrix \( G \)

† See: “Support Vector Machine Solvers” by Bottou and Lin
Recall the final dual objectives for hard-margin and soft-margin SVM

Hard-Margin SVM: \[
\max_{\alpha \geq 0} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

Soft-Margin SVM: \[
\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha
\]

The dual formulation is nice due to two primary reasons:

- Allows conveniently handling the margin based constraint (via Lagrangians)
- **Important**: Allows learning nonlinear separators by replacing inner products (e.g., \(G_{mn} = y_m y_n x_m^\top x_n\)) by kernelized similarities (kernelized SVMs)

However, the dual formulation can be expensive if \(N\) is large. Have to solve for \(N\) variables \(\alpha = [\alpha_1, \ldots, \alpha_N]\), and also need to store an \(N \times N\) matrix \(G\)

A lot of work† on speeding up SVM in these settings (e.g., can use co-ord. descent for \(\alpha\))

---

†See: “Support Vector Machine Solvers” by Bottou and Lin

Intro to Machine Learning (CS771A)

SVM (Contd), Multiclass and One-Class SVM

18
SVM: The Regularized Loss Function View

Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\arg \min_{w, b, \xi} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n
\]

subject to \( y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N \)
Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\begin{align*}
\text{arg min}_{w,b,\xi} & \quad \frac{1}{2} ||w||^2 + C \sum_{n=1}^{N} \xi_n \\
\text{subject to} & \quad y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0, \quad n = 1, \ldots, N
\end{align*}
\]

Note that the slack \( \xi_n \) is the same as \( \max\{0, 1 - y_n(w^T x_n + b)\} \), i.e., hinge loss for \((x_n, y_n)\)
SVM: The Regularized Loss Function View

- Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\arg \min_{w, b, \xi} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to } y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N
\]

- Note that the slack \(\xi_n\) is the same as \(\max\{0, 1 - y_n(w^T x_n + b)\}\), i.e., hinge loss for \((x_n, y_n)\)

- Another View: Thus the above is equivalent to minimizing the \(\ell_2\) regularized hinge loss

\[
L(w, b) = \sum_{n=1}^{N} \max\{0, 1 - y_n(w^T x_n + b)\} + \frac{\lambda}{2} w^T w
\]
SVM: The Regularized Loss Function View

- Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\arg \min_{w, b, \xi} \frac{||w||^2}{2} + C \sum_{n=1}^{N} \xi_n \\
\text{subject to } y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N
\]

- Note that the slack \(\xi_n\) is the same as \(\max\{0, 1 - y_n(w^T x_n + b)\}\), i.e., hinge loss for \((x_n, y_n)\)

- **Another View:** Thus the above is equivalent to minimizing the \(\ell_2\) regularized hinge loss

\[
\mathcal{L}(w, b) = \sum_{n=1}^{N} \max\{0, 1 - y_n(w^T x_n + b)\} + \frac{\lambda}{2} w^T w
\]

- **Comparing the two:** Sum of slacks is like sum of hinge losses, \(C\) and \(\lambda\) play similar roles
SVM: The Regularized Loss Function View

- Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\arg \min_{w,b,\xi} \frac{1}{2} \|w\|^2 + C \sum_{n=1}^{N} \xi_n \\
\text{subject to } y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N
\]

- Note that the slack \(\xi_n\) is the same as \(\max\{0, 1 - y_n(w^T x_n + b)\}\), i.e., hinge loss for \((x_n, y_n)\)

- **Another View:** Thus the above is equivalent to minimizing the \(\ell_2\) regularized hinge loss

\[
L(w, b) = \sum_{n=1}^{N} \max\{0, 1 - y_n(w^T x_n + b)\} + \frac{\lambda}{2} w^T w
\]

- **Comparing the two:** Sum of slacks is like sum of hinge losses, \(C\) and \(\lambda\) play similar roles
- Can learn \((w, b)\) directly by minimizing \(L(w, b)\) using (stochastic)(sub)gradient descent
Maximize the margin subject to constraints led to the soft-margin formulation of SVM

\[
\arg\min_{w,b,\xi} \frac{\|w\|^2}{2} + C \sum_{n=1}^{N} \xi_n
\]
subject to \( y_n(w^T x_n + b) \geq 1 - \xi_n, \quad \xi_n \geq 0 \quad n = 1, \ldots, N \)

Note that the slack \( \xi_n \) is the same as \( \max\{0, 1 - y_n(w^T x_n + b)\} \), i.e., hinge loss for \((x_n, y_n)\)

Another View: Thus the above is equivalent to minimizing the \( \ell_2 \) regularized hinge loss

\[
\mathcal{L}(w, b) = \sum_{n=1}^{N} \max\{0, 1 - y_n(w^T x_n + b)\} + \frac{\lambda}{2} w^T w
\]

Comparing the two: Sum of slacks is like sum of hinge losses, \( C \) and \( \lambda \) play similar roles

Can learn \((w, b)\) directly by minimizing \(\mathcal{L}(w, b)\) using (stochastic)(sub)gradient descent

- Hinge-loss version preferred for linear SVMs, or with other regularizers on \( w \) (e.g., \( \ell_1 \))
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $W = [w_1, w_2, \ldots, w_K]$ (similar to softmax regression)
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ (similar to softmax regression)
  \[ \hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n \quad \text{(prediction rule)} \]

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)
  \[
  \hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{k=1}^{K} ||\mathbf{w}_k||_2^2 \quad \text{s.t.} \quad \mathbf{w}_k^\top \mathbf{y}_n \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 - \xi_n \quad \forall k \neq y_n
  \]
  \[
  \hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{k=1}^{K} ||\mathbf{w}_k||_2^2 + C \sum_{n=1}^N \xi_n \quad \text{s.t.} \quad \mathbf{w}_k^\top \mathbf{y}_n \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 - \xi_n \quad \forall k \neq y_n
  \]

Want score w.r.t. correct class to be at least 1 more than score w.r.t. all other classes

The version with slack corresponds to minimizing a multi-class hinge loss

\[ L(\mathbf{W}) = \max \{0, 1 + \max_{k \neq y_n} \mathbf{w}_k^\top \mathbf{x}_n - \mathbf{w}_{y_n}^\top \mathbf{x}_n\} \quad \text{(Crammer-Singer multiclass SVM)} \]

Loss = 0 if score on correct class is at least 1 more than score on next best scoring class

Can optimize these similar to how we did it for binary SVM
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ (similar to softmax regression)

\[ \hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n \quad \text{(prediction rule)} \]

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

\[ \hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{k=1}^K \frac{||\mathbf{w}_k||^2}{2} \]

s.t. \[ \mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 \quad \forall k \neq y_n \]
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ (similar to softmax regression)

$$\hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n \quad \text{(prediction rule)}$$

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

$$\hat{W} = \arg \min_{\mathbf{W}} \sum_{k=1}^K \frac{||\mathbf{w}_k||^2}{2}$$

$$\hat{W} = \arg \min_{\mathbf{W}} \sum_{k=1}^K \frac{||\mathbf{w}_k||^2}{2} + C \sum_{n=1}^N \xi_n$$

s.t. $w_{y_n}^\top \mathbf{x}_n \geq w_k^\top \mathbf{x}_n + 1 \quad \forall k \neq y_n$

s.t. $w_{y_n}^\top \mathbf{x}_n \geq w_k^\top \mathbf{x}_n + 1 - \xi_n \quad \forall k \neq y_n$
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ (similar to softmax regression)

$$\hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n \quad \text{(prediction rule)}$$

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

$$\hat{\mathbf{W}} = \arg \min_{\mathbf{W}} K \sum_{k=1}^{K} \frac{\lVert \mathbf{w}_k \rVert^2}{2} \quad \hat{\mathbf{W}} = \arg \min_{\mathbf{W}} K \sum_{k=1}^{K} \frac{\lVert \mathbf{w}_k \rVert^2}{2} + C \sum_{n=1}^{N} \xi_n$$

  s.t. $\mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 \quad \forall k \neq y_n$

  s.t. $\mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 - \xi_n \quad \forall k \neq y_n$

- Want score w.r.t. correct class to be at least 1 more than score w.r.t. all other classes

The version with slack corresponds to minimizing a multi-class hinge loss

$$L(\mathbf{W}) = \max\left\{0, 1 + \max_{k \neq y_n} \mathbf{w}_k^\top \mathbf{x}_n - \mathbf{w}_{y_n}^\top \mathbf{x}_n\right\}$$

(Crammer-Singer multiclass SVM)

Loss = 0 if score on correct class is at least 1 more than score on next best scoring class

Can optimize these similar to how we did it for binary SVM
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K]$ (similar to softmax regression)

$$\hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n$$  \hspace{1cm} \text{(prediction rule)}

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

$$\hat{\mathbf{W}} = \arg \min_\mathbf{W} \sum_{k=1}^K \frac{||\mathbf{w}_k||^2}{2} \hspace{1cm} \hat{\mathbf{W}} = \arg \min_\mathbf{W} \sum_{k=1}^K \frac{||\mathbf{w}_k||^2}{2} + C \sum_{n=1}^N \xi_n$$

\hspace{1cm} \text{s.t.} \hspace{1cm} \mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 \hspace{1cm} \forall k \neq y_n \hspace{1cm} \text{s.t.} \hspace{1cm} \mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 - \xi_n \hspace{1cm} \forall k \neq y_n

- Want score w.r.t. correct class to be at least 1 more than score w.r.t. all other classes

- The version with slack corresponds to minimizing a multi-class hinge loss

$$\mathcal{L}(\mathbf{W}) = \max\{0, 1 + \max_{k \neq y_n} \mathbf{w}_k^\top \mathbf{x}_n - \mathbf{w}_{y_n}^\top \mathbf{x}_n\}$$  \hspace{1cm} \text{(Crammer-Singer multiclass SVM)}
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $\mathbf{W} = [ \mathbf{w}_1, \mathbf{w}_2, \ldots, \mathbf{w}_K ]$ (similar to softmax regression)

$$\hat{y}_* = \arg \max_k \mathbf{w}_k^\top \mathbf{x}_n \quad \text{(prediction rule)}$$

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

$$\hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{k=1}^{K} \frac{||\mathbf{w}_k||^2}{2} \quad \hat{\mathbf{W}} = \arg \min_{\mathbf{W}} \sum_{k=1}^{K} \frac{||\mathbf{w}_k||^2}{2} + C \sum_{n=1}^{N} \xi_n$$

s.t. $\mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 \quad \forall k \neq y_n$

s.t. $\mathbf{w}_{y_n}^\top \mathbf{x}_n \geq \mathbf{w}_k^\top \mathbf{x}_n + 1 - \xi_n \quad \forall k \neq y_n$

- Want score w.r.t. correct class to be at least 1 more than score w.r.t. all other classes

- The version with slack corresponds to minimizing a multi-class hinge loss

$$\mathcal{L}(\mathbf{W}) = \max\{0, 1 + \max_{k \neq y_n} \mathbf{w}_k^\top \mathbf{x}_n - \mathbf{w}_{y_n}^\top \mathbf{x}_n\} \quad \text{(Crammer-Singer multiclass SVM)}$$

- Loss = 0 if score on correct class is at least 1 more than score on next best scoring class
Multiclass SVM

- Multiclass SVMs use $K$ weight vectors $W = [w_1, w_2, \ldots, w_K]$ (similar to softmax regression)

$$\hat{y}_* = \arg \max_k w_k^T x_n \quad \text{(prediction rule)}$$

- Just like binary case, we can formulate a maximum-margin problem (without or with slacks)

$$\hat{W} = \arg \min_W \sum_{k=1}^K \frac{||w_k||^2}{2} \quad \hat{W} = \arg \min_W \sum_{k=1}^K \frac{||w_k||^2}{2} + C \sum_{n=1}^N \xi_n$$

s.t. $w_{y_n}^T x_n \geq w_k^T x_n + 1 \quad \forall k \neq y_n$

s.t. $w_{y_n}^T x_n \geq w_k^T x_n + 1 - \xi_n \quad \forall k \neq y_n$

- Want score w.r.t. correct class to be at least 1 more than score w.r.t. all other classes

- The version with slack corresponds to minimizing a multi-class hinge loss

$$\mathcal{L}(W) = \max\{0, 1 + \max_{k \neq y_n} w_k^T x_n - w_{y_n}^T x_n\} \quad \text{(Crammer-Singer multiclass SVM)}$$

- Loss = 0 if score on correct class is at least 1 more than score on next best scoring class

- Can optimize these similar to how we did it for binary SVM
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

\[
y^* = \arg \max_k \sum_{j \neq k} w_j^\top x^* \quad \text{(predict $k$ that wins over all others the most)}
\]

All-Pairs approach can be expensive at training and test time (but ways to speed up)
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

$y^* = \arg\max_k \sum_{j \neq k} w_j^\top x^*$ (predict $k$ that wins over all others the most)

All-Pairs approach can be expensive at training and test time (but ways to speed up)
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

$$
y^* = \arg \max_k \sum_{j \neq k} w_j \top x^*(\text{predict } k \text{ that wins over all others the most})$$

All-Pairs approach can be expensive at training and test time (but ways to speed up)
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

$$y_* = \arg \max_k w_k^T x_*$$
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

$$y^* = \arg \max_k w_k^T x^*$$

*One-vs-All Boundaries*

*Effective Pair-wise Boundaries*
Can use binary classifiers to solve multiclass problems

Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)

One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems

All-Pairs: Learn $K$-choose-2 binary classifiers, one for each pair of classes $(j, k)$

$$y_* = \arg \max_k \sum_{j \neq k} w^\top_{j,k} x_*$$

(predict $k$ that wins over all others the most)
Multiclass SVM using Binary SVM?

- Can use binary classifiers to solve multiclass problems
- Note: These approaches can be used with other binary classifiers too (e.g., logistic regression)
- One-vs-All (also called One-vs-Rest): Construct $K$ binary classification problems
  - All-Pairs: Learn $K$-choose-2 binary classifiers, one for each pair of classes $(j, k)$
    
    \[
    y_\ast = \arg \max_k \sum_{j \neq k} w_{j,k}^\top x_\ast
    \]
    
    (predict $k$ that wins over all others the most)

- All-Pairs approach can be expensive at training and test time (but ways to speed up)
One-Class Classification

- Can we learn from examples of just one class, say positive examples?
- May be desirable if there are many types of negative examples

Figure credit: Refael Chickvashvili

Intro to Machine Learning (CS771A) SVM (Contd). Multiclass and One-Class SVM 22
One-Class Classification

- Can we learn from examples of just one class, say positive examples?
- May be desirable if there are many types of negative examples

“Outlier/Novelty Detection” problems can also be formulated like this.

Figure credit: Refael Chickvashvili
There are two popular SVM-type approaches to solve one-class problems.

**Support Vector Data Description (SVDD)**
- Assume positives lie within a ball with smallest possible radius (and allow slacks)
- Known as "Support Vector Data Description" (SVDD). Proposed by [Tax and Duin, 2004]

**One-Class SVM (OC-SVM)**
- Find a max-margin hyperplane separating positives from origin (representing negatives)
- Known as "One-Class SVM" (OC-SVM). Proposed by [Schölkopf et al., 2001]

Optimization problems for both cases can be solved similarly as in binary SVM (e.g., via Lagrangian).
There are two popular SVM-type approaches to solve one-class problems:

1. **Approach 1**: Assume positives lie within a ball with **smallest possible radius** (and allow slacks)
   - Known as "**Support Vector Data Description**" (SVDD). Proposed by [Tax and Duin, 2004]

2. **Approach 2**: Find a max-margin hyperplane separating positives from origin (representing negatives)

Optimization problems for both cases can be solved similarly as in binary SVM (e.g., via Lagrangian)
There are two popular SVM-type approaches to solve one-class problems.

**Approach 1:** Assume positives lie within a ball with smallest possible radius (and allow slacks)
- Known as “Support Vector Data Description” (SVDD). Proposed by [Tax and Duin, 2004]

**Approach 2:** Find a max-marg hyperplane separating positives from origin (representing negatives)
- Known as “One-Class SVM” (OC-SVM). Proposed by [Schölkopf et al., 2001]
There are two popular SVM-type approaches to solve one-class problems

**Support Vector Data Description (SVDD)**

\[
\underset{R,c,\xi}{\operatorname{arg\,min}} \ R^2 + \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n
\]

s.t. \[\|x_n - c\| \leq R^2 + \xi_n \quad \forall n\]

\[\xi_n \geq 0\]

Prediction Rule: \(y_\ast = +1 \quad \text{if} \quad \|x_\ast - c\|^2 - R^2 < 0\)

**One-Class SVM (OC-SVM)**

\[
\underset{w,\rho,\xi}{\operatorname{arg\,min}} \ |w|^2 + \frac{1}{\nu N} \sum_{n=1}^{N} \xi_n - \rho
\]

s.t. \[w^\top x_n \geq \rho - \xi_n \quad \forall n\]

\[\xi_n \geq 0\]

Prediction Rule: \(y_\ast = +1 \quad \text{if} \quad w^\top x_\ast > \rho\)

- **Approach 1**: Assume positives lie within a ball with **smallest possible radius** (and allow slacks)
  - Known as "Support Vector Data Description" (SVDD). Proposed by [Tax and Duin, 2004]
- **Approach 2**: Find a **max-marg hyperplane** separating positives from origin (representing negatives)
  - Known as "One-Class SVM" (OC-SVM). Proposed by [Schölkopf et al., 2001]
- Optimization problems for both cases can be solved similarly as in binary SVM (e.g., via Lagrangian)
A nice property of SVM (and many other models) is that inputs only appear as inner products.

For example, recall the dual problem for soft-margin SVM had the form:

\[
\arg \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2} \alpha^T G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^T x_n \), and \( 1 \) is a vector of 1s.
A nice property of SVM (and many other models) is that inputs only appear as inner products.

For example, recall the dual problem for soft-margin SVM had the form

$$\arg\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top 1 - \frac{1}{2} \alpha^\top G \alpha$$

where $G$ is an $N \times N$ matrix with $G_{mn} = y_m y_n x_m^\top x_n$, and $1$ is a vector of 1s.

We can replace each inner-product by any general form of inner product, e.g.

$$k(x_n, x_m) = \phi(x_n)^\top \phi(x_m)$$
Nonlinear SVM?

- A nice property of SVM (and many other models) is that inputs only appear as inner products.
- For example, recall the dual problem for soft-margin SVM had the form
  \[
  \arg \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top \mathbf{G} \alpha
  \]
  where \( \mathbf{G} \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^\top x_n \), and \( \mathbf{1} \) is a vector of 1s.
- We can replace each inner-product by any general form of inner product, e.g.
  \[
  k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)
  \]
  .. where \( \phi \) is some transformation (e.g., a higher-dimensional mapping) of the data.
A nice property of SVM (and many other models) is that inputs only appear as inner products.

For example, recall the dual problem for soft-margin SVM had the form

$$\arg \max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^\top \mathbf{1} - \frac{1}{2} \alpha^\top \mathbf{G} \alpha$$

where $\mathbf{G}$ is an $N \times N$ matrix with $G_{mn} = y_m y_n \mathbf{x}_m \mathbf{x}_n^\top$, and $\mathbf{1}$ is a vector of 1s.

We can replace each inner-product by any general form of inner product, e.g.

$$k(\mathbf{x}_n, \mathbf{x}_m) = \phi(\mathbf{x}_n)^\top \phi(\mathbf{x}_m)$$

.. where $\phi$ is some transformation (e.g., a higher-dimensional mapping) of the data.

Note: Often the mapping $\phi$ doesn’t need to be explicitly computed (“kernel” magic - next class)!
Nonlinear SVM?

- A nice property of SVM (and many other models) is that inputs only appear as inner products.
- For example, recall the dual problem for soft-margin SVM had the form:

\[
\arg\max_{\alpha \leq C} \mathcal{L}_D(\alpha) = \alpha^T 1 - \frac{1}{2} \alpha^T G \alpha
\]

where \( G \) is an \( N \times N \) matrix with \( G_{mn} = y_m y_n x_m^T x_n \), and \( 1 \) is a vector of 1s.
- We can replace each inner-product by any general form of inner product, e.g.

\[
k(x_n, x_m) = \phi(x_n)^T \phi(x_m)
\]

where \( \phi \) is some transformation (e.g., a higher-dimensional mapping) of the data.

- Note: Often the mapping \( \phi \) doesn’t need to be explicitly computed (“kernel” magic - next class)!
- Can still learn a linear model in the new space but be nonlinear in the original space (wonderful!)
A hugely (perhaps the most!) popular classification algorithm

Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)

- Some popular ones: libSVM, LIBLINEAR, scikit-learn also provides SVM

† See: “Support Vector Machine Solvers” by Bottou and Lin
A hugely (perhaps the most!) popular classification algorithm

Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)

- Some popular ones: libSVM, LIBLINEAR, scikit-learn also provides SVM

Lots of work on scaling up SVMs† (both large $N$ and large $D$)

† See: “Support Vector Machine Solvers” by Bottou and Lin
A hugely (perhaps the most!) popular classification algorithm

Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)

- Some popular ones: libSVM, LIBLINEAR, scikit-learn also provides SVM

Lots of work on scaling up SVMs† (both large $N$ and large $D$)

Extensions beyond binary classification (e.g., multiclass, one-class, structured outputs)

† See: “Support Vector Machine Solvers” by Bottou and Lin
SVM: Some Notes

- A hugely (perhaps the most!) popular classification algorithm
- Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)
  - Some popular ones: libSVM, LIBLINEAR, scikit-learn also provides SVM
- Lots of work on scaling up SVMs† (both large $N$ and large $D$)
- Extensions beyond binary classification (e.g., multiclass, one-class, structured outputs)
- Can even be used for regression problems (Support Vector Regression)
  - The $\epsilon$-insensitive loss for regression does precisely that!

† See: “Support Vector Machine Solvers” by Bottou and Lin
SVM: Some Notes

- A hugely (perhaps the most!) popular classification algorithm
- Reasonably mature, highly optimized SVM softwares freely available (perhaps the reason why it is more popular than various other competing algorithms)
  - Some popular ones: libSVM, LIBLINEAR, scikit-learn also provides SVM
- Lots of work on scaling up SVMs† (both large $N$ and large $D$)
- Extensions beyond binary classification (e.g., multiclass, one-class, structured outputs)
- Can even be used for regression problems (Support Vector Regression)
  - The $\epsilon$-insensitive loss for regression does precisely that!
- Nonlinear extensions possible via kernels (next class)

† See: “Support Vector Machine Solvers” by Bottou and Lin