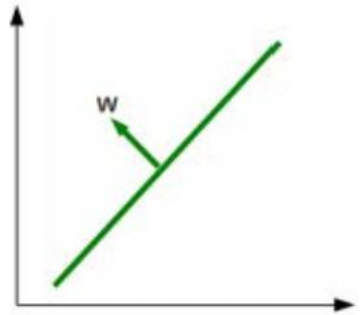
Perceptrons and SVMs

Dam Quang Tuan

Hyperplane

- Separates a D-dimensional space into two **half-spaces** (positive and negative)
- Defined by a normal vector $\boldsymbol{w} \in \mathbb{R}^D$ (pointing towards positive half-space)



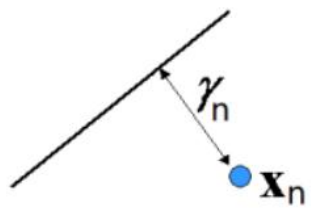
$b > 0$ means moving $\boldsymbol{w}^\top \boldsymbol{x} = 0$ along the direction of $\boldsymbol{w}$; $b < 0$ means in opp. dir.

$$\boldsymbol{w}^\top \boldsymbol{x} + b = 0$$

- Equation of the hyperplane: $\boldsymbol{w}^\top \boldsymbol{x} = 0$
- Assumption: The hyperplane passes through origin. If not, add a bias term b
- Distance of a point $\boldsymbol{x_n}$ from a hyperplane $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$

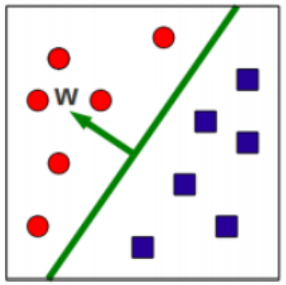
Can be positive or negative

$$\gamma_n = \frac{\boldsymbol{w}^T \boldsymbol{x}_n + b}{||\boldsymbol{w}||}$$



Hyperplane based (binary) classification

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



Prediction Rule

$$y_* = \text{sign}(\boldsymbol{w}^\top \boldsymbol{x}_* + b)$$

For multi-class classification with hyperplanes, there will be multiple hyperplanes (e.g., one for each pair of classes); more on this later

- 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 (SVM), etc
- The "direct" approach defines a model with params $\boldsymbol{w}$ (and optionally a bias param b)
  - The parameters are learned by optimizing a classification loss function (will soon see examples)
  - These are also discriminative approaches – $\boldsymbol{x}$ is not modeled but treated as fixed (given)
- The hyperplane need not be linear (e.g., can be made nonlinear using kernels; later)

Loss Functions for Classification

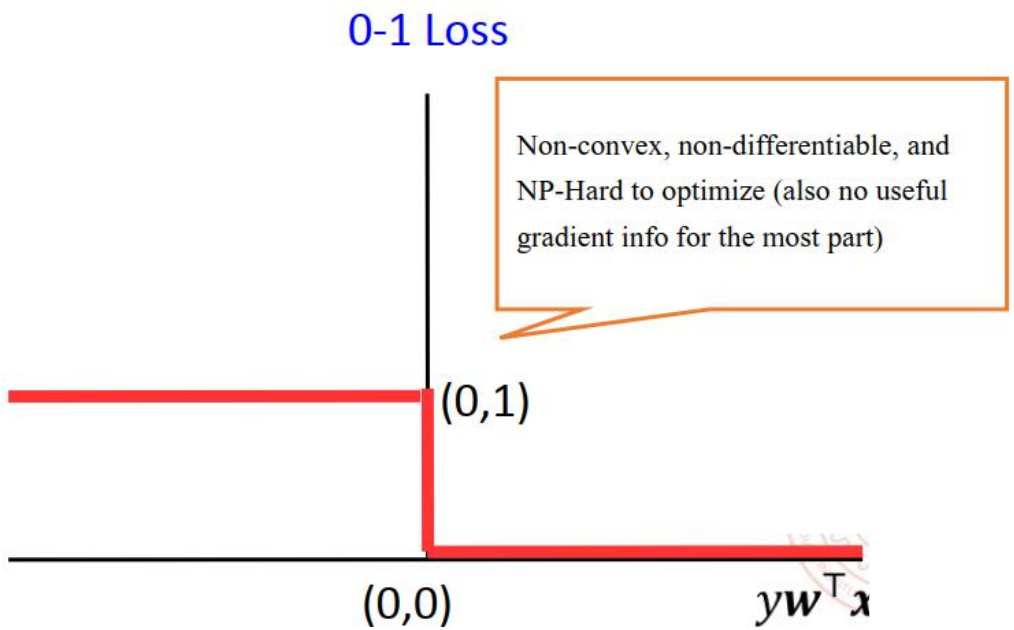
- In regression (assuming linear model $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$), some common loss fn

$$\ell(y, \hat{y}) = (y - \hat{y})^2 \qquad \ell(y, \hat{y}) = |y - \hat{y}|$$

- These measure the difference between the true output and model's prediction
- What about loss functions for classification where $\hat{y} = \mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$?
- Perhaps the most natural classification loss function would be a "0-1 Loss"
  - Loss = 1 if $\hat{y} \neq y$ and Loss = 0 if $\hat{y} = y$.
  - Assuming labels as +1/-1, it means

$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} < 0 \\ 0 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} \geq 0 \end{cases}$$

Same as $\mathbb{I}[y\boldsymbol{w}^\top \boldsymbol{x} < 0]$ or $\mathbb{I}[\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x}) \neq y]$

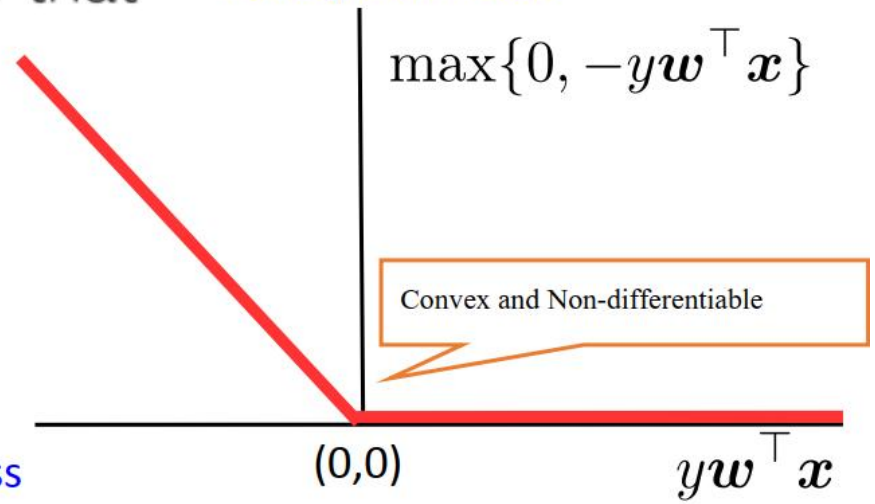


Loss Functions for Classification

- An ideal loss function for classification should be such that
  - Loss is small/zero if y and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ match
  - Loss is large/non-zero if y and $\mathbf{sign}(\boldsymbol{w}^\top \boldsymbol{x})$ do not match
  - Large positive $y\boldsymbol{w}^\top \boldsymbol{x} \Rightarrow$ small/zero loss
  - Large negative $y\boldsymbol{w}^\top \boldsymbol{x} \Rightarrow$ large/non-zero loss

"Perceptron" Loss

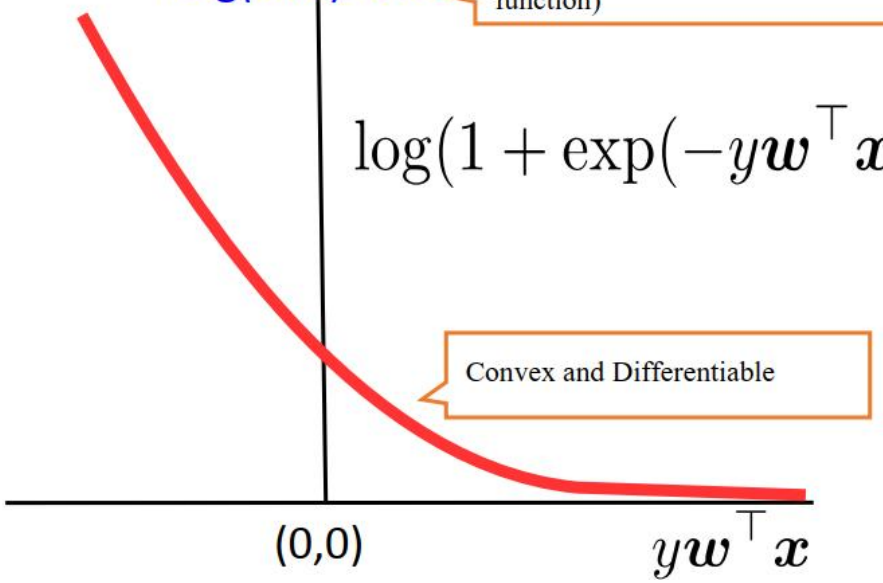
$$\max\{0, -y\boldsymbol{w}^\top \boldsymbol{x}\}$$



Log(istic) Loss

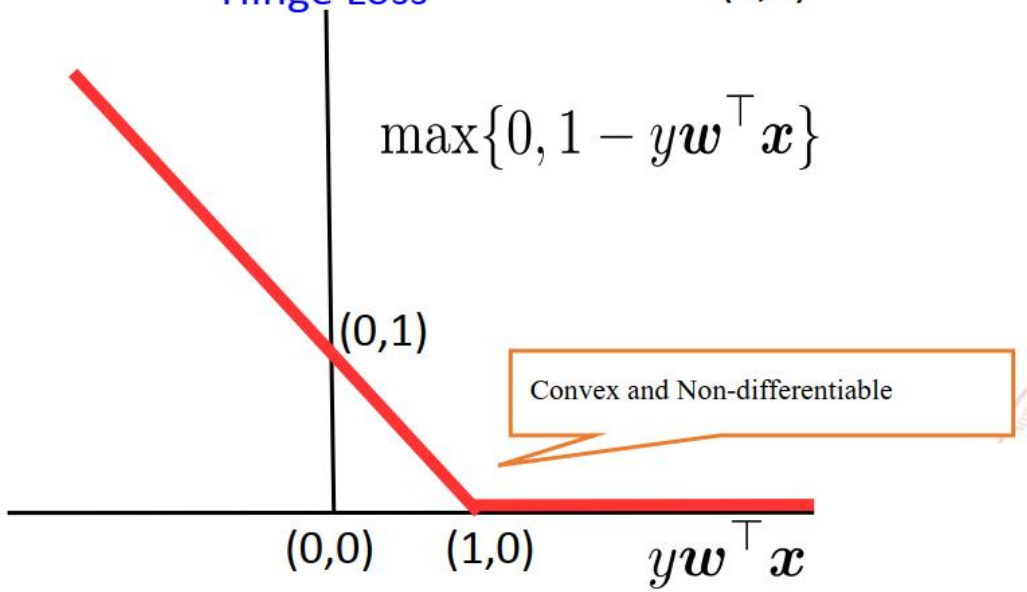
Already saw this in logistic regression (the likelihood resulted in this loss function)

$$\log(1 + \exp(-y\boldsymbol{w}^\top \boldsymbol{x}))$$



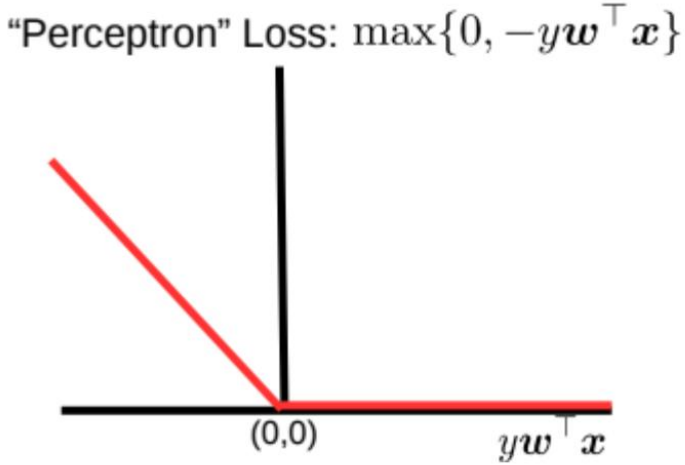
Hinge Loss

$$\max\{0, 1 - y\boldsymbol{w}^\top \boldsymbol{x}\}$$



Learning by Optimizing Perceptron Loss

- Let's ignore the bias term b for now. So the hyperplane is simply $\boldsymbol{w}^\top \boldsymbol{x} = 0$
- The Perceptron loss function: $L(w) = \sum_{n=1}^{N} \max\{0, -y_n \boldsymbol{w}^\top \boldsymbol{x}_n\}$. Let's do SGD



Subgradients w.r.t. $\boldsymbol{w}$

One randomly chosen example in each iteration

$$\boldsymbol{g}_n = \begin{cases} 0, & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n > 0 \\ -y_n \boldsymbol{x}_n & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0 \\ k y_n \boldsymbol{x}_n & \text{for } y_n \boldsymbol{w}^\top \boldsymbol{x}_n = 0 \end{cases} \quad (\text{where } k \in [-1, 0])$$

- If we use $k = 0$ then $\boldsymbol{g}_n = 0$ for $y_n \boldsymbol{w}^\top \boldsymbol{x}_n \geq 0$, and $\boldsymbol{g}_n = -y_n \boldsymbol{x}_n$ for $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < 0$
- Non-zero gradients only when the model makes a mistake on current example $(\boldsymbol{x}_n, y_n)$
- Thus SGD will update $\boldsymbol{w}$ only when there is a mistake (mistake-driven learning)

The Perceptron Algorithm

- Stochastic Sub-grad desc on Perceptron loss is also known as the Perceptron algorithm

Stochastic SubGD

1. Initialize $\boldsymbol{w} = \boldsymbol{w}^{(0)}, t = 0$, set $\eta_t = 1, \forall t$
2. Pick some $(\boldsymbol{x}_n, y_n)$ randomly.
3. If current $\boldsymbol{w}$ makes a mistake on $(\boldsymbol{x}_n, y_n)$, i.e., $y_n \boldsymbol{w}^{(t)\top} \boldsymbol{x}_n < 0$

$$\boldsymbol{w}^{(t+1)} = \boldsymbol{w}^{(t)} + y_n \boldsymbol{x}_n$$
$$t = t + 1$$

4. If not converged, go to step 2.

Note: An example may get chosen several times during the entire run

Mistake condition

Updates are "corrective": If $y\_n = +1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n < 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less negative. Likewise, if $y_n = -1$ and $\boldsymbol{w}^\top \boldsymbol{x}_n > 0$, after the update $\boldsymbol{w}^\top \boldsymbol{x}_n$ will be less positive

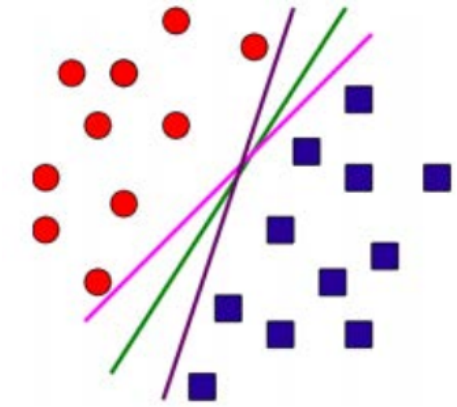
If training data is linearly separable, the Perceptron algo will converge in a finite number of iterations
(Block & Novikoff theorem)

- An example of an online learning algorithm (processes one training ex. at a time)
- Assuming $\boldsymbol{w}^{(0)} = \boldsymbol{0}$, easy to see that the final $\boldsymbol{w}$ has the form $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$
  - α_n is total number of mistakes made by the algorithm on example $(\boldsymbol{x}_n, y_n)$
  - As we'll see, many other models also have weights $\boldsymbol{w}$ in the form $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$

Meaning of α_n may be different

Perceptron and (lack of) Margins

- Perceptron would learn a hyperplane (of many possible) that separates the classes



Basically, it will learn the hyperplane which corresponds to the $\boldsymbol{w}$ that minimizes the Perceptron loss

- Doesn't guarantee any "margin" around the hyperplane
  - The hyperplane can get arbitrarily close to some training example(s) on either side
  - This may not be good for generalization performance

Kind of an "unsafe" situation to have – ideally would like it to be reasonably away from closest training examples from either class

- Can artificially introduce margin by changing the mistake condition to $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < \gamma$

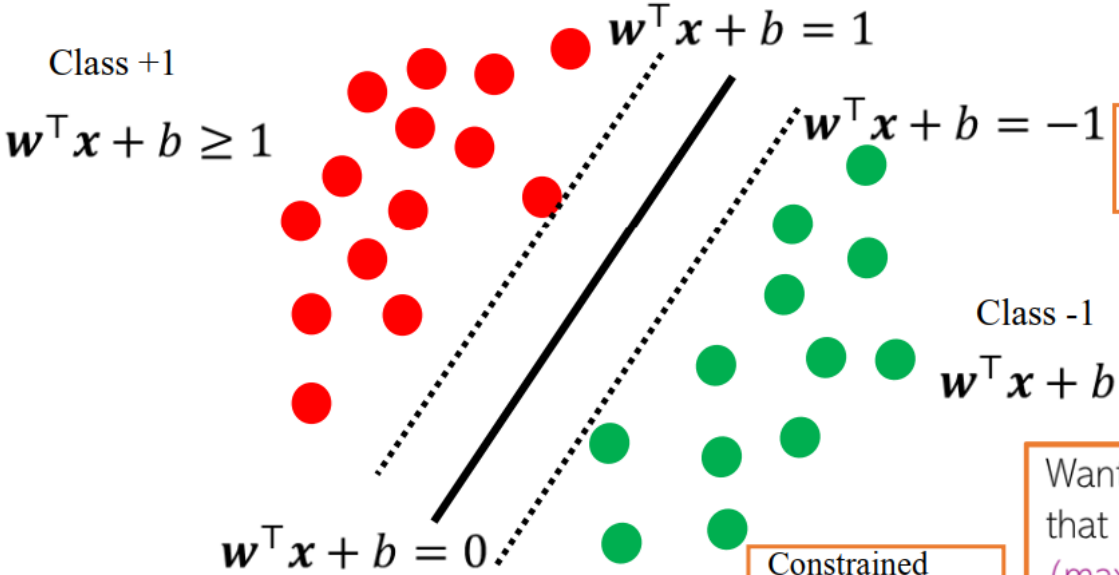
$\gamma > 0$ is some pre-specified margin

- Support Vector Machine (SVM) does it directly by learning the max. margin hyperplane

Support Vector Machine (SVM)

SVM originally proposed by Vapnik and colleagues in early 90s

- Hyperplane based classifier. Ensures a large margin around the hyperplane
- Will assume a linear hyperplane to be of the form $\boldsymbol{w}^\top \boldsymbol{x} + b = 0$ (nonlinear ext. later)



$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 \quad \text{if} \quad y_n = +1$$
$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 \quad \text{if} \quad y_n = -1$$
$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$$

Distance from the closest point (on either side)

"Margin" of the hyperplane

Distance of an input $\boldsymbol{x}_n$ from the h.p.

$$\gamma = \min_{1 \leq n \leq N} \frac{|\boldsymbol{w}^\top \boldsymbol{x}_n + b|}{\|\boldsymbol{w}\|} = \frac{1}{\|\boldsymbol{w}\|}$$

Want the hyperplane $(\boldsymbol{w}, b)$ such that this margin is maximized (max-margin hyperplane) and $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 \quad \forall n$

Constrained optimization problem

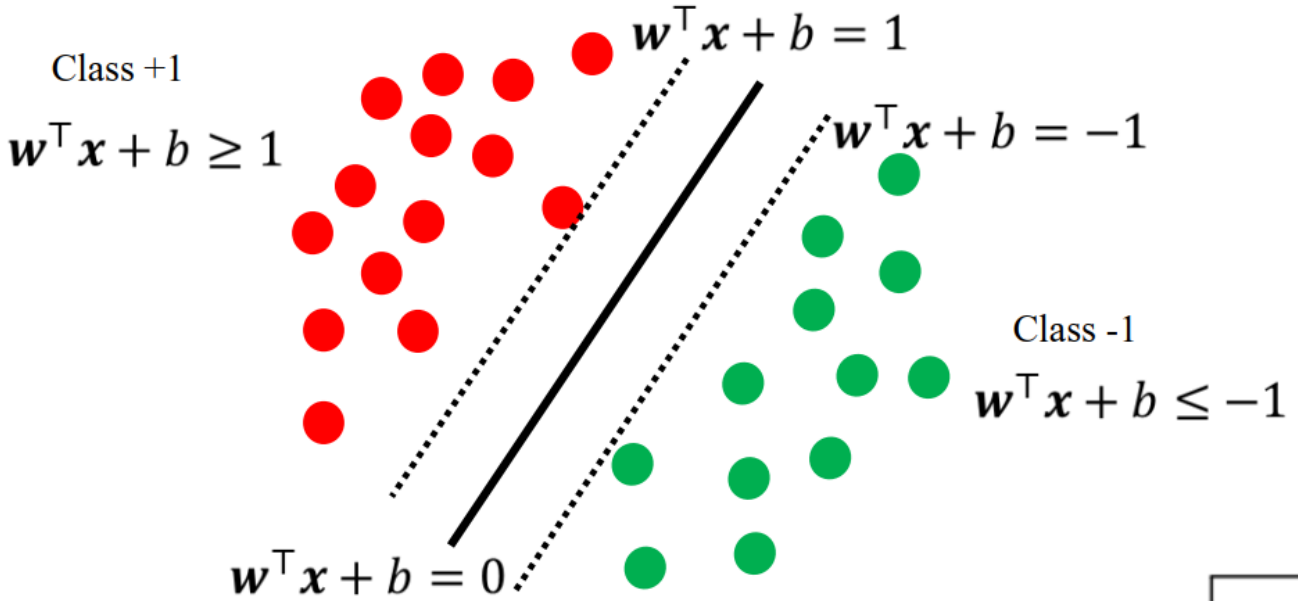
$$\text{Total margin} = \frac{2}{\|\boldsymbol{w}\|}$$

- Two other "supporting" hyperplanes defining a "no man's land"
  - Ensure that zero training examples fall in this region (will relax later)
  - The SVM idea: Position the hyperplane s.t. this region is as "wide" as possible

The 1/-1 in supp. h.p. equations is arbitrary; can replace by any scalar m/-m and solution won't change, except a simple scaling of $\boldsymbol{w}$

Hard-Margin SVM

- Hard-Margin: Every training example must fulfil margin condition $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1$
- Meaning: Must not have any example in the no-man's land



- Also want to maximize margin $2\gamma = \frac{2}{\|\boldsymbol{w}\|}$
- Equivalent to <u>minimizing</u> $\|\boldsymbol{w}\|^2$ or $\frac{\|\boldsymbol{w}\|^2}{2}$
- The objective func. for hard-margin SVM

Constrained optimization problem with N inequality constraints. Objective and constraints both are convex

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w}, b) = \frac{||\boldsymbol{w}||^2}{2}$$

$$\text{subject to} \quad y_n(\boldsymbol{w}^T \boldsymbol{x}_n + b) \geq 1, \qquad n = 1, \ldots, N$$

Solving Hard-Margin SVM

- The hard-margin SVM optimization problem is

$$\min_{\boldsymbol{w},b} \quad f(\boldsymbol{w},b) = \frac{||\boldsymbol{w}||^2}{2}$$
$$\text{subject to} \quad 1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \leq 0, \qquad n = 1,\ldots,N$$

- A constrained optimization problem. One option is to solve using Lagrange's method
- Introduce Lagrange multipliers α_n $(n = 1,\ldots,N)$, one for each constraint, and solve

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

- $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ denotes the vector of Lagrange multipliers
- It is easier (and helpful: we will soon see why) to solve the dual: min and then max

Solving Hard-Margin SVM

- The dual problem (min then max) is

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

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$ (problem becomes a bit more easy to solve)

Otherwise, the α_n's are coupled and some opt. techniques such as co-ordinate ascent can't easily be applied

- Take (partial) derivatives of $\mathcal{L}$ w.r.t. $\boldsymbol{w}$ and b and setting them to zero gives (verify)

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \Rightarrow \boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n \qquad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N} \alpha_n y_n = 0$$

α_n tells us how important training example $(\boldsymbol{x}_n, y_n)$ is

- The solution $\boldsymbol{w}$ is simply a weighted sum of all the training inputs
- Substituting $\boldsymbol{w} = \sum_{n=1}^{N} \alpha_n y_n \boldsymbol{x}_n$ in the Lagrangian, we get the dual problem as (verify)

This is also a "quadratic program" (QP) – a quadratic function of the variables $\boldsymbol{\alpha}$

$$\max_{\alpha \geq 0} \mathcal{L}_D(\boldsymbol{\alpha}) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\boldsymbol{x}_m^T \boldsymbol{x}_n)$$

Maximizing a concave function (or minimizing a convex function) s.t. $\boldsymbol{\alpha} \geq \boldsymbol{0}$ and $\sum_{n=1}^{N} \alpha_n y_n = 0$. Many methods to solve it.

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

G is an $N \times N$ p.s.d. matrix, also called the Gram Matrix, $G_{nm} = y_n y_m \boldsymbol{x}_n{}^\top \boldsymbol{x}_m$, and 1 is a vector of all 1s

(Note: For various SVM solvers, can see "Support Vector Machine Solvers" by Bottou and Lin)

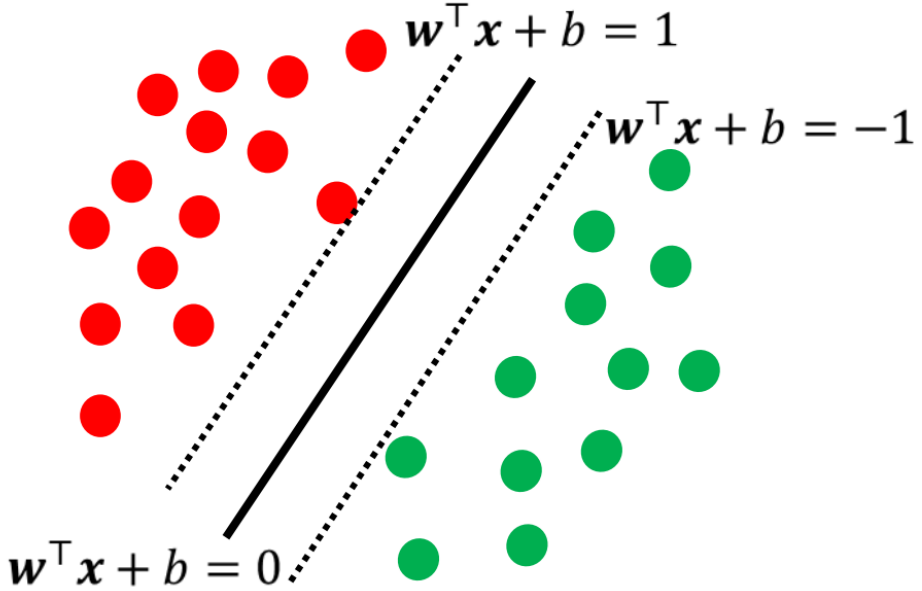
Solving Hard-Margin SVM

- One we have the α_n's by solving the dual, we can get $\boldsymbol{w}$ and b as

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

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

- A nice property: Most α_n's in the solution will be zero (sparse solution)

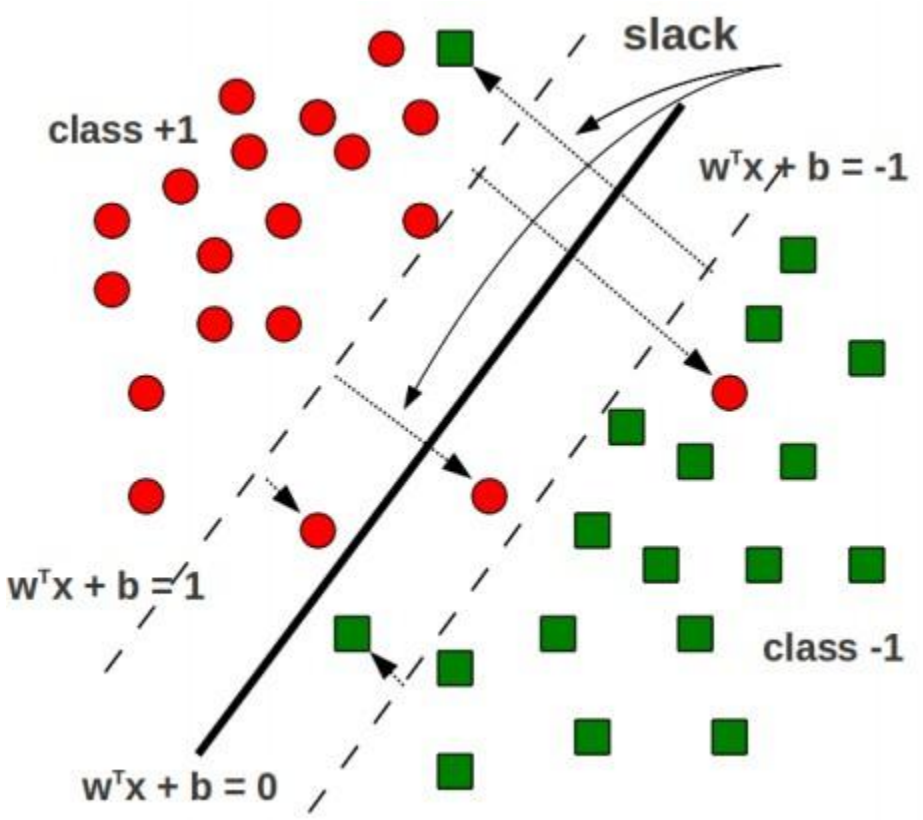


- Reason: KKT conditions
- For the optimal α_n's, we must have
- Thus α_n nonzero only if $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) = 1$, i.e., the training example lies on the boundary

$$\alpha_n\{1 - y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b)\} = 0$$

- These examples are called support vectors

Soft-Margin SVM (More Commonly Used)



- Allow some training examples to fall within the no-man's land (margin region)
- Even okay for some training examples to fall totally on the wrong side of h.p.
- Extent of "violation" by a training input $(\boldsymbol{x}_n, y_n)$ is known as slack $\xi_n \geq 0$
- $\xi_n > 1$ means totally on the wrong side

Soft-margin constraint:

$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \geq 1 - \xi_n \quad \text{if} \quad y_n = +1$$

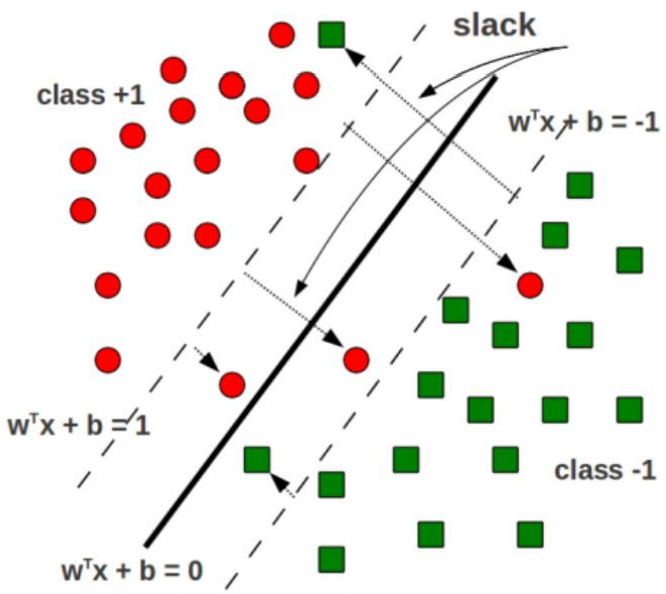
$$\boldsymbol{w}^\top \boldsymbol{x}_n + b \leq -1 + \xi_n \quad \text{if} \quad y_n = -1$$

$$y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n \quad \forall n$$

Soft-Margin SVM (Contd)

- Goal: Still want to maximize the margin such that
  - Soft-margin constraints $y_n(\boldsymbol{w}^\top \boldsymbol{x}_n + b) \geq 1 - \xi_n$ are satisfied for all training ex.
  - Do not have too many margin violations (sum of slacks $\sum_{n=1}^{N} \xi_n$ should be small)

Sum of slacks is like the training error



- The objective func. for soft-margin SVM

Inversely prop. to margin

Trade-off hyperparam

training error

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

$$\text{subject to } y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) \geq 1-\xi_n, \quad \xi_n \geq 0 \quad n = 1,\ldots,N$$

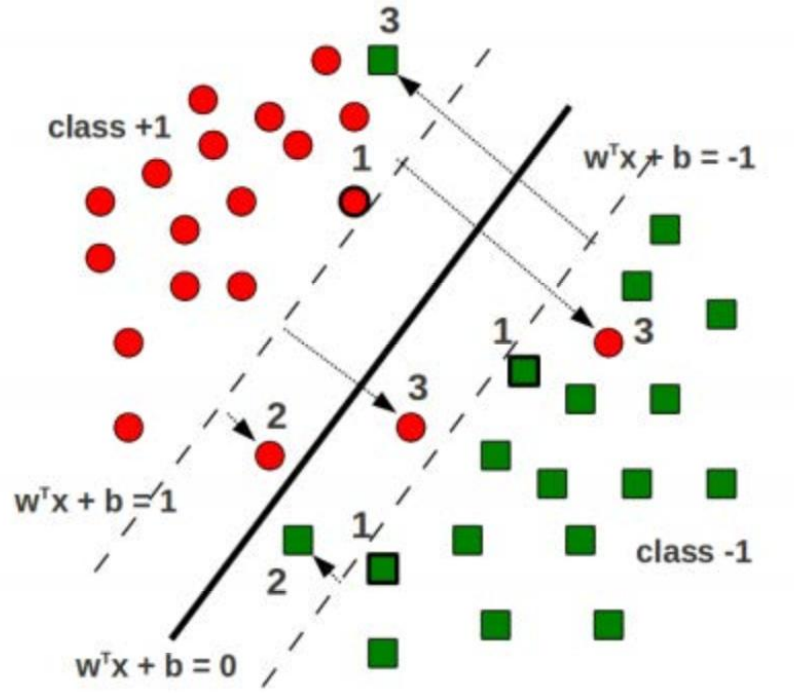
Constrained optimization problem with $2N$ inequality constraints. Objective and constraints both are convex

- Hyperparameter C controls the trade off between large margin and small training error (need to tune)
  - Large C: small training error but also small margin (bad)
  - Small C: large margin but large training error (bad)

Support Vectors in Soft-Margin SVM

- The hard-margin SVM solution had only one type of support vectors
  - All lied on the supporting hyperplanes $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_n + b = 1$ and $\boldsymbol{w}^{\mathsf{T}}\boldsymbol{x}_n + b = -1$

- The soft-margin SVM solution has <u>three</u> types of support vectors (with nonzero α_n)



1. Lying on the supporting hyperplanes
2. Lying within the margin region but still on the correct side of the hyperplane
3. Lying on the wrong side of the hyperplane (misclassified training examples)

Solving Soft-Margin SVM

- Recall the soft-margin SVM optimization problem

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \quad f(\boldsymbol{w},b,\boldsymbol{\xi}) = \frac{||\boldsymbol{w}||^2}{2} + C\sum_{n=1}^{N}\xi_n$$

$$\text{subject to} \quad 1 \le y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) + \xi_n, \quad -\xi_n \le 0 \qquad n = 1,\ldots,N$$

- Here $\boldsymbol{\xi} = [\xi_1, \xi_2, \ldots, \xi_N]$ is the vector of slack variables
- Introduce Lagrange multipliers α_n, β_n for each constraint and solve Lagrangian

$$\min_{\boldsymbol{w},b,\boldsymbol{\xi}} \max_{\alpha\ge 0,\boldsymbol{\beta}\ge 0} \mathcal{L}(\boldsymbol{w},b,\xi,\alpha,\beta) = \frac{||\boldsymbol{w}||^2}{2} + +C\sum_{n=1}^{N}\xi_n + \sum_{n=1}^{N}\alpha_n\{1 - y_n(\boldsymbol{w}^T\boldsymbol{x}_n + b) - \xi_n\} - \sum_{n=1}^{N}\beta_n\xi_n$$

- The terms in red color above were not present in the hard-margin SVM
- Two set of dual variables $\boldsymbol{\alpha} = [\alpha_1, \alpha_2, \ldots, \alpha_N]$ and $\boldsymbol{\beta} = [\beta_1, \beta_2, \ldots, \beta_N]$
- Will eliminate the primal var $\boldsymbol{w}$, b, $\boldsymbol{\xi}$ to get dual problem containing the dual variables

Solving Soft-Margin SVM

Note: if we ignore the bias term b then we don't need to handle the constraint $\sum_{n=1}^{N} \alpha_n y_n = 0$ (problem becomes a bit more easy to solve)

- The Lagrangian problem to solve

Otherwise, the α_n's are coupled and some opt. techniques such as co-ordinate aspect can't easily applied

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

- Take (partial) derivatives of $\mathcal{L}$ w.r.t. $\boldsymbol{w}, b$, and ξ_n and setting to zero gives

Weighted sum of training inputs

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{w}} = 0 \Rightarrow \boxed{\boldsymbol{w} = \sum_{n=1}^{N}\alpha_n y_n \boldsymbol{x}_n}, \quad \frac{\partial \mathcal{L}}{\partial b} = 0 \Rightarrow \sum_{n=1}^{N}\alpha_n y_n = 0, \quad \frac{\partial \mathcal{L}}{\partial \xi_n} = 0 \Rightarrow C - \alpha_n - \beta_n = 0$$

- Using $C - \alpha_n - \beta_n = 0$ and $\beta_n \geq 0$, we have $\alpha_n \leq C$ (for hard-margin, $\alpha_n \geq 0$)
- Substituting these in the Lagrangian $\mathcal{L}$ gives the Dual problem

The dual variables β don't appear in the dual problem!

Given $\boldsymbol{\alpha}$, $\boldsymbol{w}$ and b can be found just like the hard-margin SVM case

$$\max_{\boldsymbol{\alpha}\leq C,\boldsymbol{\beta}\geq 0} \mathcal{L}_D(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \sum_{n=1}^{N}\alpha_n - \frac{1}{2}\sum_{m,n=1}^{N}\alpha_m\alpha_n y_m y_n(\boldsymbol{x}_m^T\boldsymbol{x}_n) \quad \text{s.t.} \quad \sum_{n=1}^{N}\alpha_n y_n = 0$$

Maximizing a concave function (or minimizing a convex function) s.t. $\boldsymbol{\alpha} \leq \boldsymbol{C}$ and $\sum_{n=1}^{N}\alpha_n y_n = 0$. Many methods to solve it.

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

In the solution, $\boldsymbol{\alpha}$ will still be sparse just like the hard-margin SVM case. Nonzero α_n correspond to the support vectors

References

CS771: Intro to Machine Learning (Fall 2021), Nisheeth Srivastava, IIT Kanpur