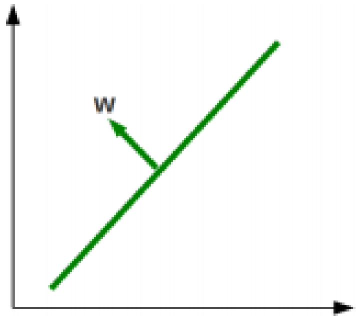
Hyperplane based Classifiers (1): The Perceptron Algorithm

CS771: Introduction to Machine Learning

Piyush Rai

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)



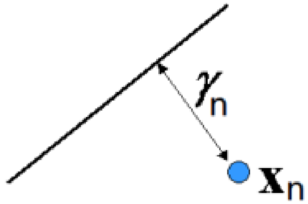
$b > 0$ means moving $\boldsymbol{w}^\top \boldsymbol{x} = 0$ along the direction of $\boldsymbol{w}$; $b < 0$ means in opp. dir.

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

- 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

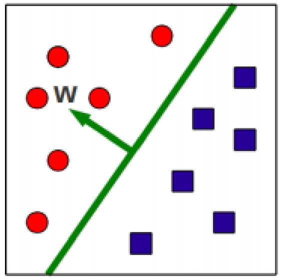
$$\gamma_n = \frac{\mathbf{w}^T \mathbf{x}_n + b}{||\mathbf{w}||}$$



Hyperplane based (binary) classification

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

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



Prediction Rule

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

- 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 (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)

Interlude: Loss Functions for Classification

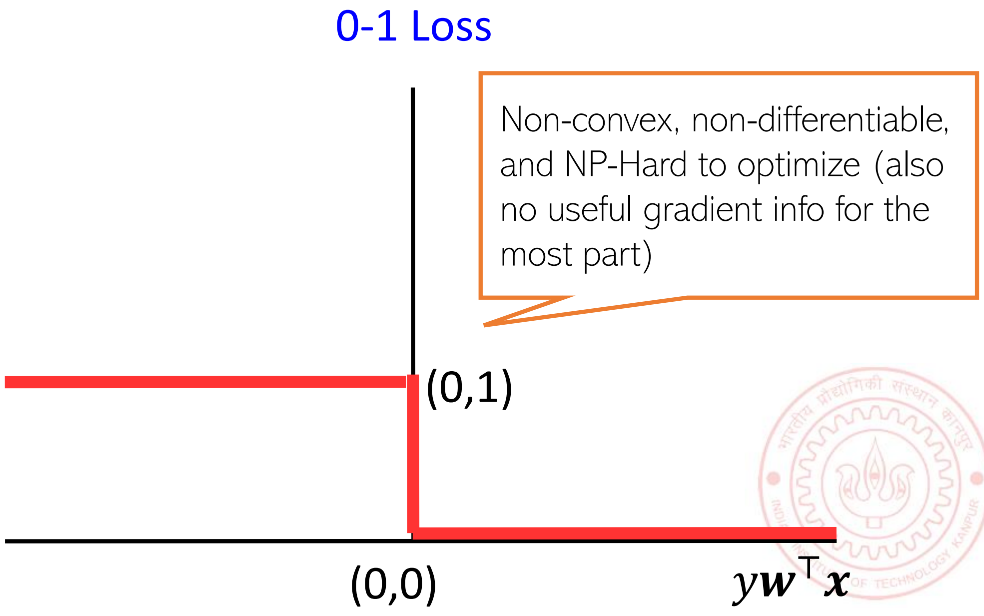
- In regression (assuming linear model $\hat{y} = \boldsymbol{w}^\top \boldsymbol{x}$), some common loss fn

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

- These measure the difference between the true output and model's prediction
- What about loss functions for <u>classification</u> 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

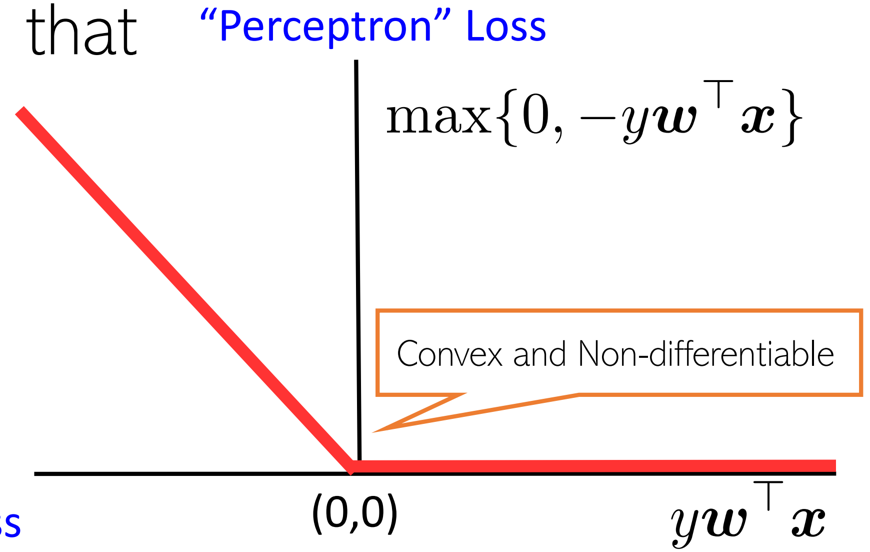
$$\ell(y, \hat{y}) = \begin{cases} 1 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} < 0 \\ 0 & \text{if } y\boldsymbol{w}^\top \boldsymbol{x} \geq 0 \end{cases}$$

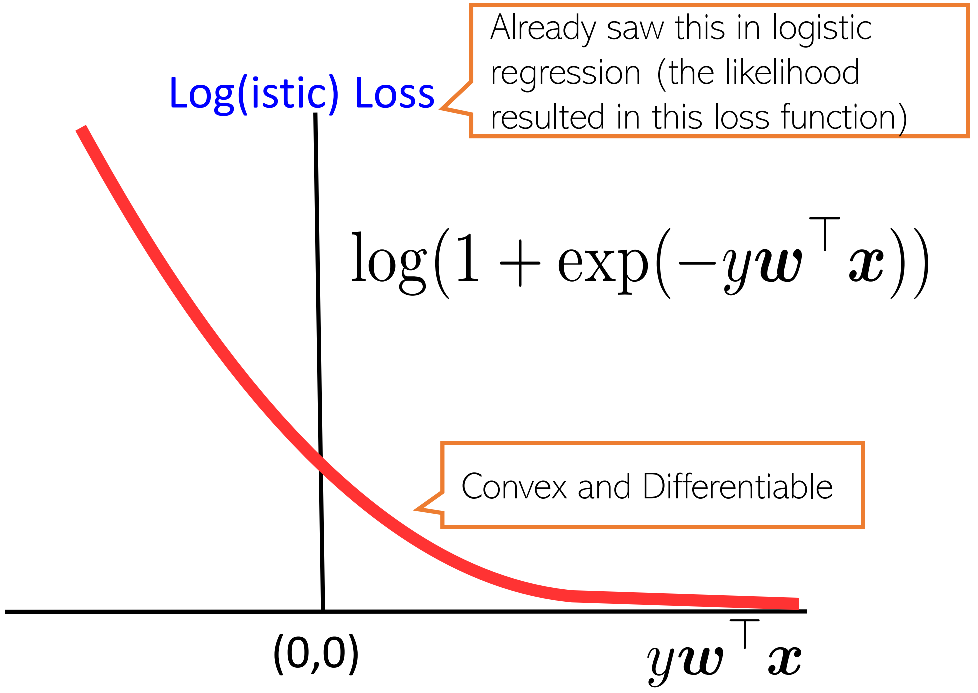
Same as $\mathbb{I}[y\boldsymbol{w}^\top \boldsymbol{x} < 0]$ or $\mathbb{I}[\mathrm{sign}(\boldsymbol{w}^\top \boldsymbol{x}) \neq y]$

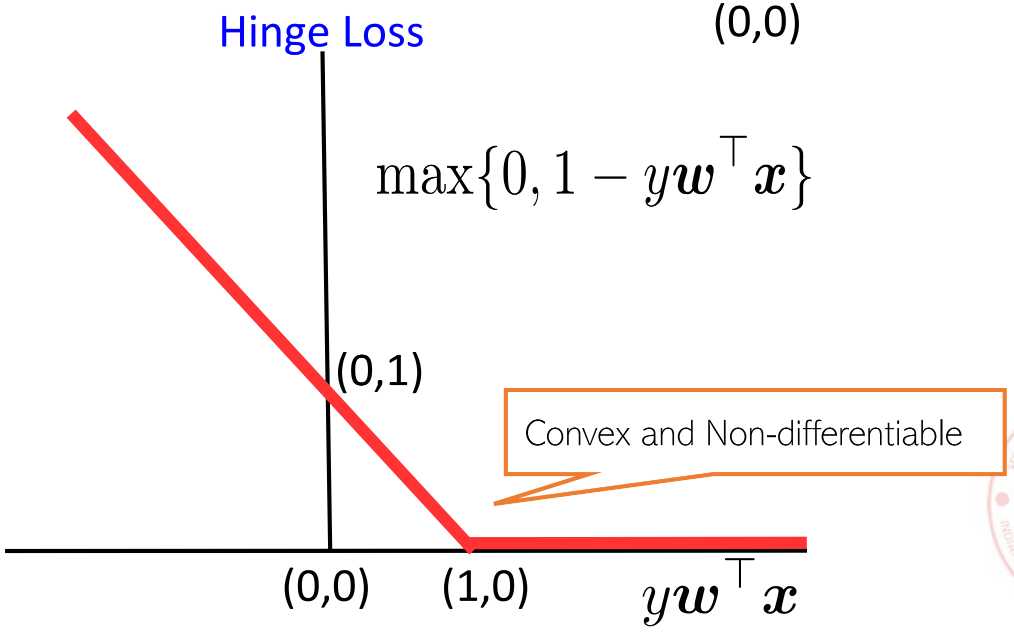


Interlude: Loss Functions for Classification

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



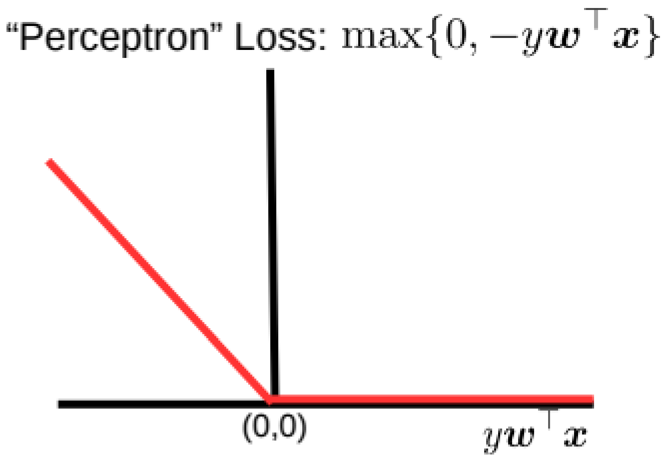




Learning by Optimizing Perceptron Loss

- Let's ignore the bias term $\boldsymbol{b}$ 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

One randomly chosen example in each iteration



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

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

- 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., $y_n \boldsymbol{w}^{(t)^\top} \boldsymbol{x}_n < 0$
$$\begin{aligned} \boldsymbol{w}^{(t+1)} &= \boldsymbol{w}^{(t)} + y_n \boldsymbol{x}_n \\ t &= t + 1 \end{aligned}$$
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

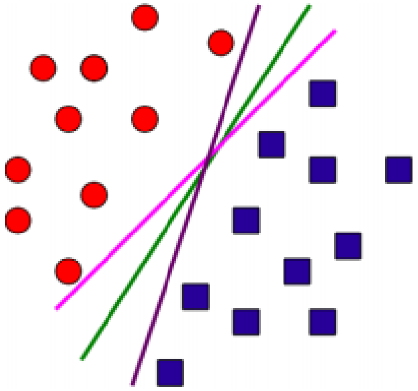
If training data is linearly separable, the Perceptron algo will converge in a finite number of iterations (Block & Novikoff theorem)

- 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

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

- 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

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

- Can artificially introduce margin by changing the mistake condition to $y_n \boldsymbol{w}^\top \boldsymbol{x}_n < \gamma$
- Support Vector Machine (SVM) does it directly by learning the max. margin hyperplane

Coming up next

- Support Vector Machines