COMS 4030A/7047A Adaptive Computation and Machine Learning

Hima Vadapalli

Semester I, 2022

Perceptron

Slides based heavily on course material by Eric Eton, Alan Fern, Piyush Rai and Andrew Ng

Logistic Regression

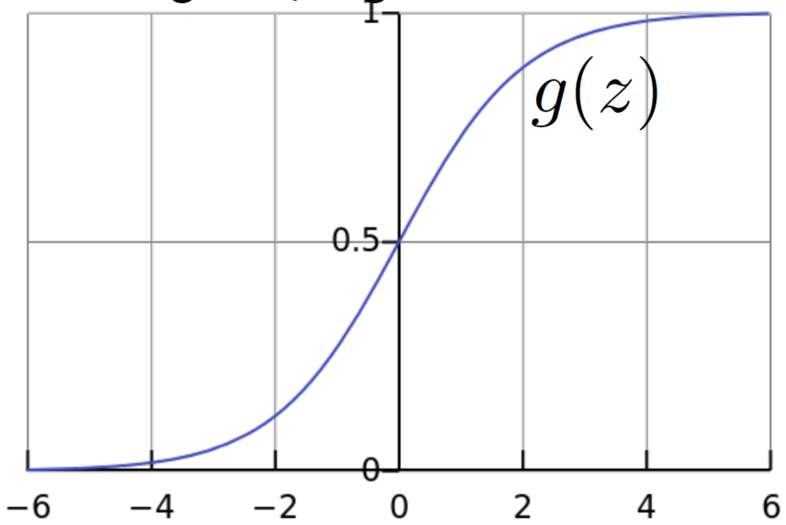
- Takes a probabilistic approach to learning discriminative functions (i.e., a classifier)
- $h_{\boldsymbol{\theta}}(\boldsymbol{x})$ should give $p(y = 1 \mid \boldsymbol{x}; \boldsymbol{\theta})$
  - Want $0 \leq h_{\boldsymbol{\theta}}(\boldsymbol{x}) \leq 1$
- Logistic regression model:

$$h_{\boldsymbol{\theta}}(\boldsymbol{x}) = g\left(\boldsymbol{\theta}^{\intercal}\boldsymbol{x}\right)$$

$$g(z) = \frac{1}{1 + e^{-z}}$$

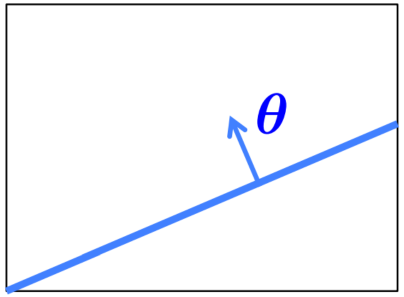
$$\boxed{h_{\boldsymbol{\theta}}(\boldsymbol{x}) = \frac{1}{1 + e^{-\boldsymbol{\theta}^{\intercal}\boldsymbol{x}}}}$$

Logistic / Sigmoid Function



Linear Classifiers

- A **hyperplane** partitions $\mathbb{R}^d$ into two half-spaces
  - Defined by the normal vector $\boldsymbol{\theta} \in \mathbb{R}^d$
    - $\boldsymbol{\theta}$ is orthogonal to any vector lying on the hyperplane

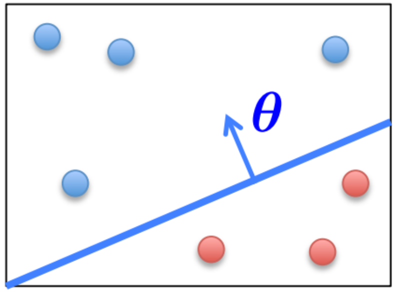


- Consider classification with +1, -1 labels ...

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

– Note that: $\boldsymbol{\theta}^\intercal \boldsymbol{x} > 0 \implies y = +1$

$\boldsymbol{\theta}^\intercal \boldsymbol{x} < 0 \implies y = -1$



The Perceptron

$$h(\boldsymbol{x}) = \text{sign}(\boldsymbol{\theta}^\intercal \boldsymbol{x}) \quad \text{where} \quad \text{sign}(z) = \begin{cases} 1 & \text{if } z \geq 0 \\ -1 & \text{if } z < 0 \end{cases}$$

- The perceptron uses the following update rule each time it receives a new training instance $(\boldsymbol{x}^{(i)}, y^{(i)})$

$$\theta_j \leftarrow \theta_j - \frac{\alpha}{2} \underbrace{\left(h_{\boldsymbol{\theta}}\left(\boldsymbol{x}^{(i)}\right) - y^{(i)}\right)}_{\text{either 2 or -2}} x_j^{(i)}$$

  - If the prediction matches the label, make no change
  - Otherwise, adjust $\boldsymbol{\theta}$

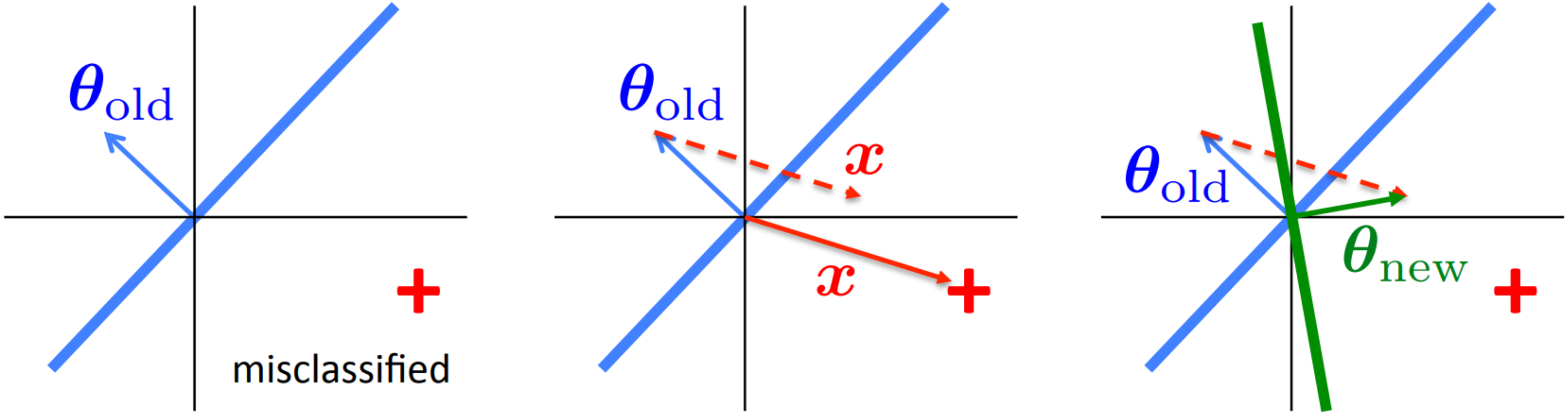
The Perceptron

Re-write as $\theta_j \leftarrow \theta_j + \alpha y^{(i)} x_j^{(i)}$ (only upon misclassification)

- Can eliminate α in this case, since its only effect is to scale $\boldsymbol{\theta}$ by a constant, which doesn't affect performance

Perceptron Rule: If $\boldsymbol{x}^{(i)}$ is misclassified, do $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \boldsymbol{x}^{(i)}$

Why the Perceptron Update Works



Why the Perceptron Update Works

- Consider the misclassified example (y = +1)
  - Perceptron wrongly thinks that $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}} \boldsymbol{x} < 0$
- Update:

$$\boldsymbol{\theta}_{\text{new}} = \boldsymbol{\theta}_{\text{old}} + y\boldsymbol{x} = \boldsymbol{\theta}_{\text{old}} + \boldsymbol{x} \qquad (\text{since } y = +1)$$

- Note that

$$\begin{aligned} \boldsymbol{\theta}_{\text{new}}^{\mathsf{T}} \boldsymbol{x} &= (\boldsymbol{\theta}_{\text{old}} + \boldsymbol{x})^{\mathsf{T}} \boldsymbol{x} \\ &= \boldsymbol{\theta}_{\text{old}}^{\mathsf{T}} \boldsymbol{x} + \boldsymbol{x}^{\mathsf{T}} \boldsymbol{x} \end{aligned}$$

$\|\boldsymbol{x}\|_2^2 > 0$

- Therefore, $\boldsymbol{\theta}_{\text{new}}^{\mathsf{T}} \boldsymbol{x}$ is less negative than $\boldsymbol{\theta}_{\text{old}}^{\mathsf{T}} \boldsymbol{x}$
  - So, we are making ourselves more correct on this example!

The Perceptron Cost Function

- The perceptron uses the following cost function

$$J_p(\theta) = \frac{1}{n}\sum_{i=1}^{n} \max\left(0, -y^{(i)}\theta^T x^{(i)}\right)$$

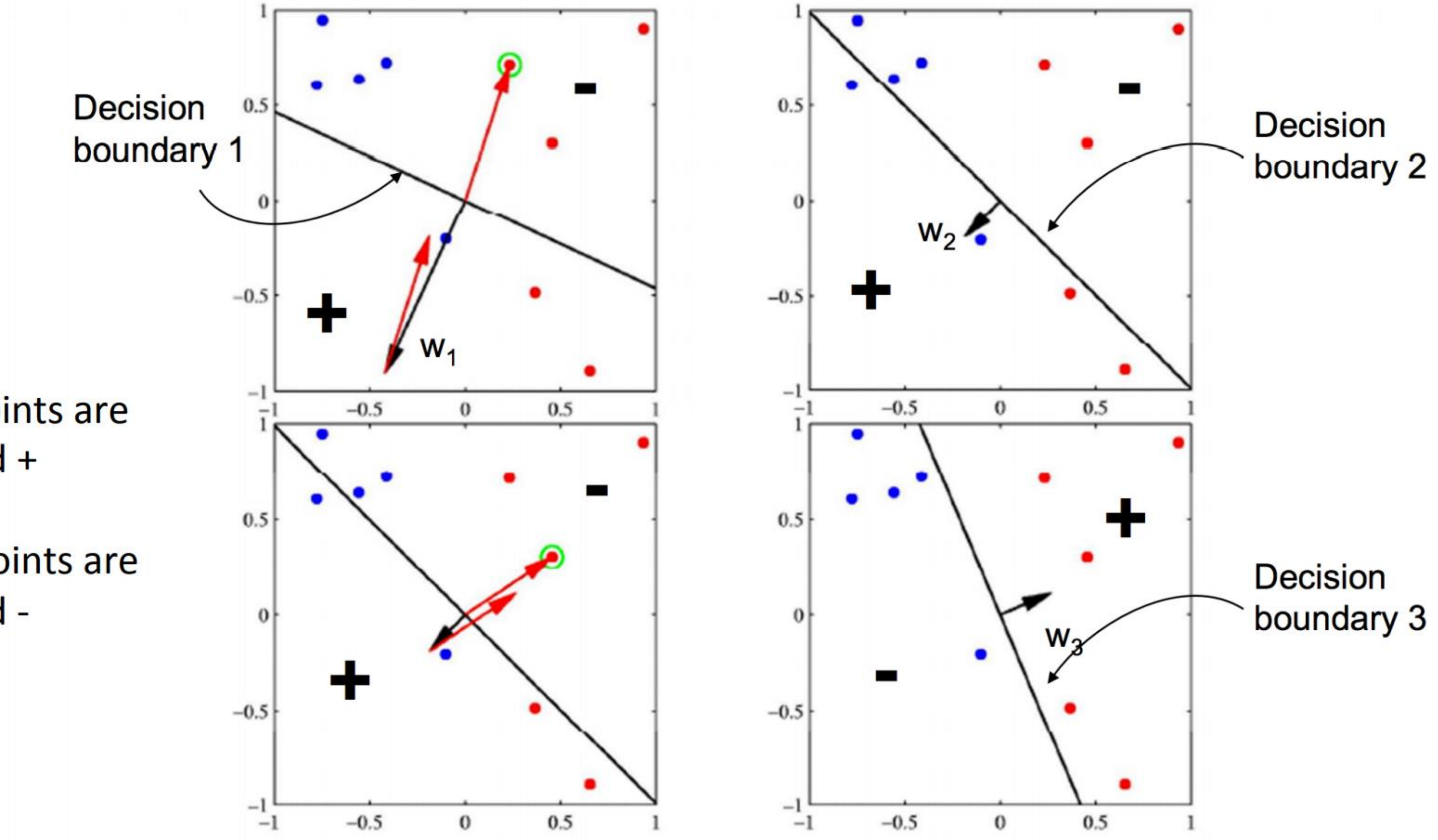
Where $\max\left(0, -y^{(i)}\theta^T x^{(i)}\right)$

= 0 if prediction is correct

otherwise it is the confidence in the misprediction

Online Perceptron Algorithm

When an error is made, moves the weight in a direction that corrects the error



Red points are labeled +

Blue points are labeled -

Improving the Perceptron

- The Perceptron produces many $\boldsymbol{\theta}$'s during training
- The standard Perceptron simply uses the final $\boldsymbol{\theta}$ at test time
  - This may sometimes not be a good idea!
  - Some other $\boldsymbol{\theta}$ may be correct on 1,000 consecutive examples, but one mistake ruins it!

- **Idea:** Use a combination of multiple perceptrons
  - (i.e., neural networks!)
- **Idea:** Use the intermediate $\boldsymbol{\theta}$'s
  - **Voted Perceptron**: vote on predictions of the intermediate $\boldsymbol{\theta}$'s
  - **Averaged Perceptron**: average the intermediate $\boldsymbol{\theta}$'s