University of Colorado **Boulder**

CSCI 5622: Machine Learning

Lecture 3

Outline

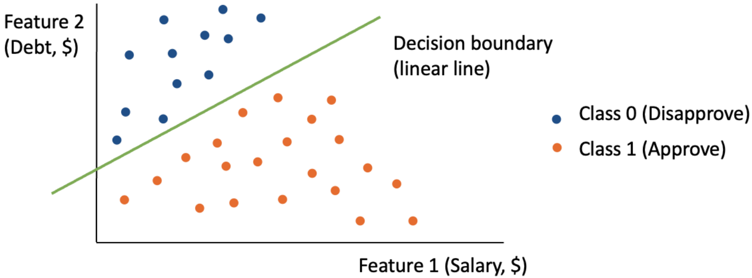
- Perceptron
- Example & Representation
- Perceptron Learning Algorithm (PLA)

(* Part of the following slides are taken from Dr. Malik Magdon-Ismail's Machine Learning class.)

Perceptron: Example

Credit approval or denial

- Task: Approve or deny credit (binary classification task)
- Features: Salary, debt, years in residence, etc.



Perceptron: Example & Representation

- Input features $\{x_1, \ldots, x_D\}$
- Assign importance to input features and compute a Credit Score

$$CreditScore = \sum_{i=1}^{D} w_i x_i$$

In the above, weights convey importance if features are in the same range

- Approve credit if $\sum_{i=1}^{D} w_i x_i > threshold$
- Deny credit if $\sum_{i=1}^{D} w_i x_i < threshold$
- How to choose the importance of weights?

Perceptron: Example & Representation

- Approve credit if $\sum_{i=1}^{D} w_i x_i > \textit{threshold}$
- Deny credit if $\sum_{i=1}^{D} w_i x_i < \textit{threshold}$
- Can be written more formally as

$$h(\mathbf{x}) = sign\left(\left(\sum_{i=1}^{d} w_i x_i\right) + w_0\right)$$

or

$$h(\mathbf{x}) = sign(\mathbf{w}^T \mathbf{x})$$

where $\mathbf{w} = [w_0, w_1, \ldots, w_d]^T$ and $\mathbf{x} = [1, x_1, \ldots, x_d]^T$

Perceptron: Representation

Classify data into two classes

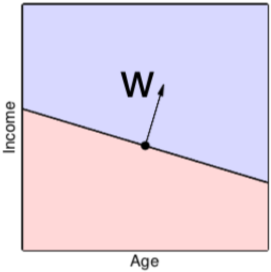
- Input: $\mathbf{x} \in \mathbb{R}^D$ (features, attributes, etc.)
- Output: $y \in \{-1, 1\}$ (labels)
- Model: $h : \mathbf{x} \rightarrow y$

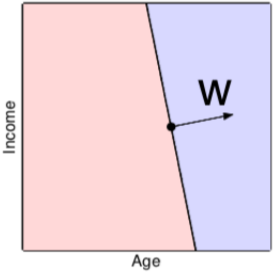
$$h(\mathbf{x}) = sign(\mathbf{w}^T\mathbf{x}) = \begin{cases} 1\,, & \mathbf{w}^T\mathbf{x} > 0 \\ -1\,, & \mathbf{w}^T\mathbf{x} < 0 \end{cases}$$

Perceptron: Representation

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^{\mathrm{T}}\mathbf{x})$$

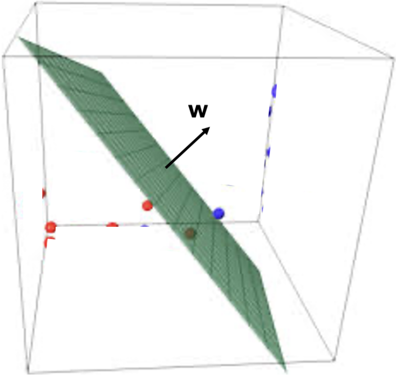
(Problem 1.2 in LFD)





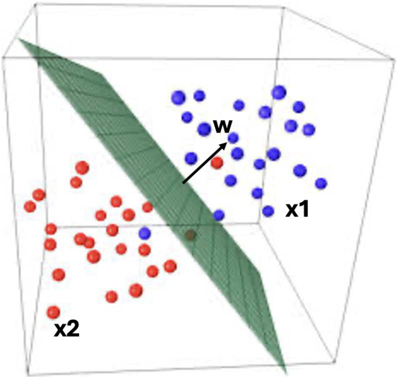
How to find the perpendicular vector to a line?
The vector perpendicular to line $w_1x_1 + w_2x_2 + w_0 = 0$ can be written as $\mathbf{w} = [w_0, w_1, w_2]^T$ (if the line does not pass through the beginning of the axis) or $\mathbf{w} = [w_1, w_2]^T$ (if the line does passes through the beginning of the axis) (See Supplementary Handout)

Perceptron: Representation



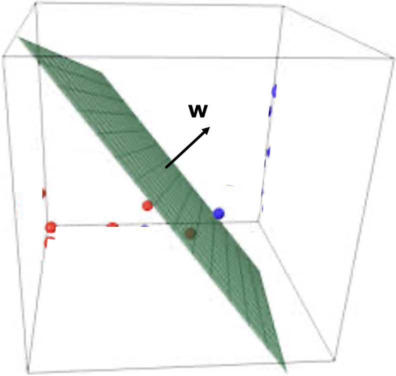
The vector perpendicular to a hyperplane $w_1 x_1 + w_2 x_2 + \ldots + w_D x_D + w_0 = 0$ can be written as $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_D]^T$

Perceptron: Representation



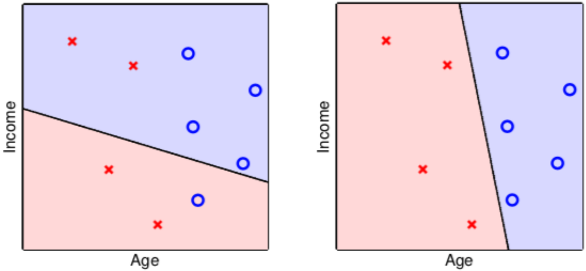
$\mathbf{w}^T\mathbf{x_1} > 0$, since $\mathbf{w}$ and $\mathbf{x_1}$ point toward the same direction
$\mathbf{w}^T\mathbf{x_2} < 0$, since $\mathbf{w}$ and $\mathbf{x_2}$ point toward the opposite direction

Perceptron: Representation



The vector perpendicular to a hyperplane $w_1 x_1 + w_2 x_2 + \ldots + w_D x_D + w_0 = 0$ can be written as $\mathbf{w} = [w_0, w_1, w_2, \ldots, w_D]^T$

Perceptron: Representation

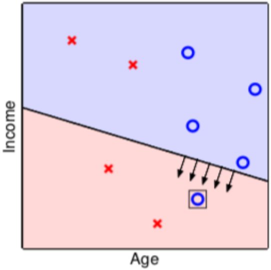


A perceptron fits the data by using a line to separate the $+1$ from -1 data.

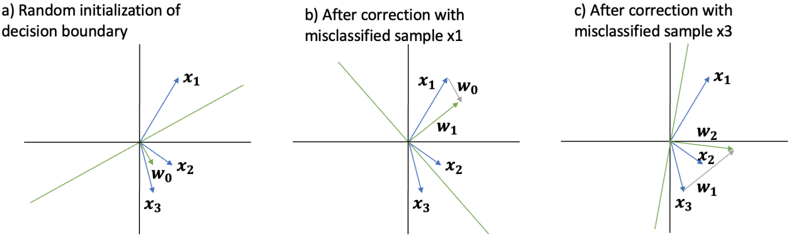
Fitting the data: How to find a hyperplane that *separates* the data?
("It's obvious - just look at the data and draw the line," is not a valid solution.)

Perceptron Learning Algorithm

Idea! Start with some weight vector and try to improve it.

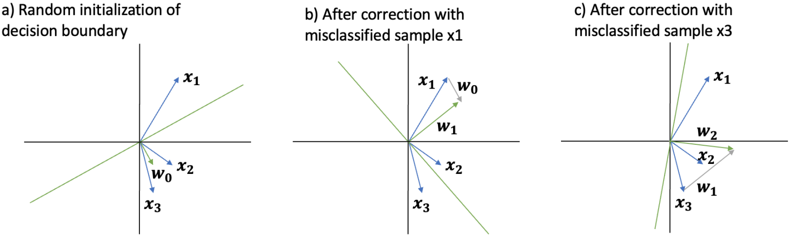


Perceptron Learning Algorithm: Example



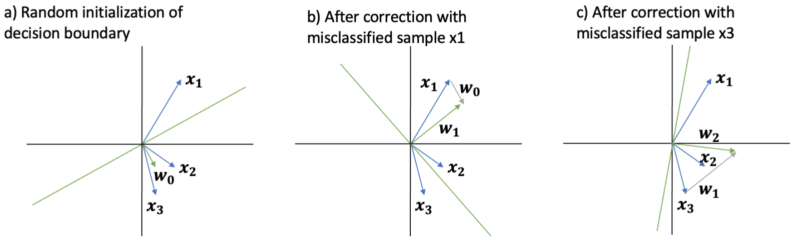
Assuming that $\{\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3\}$ belong to the same class ($y = 1$), after the initial updates, successive corrections become smaller and the algorithm "fine tunes" the position of the weight vector.

Perceptron Learning Algorithm: Example



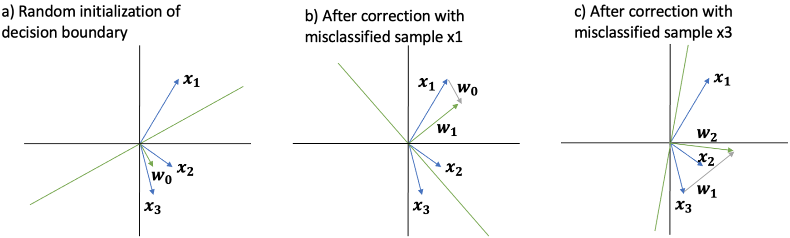
a) $\mathbf{w_0}^T\mathbf{x_1} < 0$, $\mathbf{w_0}^T\mathbf{x_2} > 0$, $\mathbf{w_0}^T\mathbf{x_3} > 0$: Sample 1 is incorrectly classified, therefore we need to adjust w_0 so that we correct this.
b) We do so by adding $\mathbf{x_1}$ to $\mathbf{w_0}$, since this will shift the line so that $\mathbf{x_1}$ lies on the right side of the line: $\mathbf{w_1} = \mathbf{w_0} + \mathbf{x_1}$

Perceptron Learning Algorithm: Example



b) $\mathbf{w_1}^T\mathbf{x_1} > 0$, $\mathbf{w_1}^T\mathbf{x_2} > 0$, $\mathbf{w_1}^T\mathbf{x_3} < 0$: Sample 3 is incorrectly classified, therefore we need to adjust w_1 so that we correct this.
c) We do so by adding $\mathbf{x_3}$ to $\mathbf{w_1}$, since this will shift the line so that $\mathbf{x_3}$ lies on the right side of the line: $\mathbf{w_2} = \mathbf{w_1} + \mathbf{x_3}$

Perceptron Learning Algorithm: Example



c) $\mathbf{w_2}^T\mathbf{x_1} > 0$, $\mathbf{w_2}^T\mathbf{x_2} > 0$, $\mathbf{w_2}^T\mathbf{x_3} > 0$: All samples are correctly classified. No further action is needed.

Perceptron Learning Algorithm

A simple iterative method

Incremental learning on single example at a time

1. Initialization $\mathbf{w}(\mathbf{0}) = 0$ (or any other vector)
2. for $t = 1, 2, 3, \ldots$
   a. From $\{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$ pick a misclassified sample
   b. Call the misclassified sample $(\mathbf{x_s}, y_s)$: $sign(\mathbf{w(t)}^T\mathbf{x_s}) \neq y_s$ $(\mathbf{w(t)}^T\mathbf{x_s} = -1$ if $y_s = 1$; $\mathbf{w(t)}^T\mathbf{x_s} = 1$ if $y_s = -1)$
   c. Update the weight:
      $\mathbf{w(t+1)} = \mathbf{w(t)} + y_s\mathbf{x_s}$, i.e.,
      $\mathbf{w(t+1)} = \mathbf{w(t)} + \mathbf{x_s}$, if $y_s = 1$ (the vector of the new hyperplane should point toward the **same direction** as the misclassified sample)
      $\mathbf{w(t+1)} = \mathbf{w(t)} - \mathbf{x_s}$, if $y_s = -1$ (the vector of the new hyperplane should point toward the **opposite direction** from the misclassified sample)
   d. $t \leftarrow t + 1$

Perceptron Learning Algorithm

Algorithm Learning Rate

- Vectors $\mathbf{w}$ are not necessarily normalized
- If $\|\mathbf{w}(t)\| \gg \|\mathbf{x_s}\|$ the new weight vector $\mathbf{w}(t) + \mathbf{x_s}$ is almost equal to $\mathbf{w}(t)$
- We can add a learning rate $\alpha > 0$ in the update
    - $\mathbf{w}(t+1) = \mathbf{w}(t) \pm \alpha \mathbf{x_s}$
    - if $\alpha \gg 0$, $\mathbf{x_s}$ will have a large impact on the update
    - if $\alpha \equiv 0$, $\mathbf{x_s}$ will not influence much the update
    - Learning rate a typical hyperparameter of many learning algorithms and depends on the data of each problem
    - It can be larger in the first learning steps, and decreasing later on, e.g., $\alpha = \frac{1}{t}c$, where t is the iteration and c is a constant

Perceptron Learning Algorithm

Algorithm Convergence

- The rule update considers a training sample at a time and may "destroy" the classification of other samples
- If the data can be fit by a linear separator (linearly separable), then after some finite number of steps, PLA is guaranteed to arrive to a correct solution.
- What if the data cannot be fit by a perceptron?
  - We can find infinitely many combinations of perceptrons that fit the data $\rightarrow$ neural networks

Perceptron Learning Algorithm

Algorithm Cost

- PLA is a local, greedy algorithm
- This can lead to an exponential number of updates of the weight
- In the following figure:
  - Two almost antiparallel vectors are to be classified in the same half-space
  - PLA rotates the separation line in one of the two directions and will require more and more time when the angle between the two vectors approaches 180 degrees

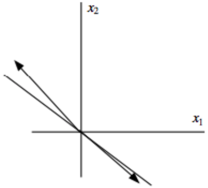


Fig. 4.13. Worst case for perceptron learning (input space)

Parametric v.s. non-parametric models

- Many possible ways to categorize learning models
- One way is to ask this question
  - Does the model have a fixed number of parameters or does it grow with the amount of training data?
- Non-parametric models (or instance/memory-based)
  - more flexible
  - computationally intractable for large datasets
  - e.g. K-NN
- Parametric models
  - faster to use
  - make strong assumptions about data
  - e.g. linear perceptron, linear regression

Perceptron Learning Algorithm

Question

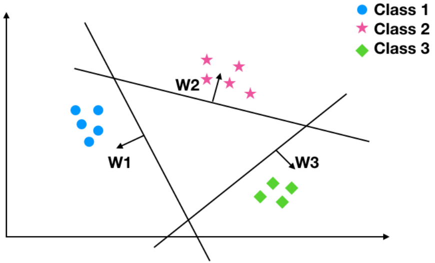
Assume the following 2-dimensional data space with three classes C_1, C_2, C_3, and three linear classification boundaries $\mathbf{w_1}$, $\mathbf{w_2}$, and $\mathbf{w_3}$. If $\mathbf{x_1} \in C_1$, $\mathbf{x_2} \in C_2$, and $\mathbf{x_3} \in C_3$, please select the equations that hold:

(A) $\mathbf{w}_1^T\mathbf{x_1} > 0$, $\mathbf{w}_2^T\mathbf{x_2} > 0$, $\mathbf{w}_3^T\mathbf{x_3} > 0$

(B) $\mathbf{w}_1^T\mathbf{x_1} < 0$, $\mathbf{w}_2^T\mathbf{x_2} < 0$, $\mathbf{w}_3^T\mathbf{x_3} < 0$

(C) $\mathbf{w}_1^T\mathbf{x_1} > 0$, $\mathbf{w}_2^T\mathbf{x_2} < 0$, $\mathbf{w}_3^T\mathbf{x_3} < 0$

(D) $\mathbf{w}_1^T\mathbf{x_1} < 0$, $\mathbf{w}_2^T\mathbf{x_2} > 0$, $\mathbf{w}_3^T\mathbf{x_3} > 0$



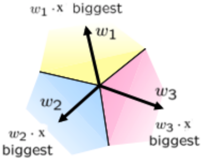
Perceptron Learning Algorithm

Question

Multiclass perceptron

Perceptron representation for more than two classes

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{1, 2, \ldots, K\}$
- Training data: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Model: $y = f(\mathbf{x}) = arg\max_{k \in \{1,\ldots,K\}} \mathbf{w_k}^T \mathbf{x}$ (the signed distance of sample $\mathbf{x}$ to its assigned class should be greater than its distance from all the other classes)
- Model parameters: weights $\mathbf{w_1}, \ldots, \mathbf{w_K}$



Multiclass perceptron

Perceptron learning for more than two classes

- Input: $\mathbf{x} \in \mathbb{R}^D$
- Output: $y \in \{1, 2, \ldots, K\}$
- Training data: $\mathcal{D}^{train} = \{(\mathbf{x_1}, y_1), \ldots, (\mathbf{x_N}, y_N)\}$
- Model: $y = f(\mathbf{x}) = arg\max_{k \in \{1,\ldots,K\}} \mathbf{w_k}^T \mathbf{x}$ (the signed distance of sample $\mathbf{x}$ to its assigned class should be greater than its distance from all the other classes)
- Model parameters: weights $\mathbf{w_1}, \ldots, \mathbf{w_K}$
- Cost function:
  $\min_{\mathbf{w_1},\ldots,\mathbf{w_K}} \frac{1}{N} \sum_{n=1}^{N} \left[\left(\max_{k=1\ldots,K} \mathbf{w_k}^T \mathbf{x_n} \right) - \mathbf{w}_{y_n}^T \mathbf{x_n} \right]$
  Miminizes the difference between signed distance of sample $\mathbf{x_n}$ from currently assigned class, and sample $\mathbf{x_n}$ from labelled (correct) class. Can be solved similar to the 2-class perceptron or via approximating the cost function followed by gradient descent.

Summary

- Perceptron is a simple learning algorithm
  - decision boundary defined by a hyper-plane
  - hyper-plane is learned from the training data
- Reading materials
  - Abu-Mostafa 1.1.2