10-301/601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

Perceptron

Matt Gormley
Lecture 6
Feb. 4, 2022

Q&A

Q: How do we define a distance function when the features are categorical (e.g. weather takes values {sunny, rainy, overcast})?

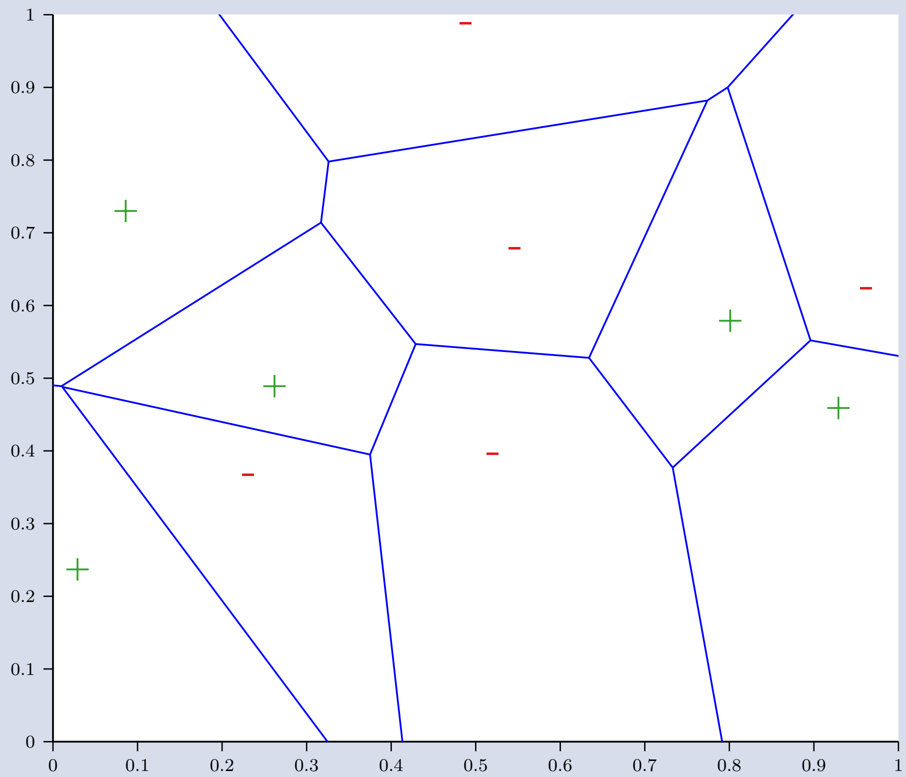
A: Step 1: Convert from categorical attributes to numeric features (e.g. binary)

Step 2: Select an appropriate distance function (e.g. Hamming distance)

Q&A

Q: Those decision boundary figures for KNN were really cool, how did you make those?

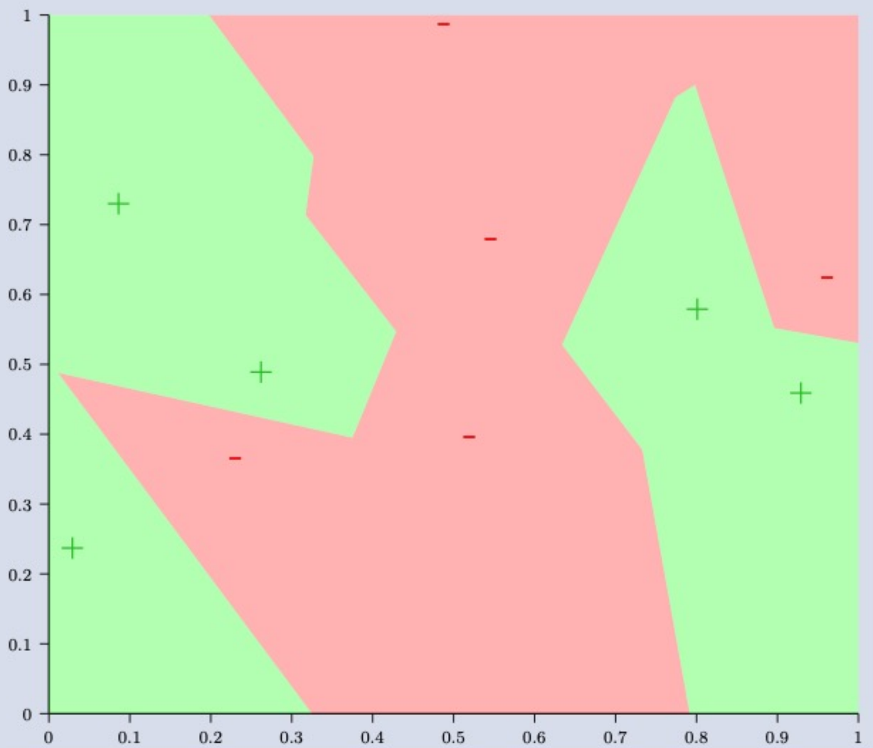
A: Well it’s a little complicated for k > 1, but here’s a way you can think about decision boundaries for a nearest neighbor hypothesis (k=1)



Q&A

Q: Those decision boundary figures for KNN were really cool, how did you make those?

A: Well it's a little complicated for k > 1, but here's a way you can think about decision boundaries for a nearest neighbor hypothesis (k=1)



Reminders

- Homework 2: Decision Trees
  - Out: Wed, Jan. 26
  - Due: Fri, Feb. 4 at 11:59pm
- HW1 Resubmission:
  - You should only resubmit if you receive email from us inviting you to resubmit.
- Homework 3: KNN, Perceptron, Lin.Reg.
  - Out: Fri, Feb. 4
  - Due: Fri, Feb. 11 at 11:59pm
  - (only two grace/late days permitted)

GEOMETRY & VECTORS

Geometry

In-Class Exercise

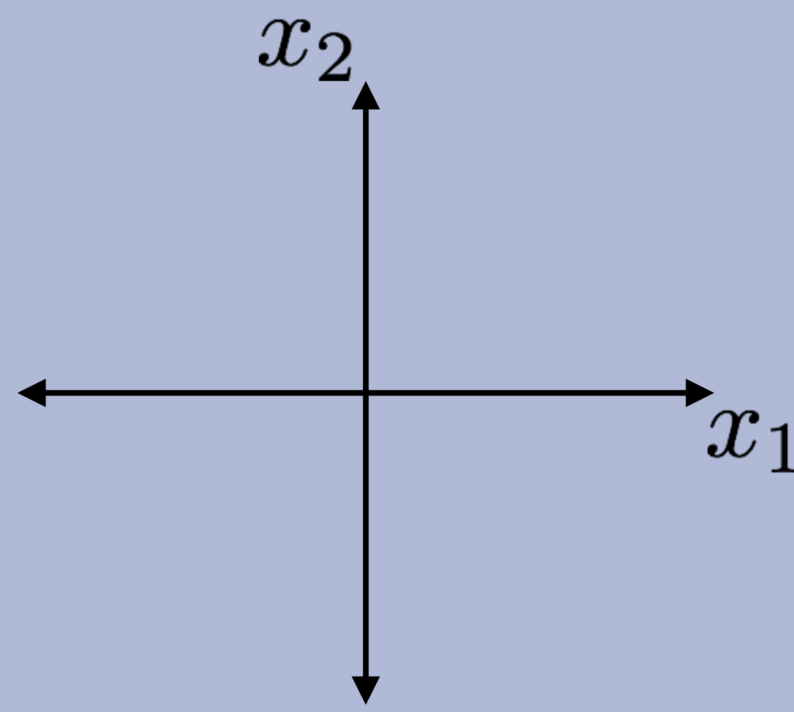
Draw a picture of the region corresponding to:

$$w_1 x_1 + w_2 x_2 + b > 0$$

where $w_1 = 2, w_2 = 3, b = 6$

Draw the vector
$\mathbf{w} = [w_1, w_2]$

Answer Here:



Visualizing Dot-Products

Whiteboard:

- definition of dot product
- definition of L2 norm
- definition of orthogonality

Vector Projection

Question:

*Which of the following is the projection of a vector **a** onto a vector **b**?*

A. $\frac{\mathbf{a}^T\mathbf{b}}{\mathbf{b}}\mathbf{a}$

B. $\frac{\mathbf{a}\cdot\mathbf{b}}{\mathbf{a}^T\mathbf{b}}$

C. $\frac{(\mathbf{a}^T\mathbf{b})}{||\mathbf{b}||_2}\mathbf{b}$

D. $\frac{(\mathbf{a}\cdot\mathbf{b})}{||\mathbf{b}||_2}\mathbf{b}$

E. $\frac{(\mathbf{a}^T\mathbf{b})}{||\mathbf{b}||_2^2}\mathbf{b}$

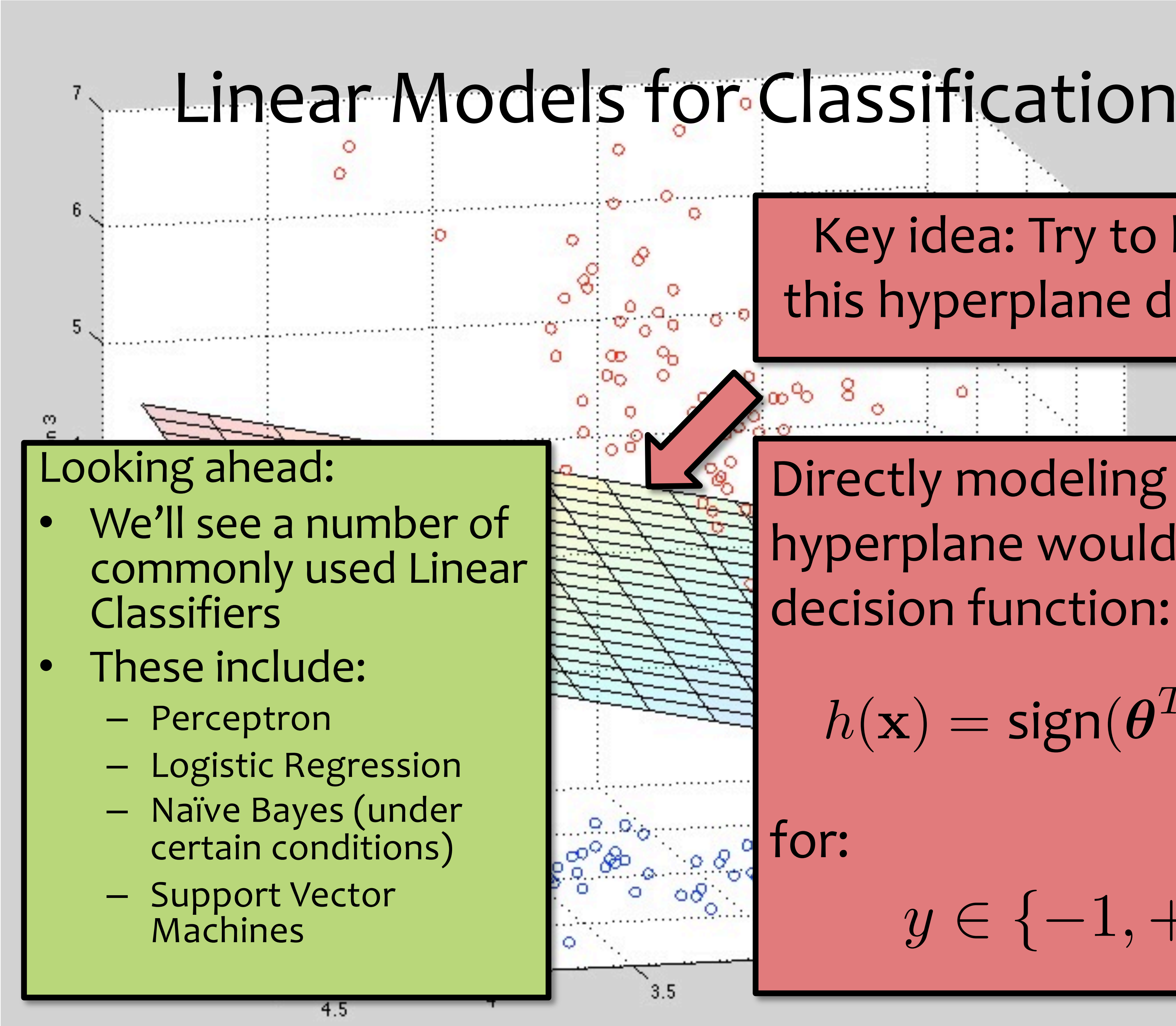
F. $\frac{(\mathbf{a}^T\mathbf{b})^2}{||\mathbf{b}||_2}\mathbf{b}$

Visualizing Dot-Products

Whiteboard:

- vector projection
- hyperplane definition
- half-space definitions

Linear Models for Classification



Key idea: Try to learn this hyperplane directly

Looking ahead:

- We'll see a number of commonly used Linear Classifiers
- These include:
  - Perceptron
  - Logistic Regression
  - Naïve Bayes (under certain conditions)
  - Support Vector Machines

Directly modeling the hyperplane would use a decision function:

$$h(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

for:

$$y \in \{-1, +1\}$$

ONLINE LEARNING

Online vs. Batch Learning

Batch Learning

Learn from all the examples at once

Online Learning

Gradually learn as each example is received

Online Learning

Examples

1. **Stock market** prediction (what will the value of Alphabet Inc. be tomorrow?)
2. **Email** classification (distribution of both spam and regular mail changes over time, but the target function stays fixed - last year's spam still looks like spam)
3. **Recommendation** systems. Examples: recommending movies; predicting whether a user will be interested in a new news article
4. **Ad placement** in a new market

Slide adapted from Nina Balcan

Online Learning

For i = 1, 2, 3, …**:**

- **Receive** an unlabeled instance $\mathbf{x}^{(i)}$
- **Predict** $y' = h_{\boldsymbol{\theta}}(\mathbf{x}^{(i)})$
- **Receive** true label $y^{(i)}$
- **Suffer loss** if a mistake was made, $y' \neq y^{(i)}$
- **Update** parameters $\boldsymbol{\theta}$

Goal:

- **Minimize** the number of **mistakes**

THE PERCEPTRON ALGORITHM

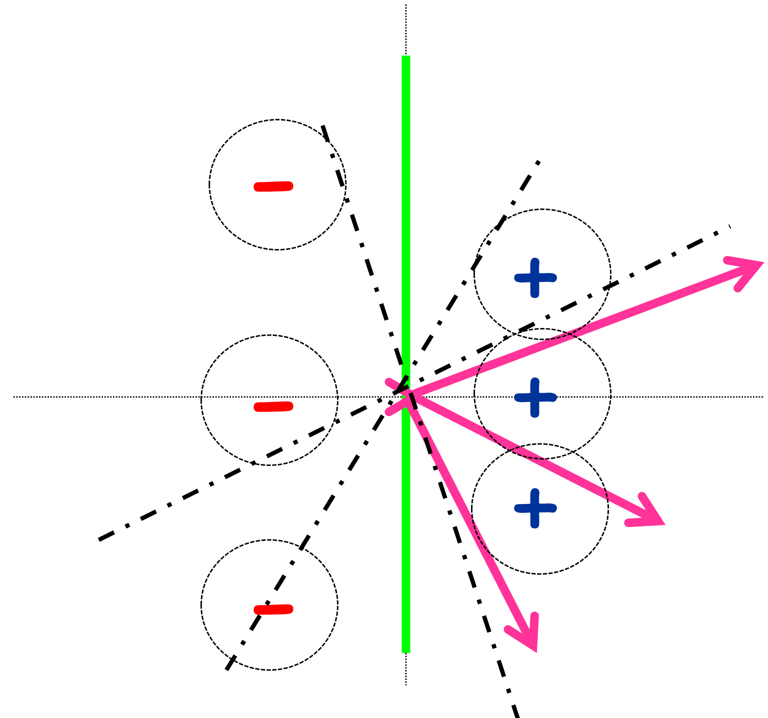
Perceptron

Whiteboard:

- (Online) Perceptron Algorithm
- Hypothesis class for Perceptron
- 2D Example of Perceptron

Perceptron Algorithm: Example

Example: $(-1,2)-$ ✗
$(1,0)+$ ✓
$(1,1)+$ ✗
$(-1,0)-$ ✓
$(-1,-2)-$ ✗
$(1,-1)+$ ✓



$w_1 = (0,0)$

$w_2 = w_1 - (-1,2) = (1,-2)$

$w_3 = w_2 + (1,1) = (2,-1)$

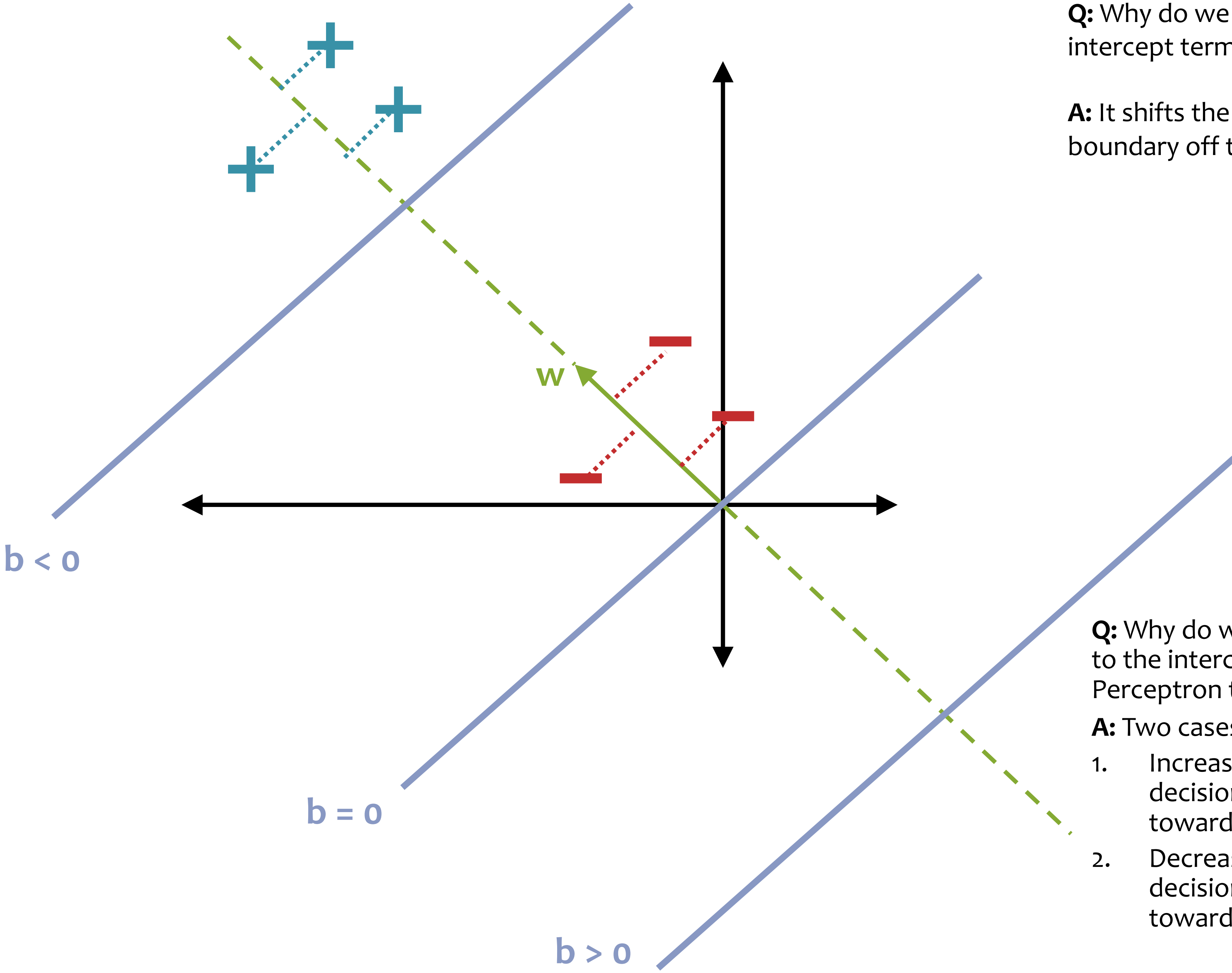
$w_4 = w_3 - (-1,-2) = (3,1)$

Perceptron Algorithm: (without the bias term)

- Set t=1, start with all-zeroes weight vector w_1.
- Given example x, predict positive iff $w_t \cdot x \geq 0$.
- On a mistake, update as follows:
  - Mistake on positive, update $w_{t+1} \leftarrow w_t + x$
  - Mistake on negative, update $w_{t+1} \leftarrow w_t - x$

Slide adapted from Nina Balcan

Intercept Term



Q: Why do we need an intercept term?

A: It shifts the decision boundary off the origin

Q: Why do we add / subtract 1.0 to the intercept term during Perceptron training?

A: Two cases

1. Increasing b shifts the decision boundary towards the negative side
2. Decreasing b shifts the decision boundary towards the positive side

Perceptron Inductive Bias

1. Decision boundary should be linear
2. Most recent mistakes are most important (and should be corrected)

Background: Hyperplanes

Notation Trick: fold the bias *b* and the weights ***w*** into a single vector **θ** by prepending a constant to **x** and increasing dimensionality by one to get **x'**!

Hyperplane (Definition 1):

$$\mathcal{H} = \{\mathbf{x} : \mathbf{w}^T \mathbf{x} = b\}$$

Hyperplane (Definition 2):

$$\mathcal{H} = \{\mathbf{x}' : \boldsymbol{\theta}^T \mathbf{x}' = 0 \text{ and } x'_1 = 1\}$$

$$\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$$

Half-spaces:

$$\mathcal{H}^+ = \{\mathbf{x} : \boldsymbol{\theta}^T \mathbf{x} > 0 \text{ and } x_1 = 1\}$$

$$\mathcal{H}^- = \{\mathbf{x} : \boldsymbol{\theta}^T \mathbf{x} < 0 \text{ and } x_1 = 1\}$$

(Online) Perceptron Algorithm

Data: Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

Prediction: Output determined by hyperplane.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$ and $x_1 = 1$

Learning: Iterative procedure:

- **initialize** parameters to vector of all zeroes
- **while** not converged
  - **receive** next example ($\mathbf{x}^{(i)}$, $y^{(i)}$)
  - **predict** $y' = h(\mathbf{x}^{(i)})$
  - **if** positive mistake: **add** $\mathbf{x}^{(i)}$ to parameters
  - **if** negative mistake: **subtract** $\mathbf{x}^{(i)}$ from parameters

(Online) Perceptron Algorithm

Data: Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

Prediction: Output determined by hyperplane.

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

$$\text{sign}(a) = \begin{cases} 1, & \text{if } a \geq 0 \\ -1, & \text{otherwise} \end{cases}$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]^T$ and $x_1 = 1$

Learning:

Algorithm 1 Perceptron Learning Algorithm (Online)

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots\}$)
2: $\boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
3: **for** $i \in \{1, 2, \ldots\}$ **do** ▷ For each example
4: $\hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
5: **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
6: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
7: **return** $\boldsymbol{\theta}$

(Online) Perceptron Algorithm

Data: Inputs are continuous vectors of length *M*. Outputs are discrete.

$$(\mathbf{x}^{(1)}, y^{(1)}), (\mathbf{x}^{(2)}, y^{(2)}), \ldots$$

where $\mathbf{x} \in \mathbb{R}^M$ and $y \in \{+1, -1\}$

Prediction: Output determine

$$\hat{y} = h_{\boldsymbol{\theta}}(\mathbf{x}) = \text{sign}(\boldsymbol{\theta}^T \mathbf{x})$$

Assume $\boldsymbol{\theta} = [b, w_1, \ldots, w_M]$

Implementation Trick: same behavior as our "*add on positive mistake and subtract on negative mistake*" version, because $y^{(i)}$ takes care of the sign

Learning:

Algorithm 1 Perceptron Learning Alg

```
procedure PERCEPTRON(𝒟 = {(x
    θ ← 0                                    ▷ Initialize parameters
    for i ∈ {1, 2, ...} do                   ▷ For each example
        ŷ ← sign(θ^T x^(i))                  ▷ Predict
        if ŷ ≠ y^(i) then                    ▷ If mistake
            θ ← θ + y^(i) x^(i)              ▷ Update parameters
    return θ
```

(Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

Algorithm 1 Perceptron Learning Algorithm (Batch)

1: **procedure** PERCEPTRON($\mathcal{D} = \{(\mathbf{x}^{(1)}, y^{(1)}), \ldots, (\mathbf{x}^{(N)}, y^{(N)})\}$)
2: $\quad \boldsymbol{\theta} \leftarrow \mathbf{0}$ ▷ Initialize parameters
3: $\quad$ **while** not converged **do**
4: $\quad\quad$ **for** $i \in \{1, 2, \ldots, N\}$ **do** ▷ For each example
5: $\quad\quad\quad \hat{y} \leftarrow \text{sign}(\boldsymbol{\theta}^T \mathbf{x}^{(i)})$ ▷ Predict
6: $\quad\quad\quad$ **if** $\hat{y} \neq y^{(i)}$ **then** ▷ If mistake
7: $\quad\quad\quad\quad \boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + y^{(i)} \mathbf{x}^{(i)}$ ▷ Update parameters
8: $\quad$ **return** $\boldsymbol{\theta}$

(Batch) Perceptron Algorithm

Learning for Perceptron also works if we have a fixed training dataset, D. We call this the "batch" setting in contrast to the "online" setting that we've discussed so far.

Discussion:

The Batch Perceptron Algorithm can be derived in two ways.

1. By extending the online Perceptron algorithm to the batch setting (as mentioned above)
2. By applying **Stochastic Gradient Descent (SGD)** to minimize a so-called **Hinge Loss** on a linear separator

Extensions of Perceptron

- **Voted Perceptron**
  - generalizes better than (standard) perceptron
  - memory intensive (keeps around every weight vector seen during training, so each one can vote)
- **Averaged Perceptron**
  - empirically similar performance to voted perceptron
  - can be implemented in a memory efficient way (running averages are efficient)
- **Kernel Perceptron**
  - Choose a kernel K(x’, x)
  - Apply the **kernel trick** to Perceptron
  - Resulting algorithm is **still very simple**
- **Structured Perceptron**
  - Basic idea can also be applied when **y** ranges over an exponentially large set
  - Mistake bound **does not** depend on the size of that set

Perceptron Exercises

Question:

The parameter vector **w** *learned by the Perceptron algorithm can be* ***written as a linear combination*** *of the feature vectors* $\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \ldots, \mathbf{x}^{(N)}$.

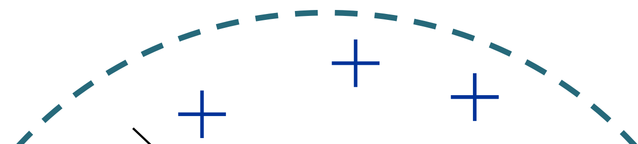
A. *True, if you replace "linear" with "polynomial" above*
B. *True, for all datasets*
C. *False, for all datasets*
D. *True, but only for certain datasets*
E. *False, but only for certain datasets*

PERCEPTRON MISTAKE BOUND

Perceptron Mistake Bound

Guarantee: if some data has margin γ and all points lie inside a ball of radius R, then the online Perceptron algorithm makes $\leq (R/\gamma)^2$ mistakes

(Normalized margin: multiplying all points by 100, or dividing all points by 100, doesn't change the number of mistakes! The algorithm is invariant to scaling.)



Def: We say that the (batch) perceptron algorithm has **converged** if it stops making mistakes on the training data (perfectly classifies the training data).

Main Takeaway: For **linearly separable** data, if the perceptron algorithm cycles repeatedly through the data, it will **converge** in a finite # of steps.