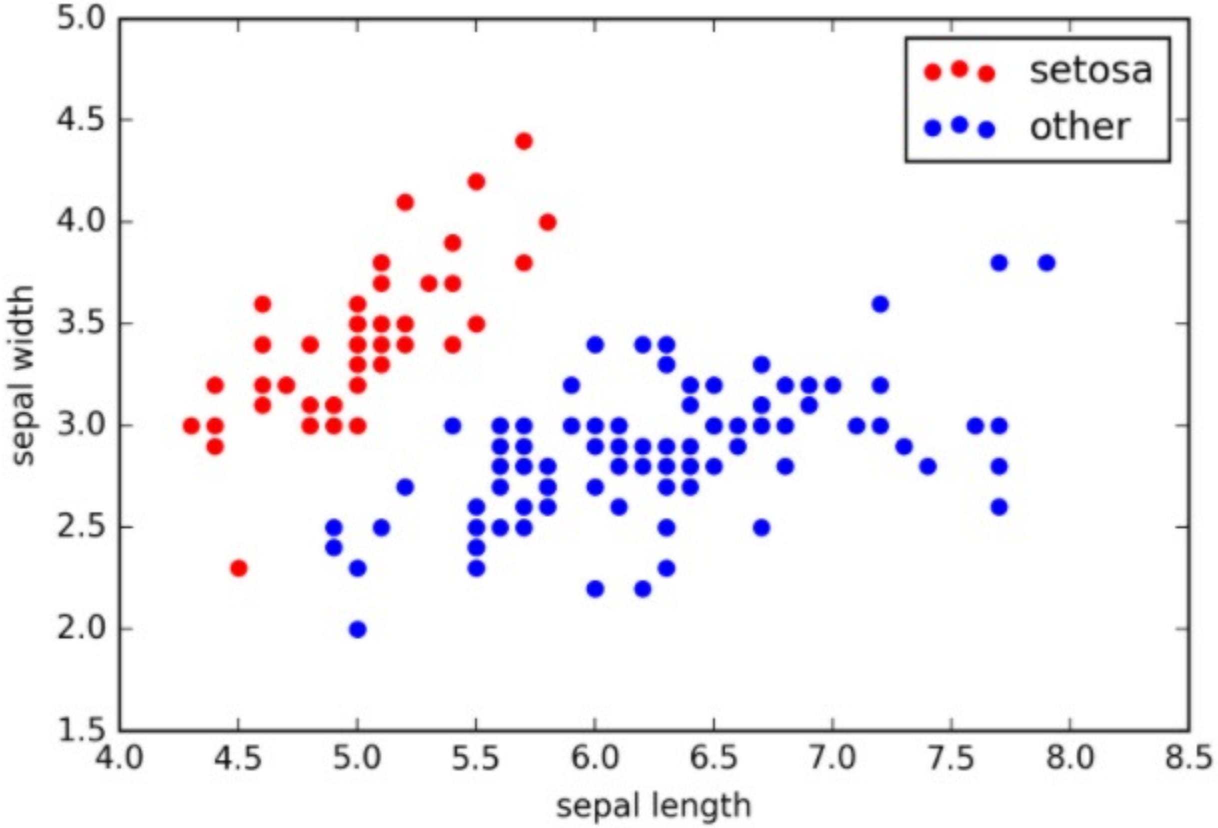
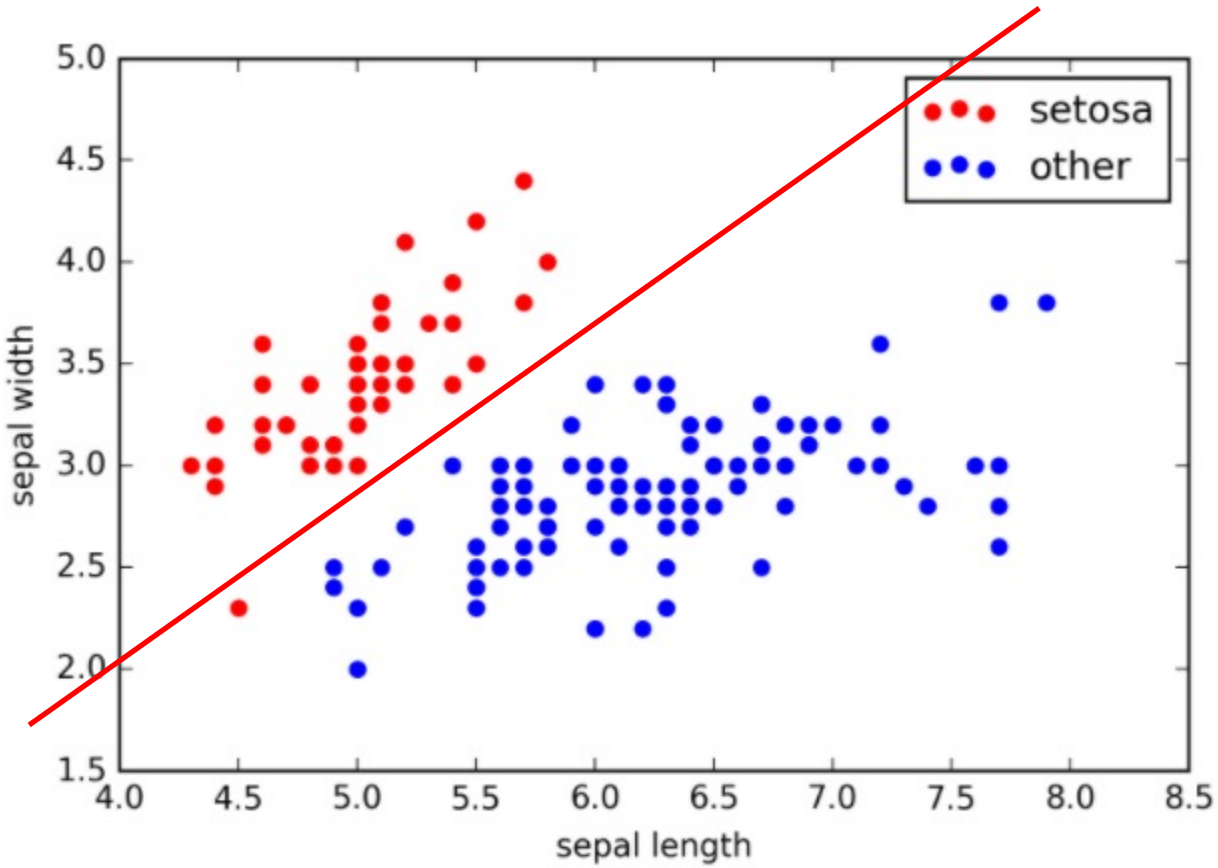
ROAD MAP

- So far
  - kNN, Decision tree, Linear regression
  - Bias and variance tradeoff, Model assessment and selection, Feature selection
- Next:
  - **Linear classifiers: perceptron, logistic regression**, Support Vector Machines
  - Generative classifiers: Naïve bayes classifier
  - Unsupervised learning: dimension reduction, k-means clustering
  - Ensemble methods: bagging, boosting, ensembles
  - Neural networks

HOW TO CLASSIFY?



HOW TO CLASSIFY?



BINARY LINEAR CLASSIFICATION MODEL

- A classifier is a hyperplane
- It's a line in two-dimensional space

$$\beta_0 + \beta_1 X_1 + \beta_2 X_2 = 0$$

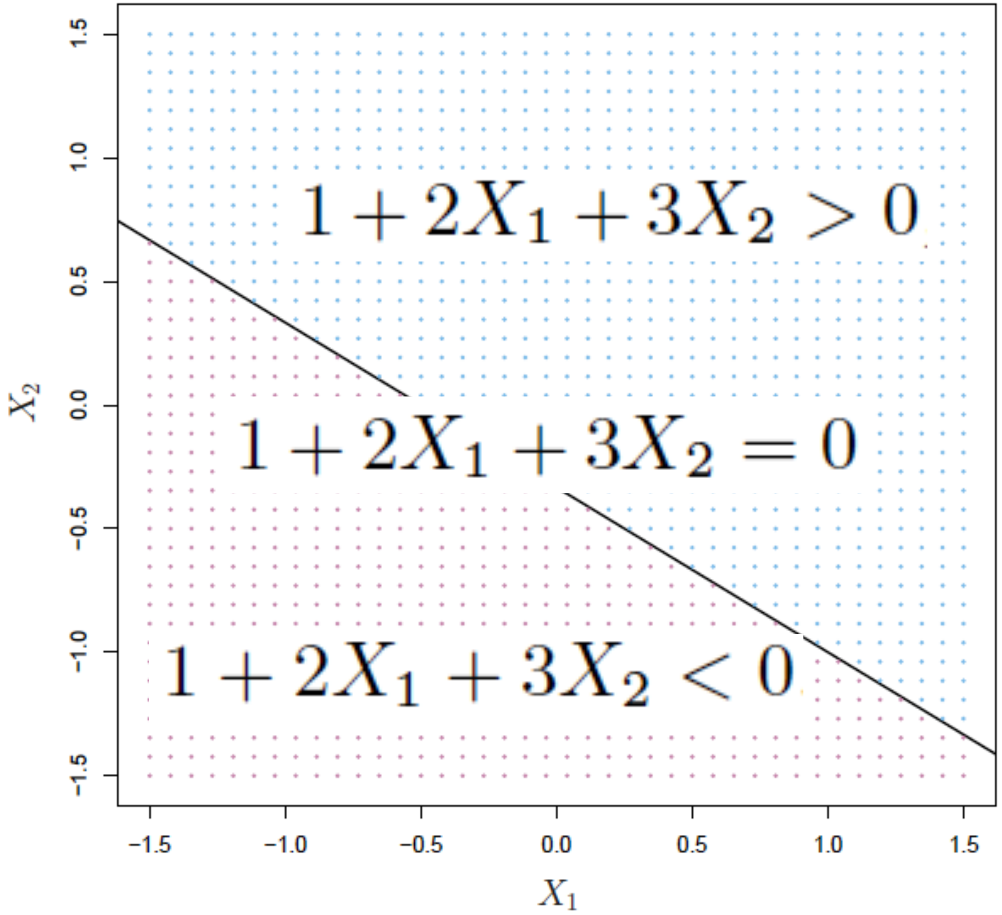


Figure 9.1 (James et al.)

HYPERPLANE

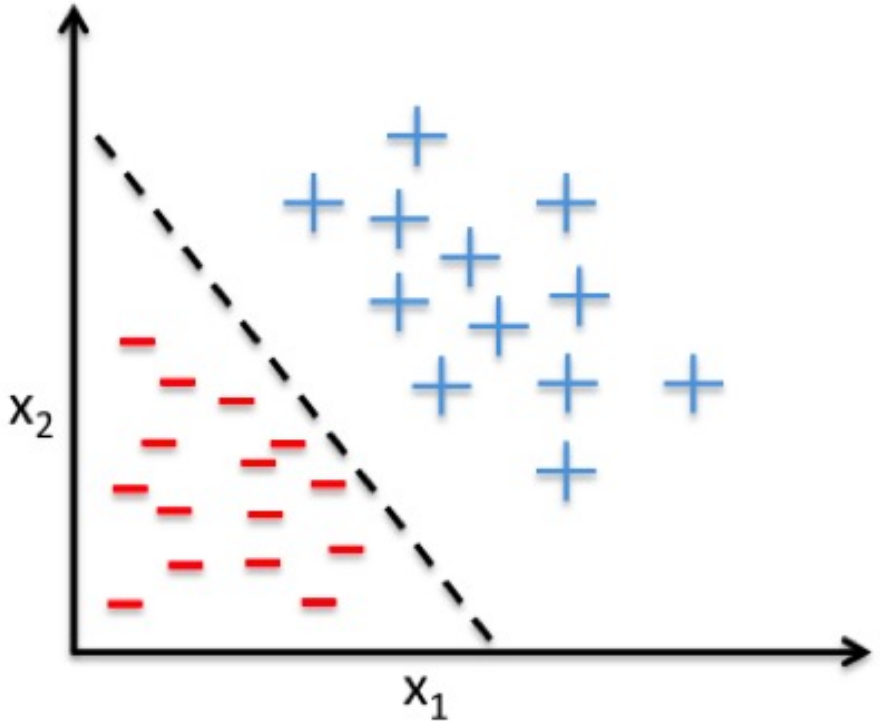
- General equation for a hyperplane in p dimensional space

$$\beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_p x_p = 0$$

- Hyperplane as classifier

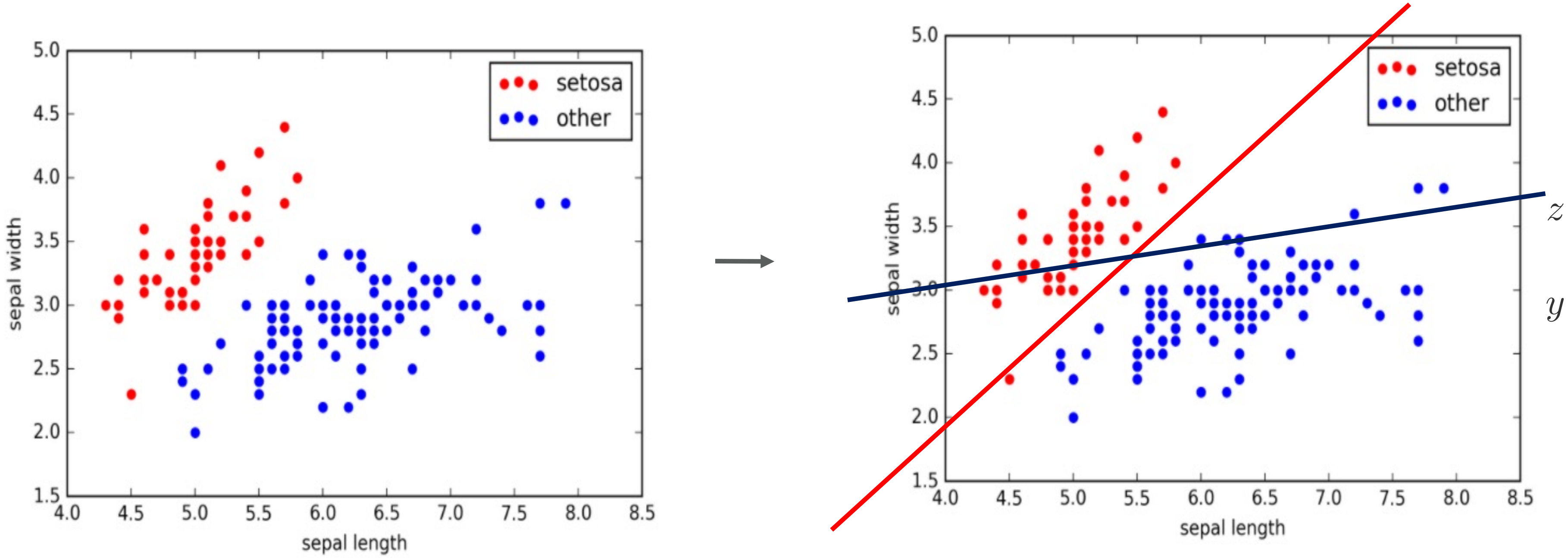
$$z = \mathbf{x}\boldsymbol{\beta}$$

$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$



GROUP ACTIVITY

BINARY LINEAR CLASSIFIER: TRAINING



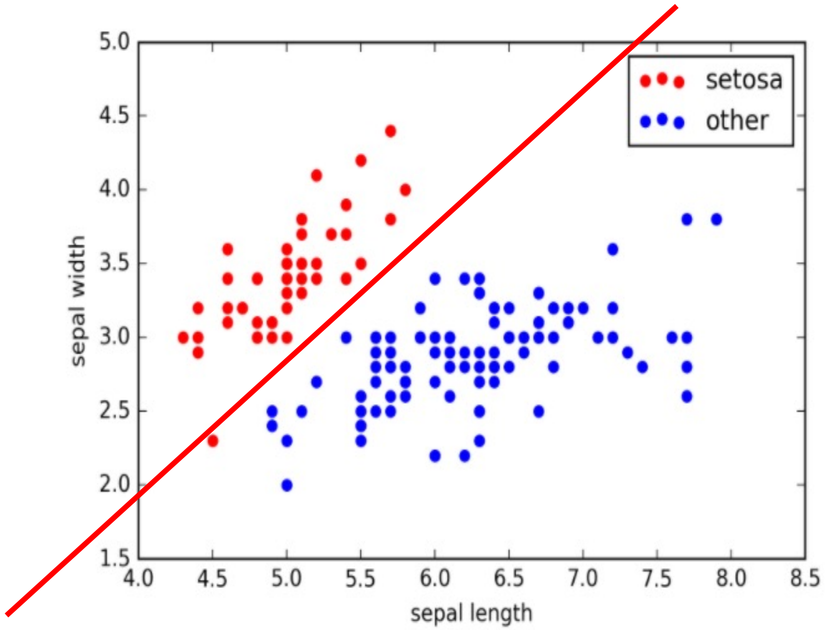
$$z = \mathbf{x}\boldsymbol{\beta}$$

$$y = \begin{cases} 1, & \text{if } z \geq 0 \\ 0, & \text{if } z < 0 \end{cases}$$

Which one is better? Why? How to learn the “best” line?

LINEAR CLASSIFIERS

- Perceptron (minimize 0-1 loss)
- Logistic regression (minimize cross-entropy loss)
- Support Vector Machines (maximize margin)



PERCEPTRON

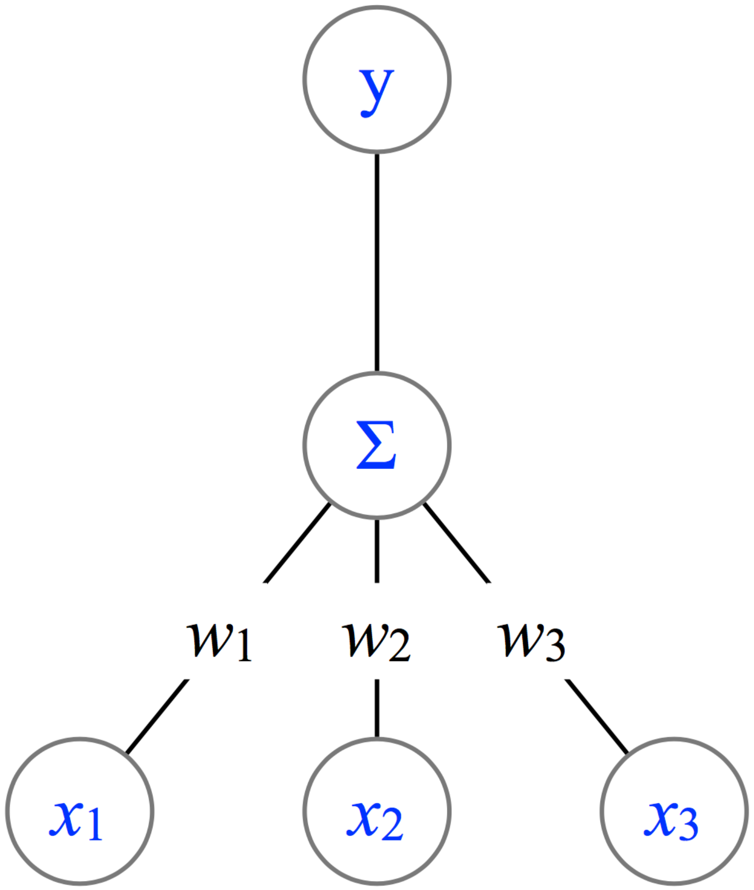
CS 334: Machine Learning

PERCEPTRON [ROSENBLATT, 1957]

- Uses hyperplane classifier to map input to binary output
- Compute linear combination of the inputs and threshold it

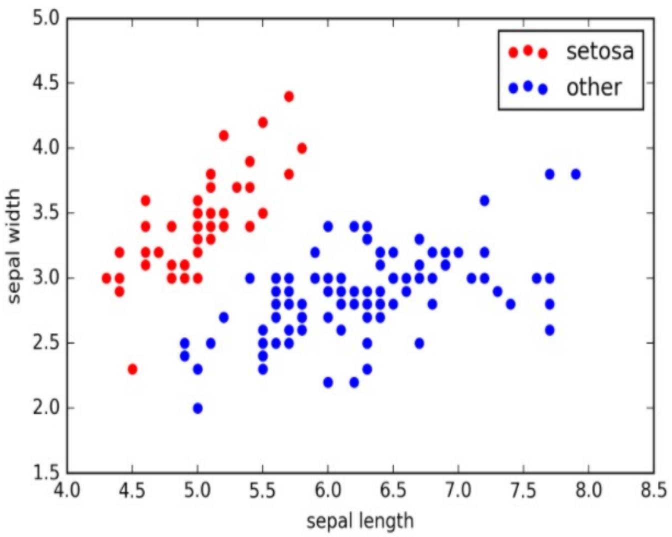
$$f_{\mathbf{w}}(\mathbf{x}) = \mathrm{sign}(\mathbf{x} \cdot \mathbf{w})$$
$$= \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

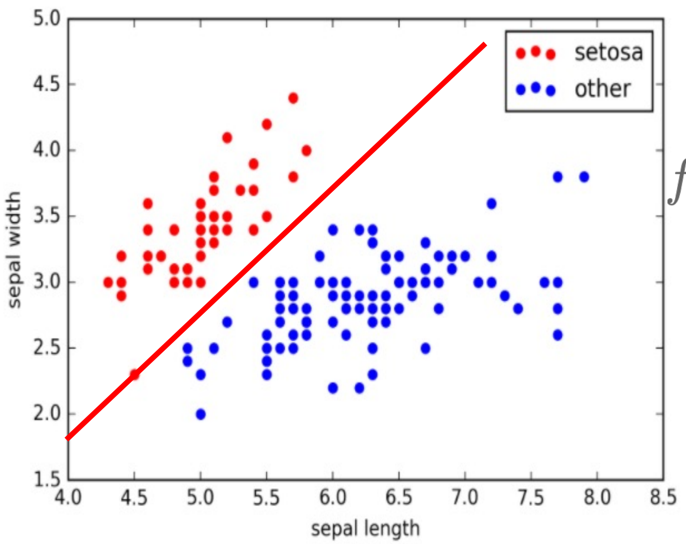
Assume threshold is set to 0 — simulate nonzero threshold using a dummy input feature that is always 1



PERCEPTRON

- Learning:



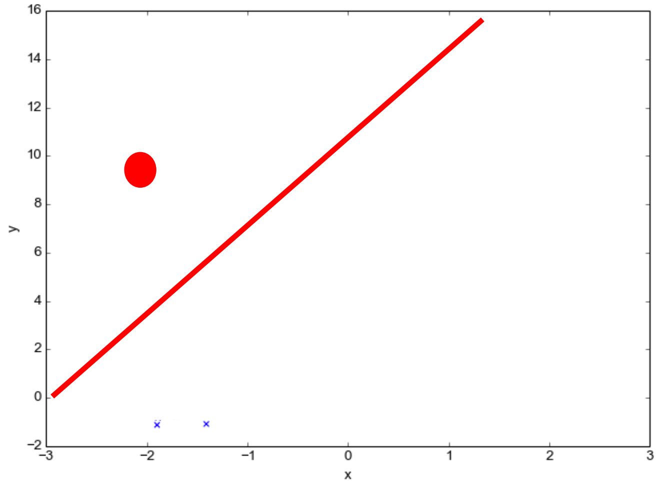


find *w*

$$f_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w})$$
$$= \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

- Prediction:

x

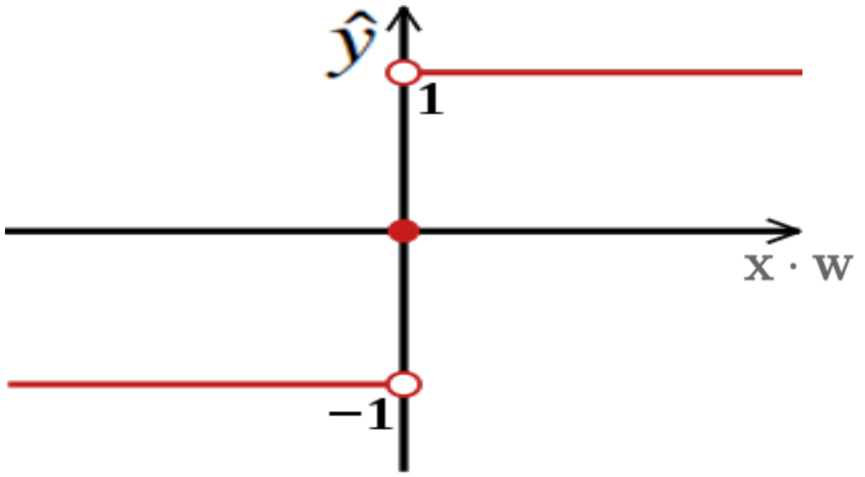


compute *fw(x)*

$$f_{\mathbf{w}}(\mathbf{x}) = \text{sign}(\mathbf{x} \cdot \mathbf{w})$$
$$= \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$

PERCEPTRON: LEARNING

$$f_{\mathbf{w}}(\mathbf{x}) = \mathrm{sign}(\mathbf{x} \cdot \mathbf{w})$$
$$= \begin{cases} +1 & \text{if } \mathbf{x} \cdot \mathbf{w} > 0 \\ -1 & \text{otherwise} \end{cases}$$



- Find weights that minimizes classification error (0-1 loss)
  - Penalty 1 if prediction is incorrect
  - Penalty 0 if prediction is correct

$$\min \sum_{i=1}^{n} \max(0, -y_i \hat{y}_i)$$

PERCEPTRON: LEARNING ALGORITHM

- For each epoch (stop at max epoch or no mistakes)
  - For each point:
    - Predict +1 iff $\mathbf{w} \cdot \mathbf{x}_i \geq 0$
    - If successfully classified, do nothing
    - If mistake, update as follows:
      - Mistake on positive (yi=1) $\mathbf{w}^+ = \mathbf{w} + \mathbf{x}_i$
      - Mistake on negative (yi = -1) $\mathbf{w}^+ = \mathbf{w} - \mathbf{x}_i$

Why does this work?
What does this look like?

PERCEPTRON: LEARNING ALGORITHM

- Perceptron uses SGD to learn the parameters
- Minimizes 0-1 loss

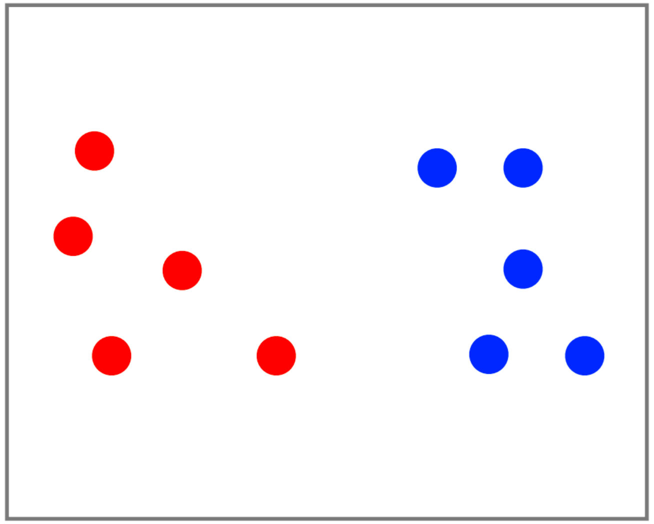
$$\min \sum_{i=1}^{n} \max(0, -y_i \hat{y}_i)$$

- Gradient update (without loss of generality, can set learning rate to be 1)

$$\mathbf{w}^+ = \mathbf{w} + \mathbf{x}_i y_i$$

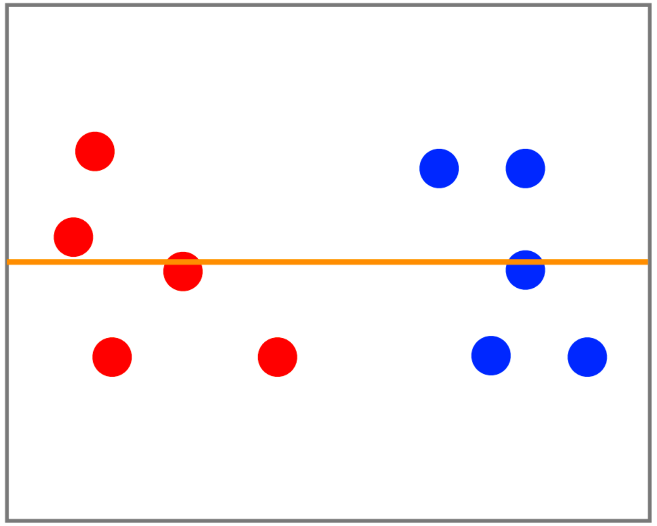
- (Connection to SGD was determined much later!)

PERCEPTRON: LEARNING EXAMPLE

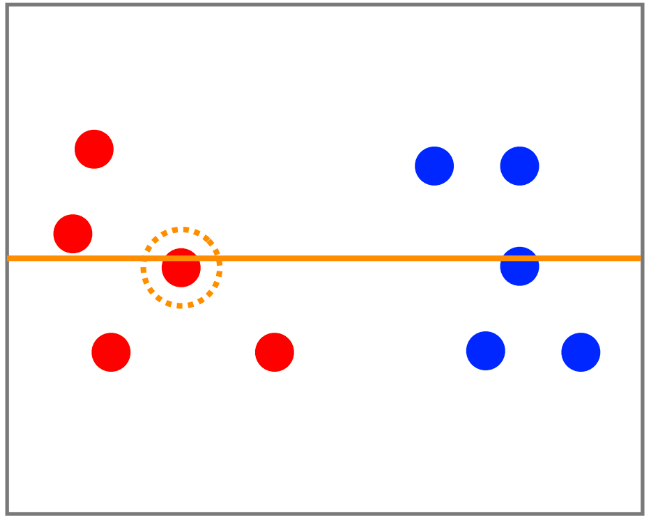


Training data

Initialize parameters

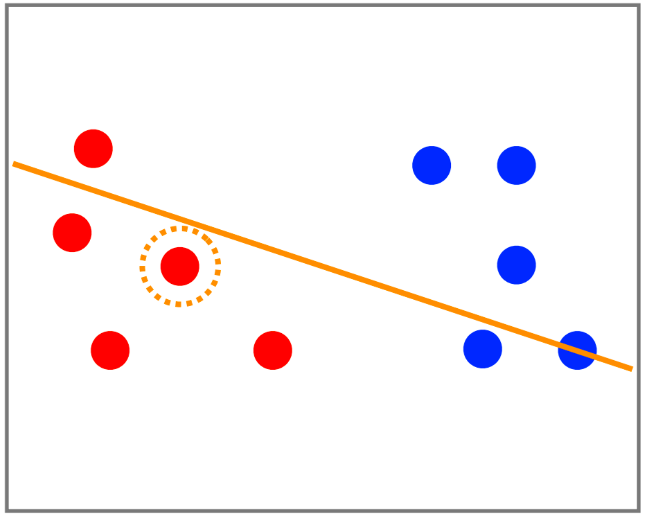


PERCEPTRON: LEARNING EXAMPLE

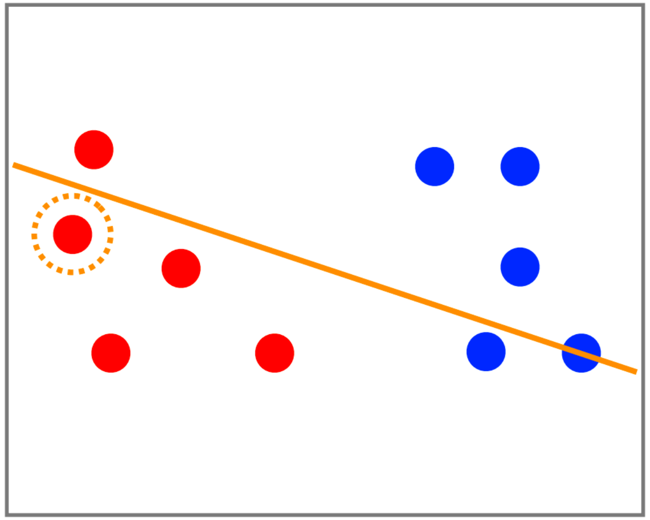


Randomly select point
— incorrectly classified

Update parameters

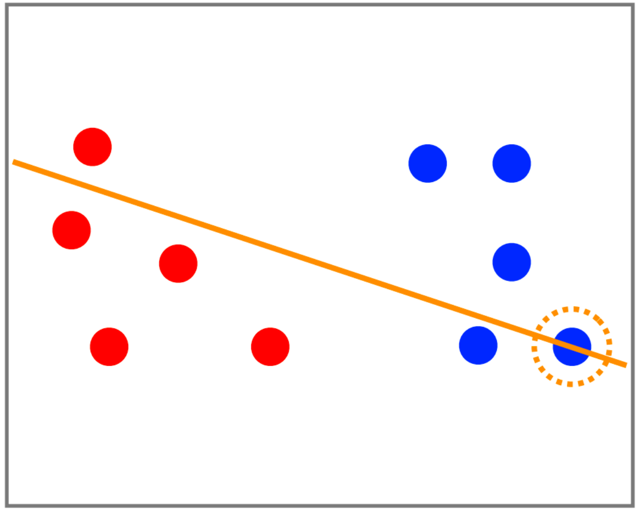


PERCEPTRON: LEARNING EXAMPLE

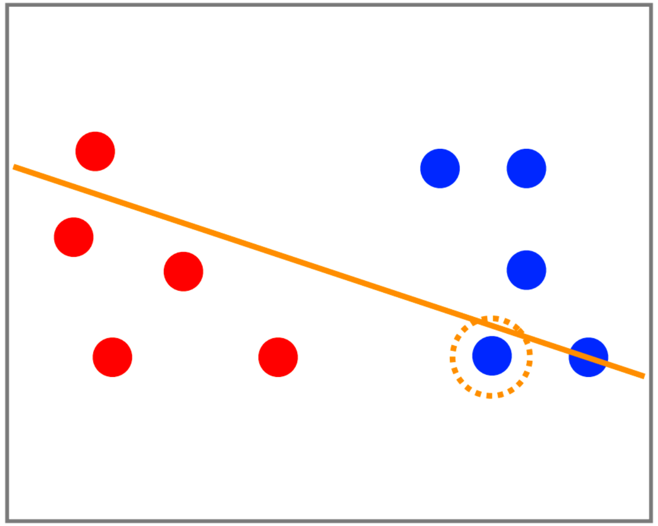


Randomly select another point — correct classified do nothing

Randomly select another point — correct classified do nothing

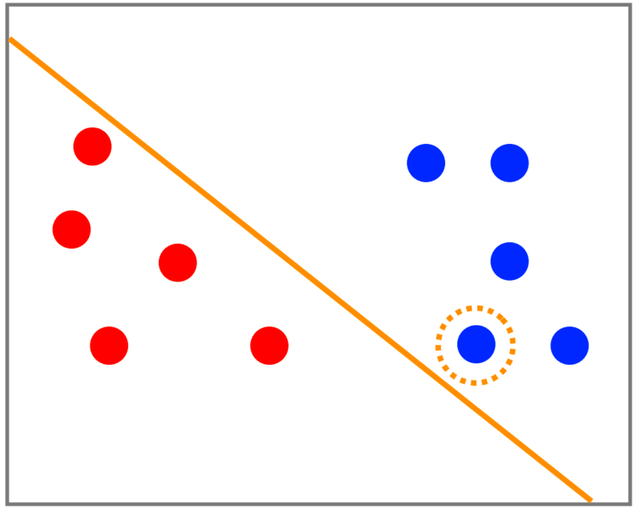


PERCEPTRON: LEARNING EXAMPLE



Randomly select another point —incorrectly classified

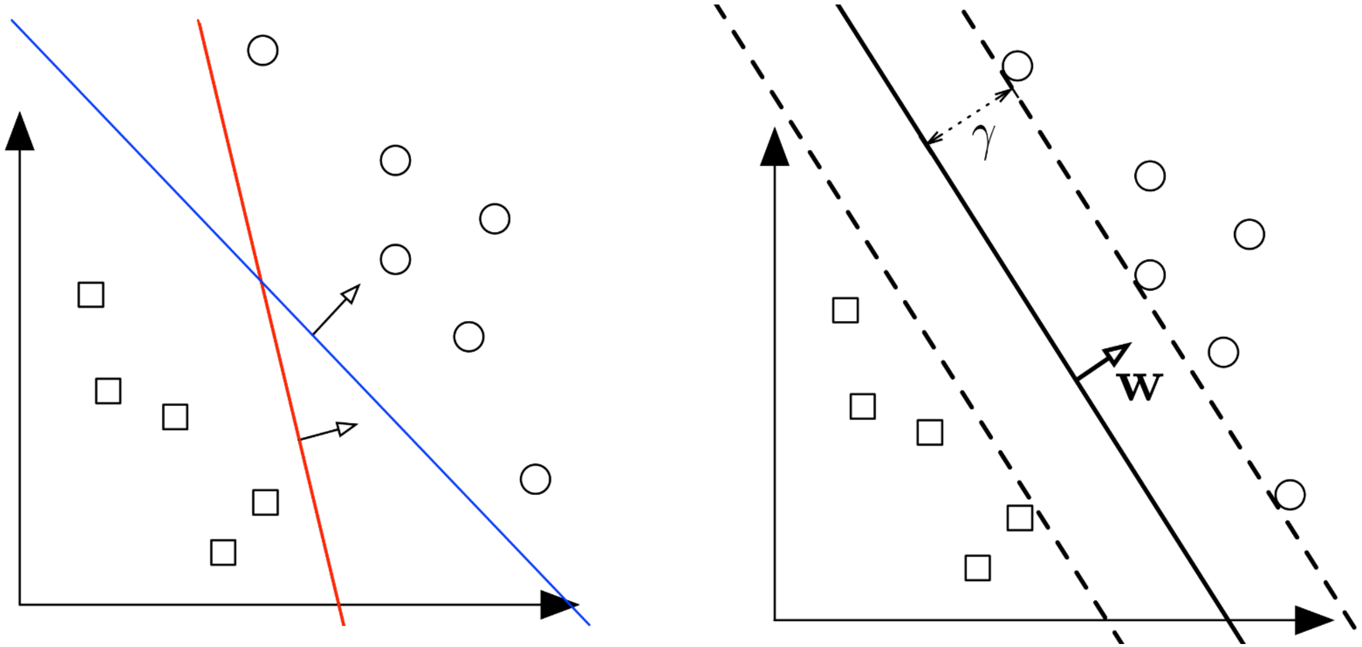
Update parameters



PERCEPTRON CONVERGENCE THEOREM

- Intuition: perceptron will converge more quickly for easy learning problems compared to hard problems

- Classify "easy" and "hard" via the margin



$$margin(\mathbf{D}, w, b) = \begin{cases} \min_{(x,y)\in\mathbf{D}} y(w \cdot x + b) & \text{if } w \text{ separates } \mathbf{D} \\ -\infty & \text{otherwise} \end{cases}$$

$$margin(\mathbf{D}) = \sup_{w,b} margin(\mathbf{D}, w, b)$$

PERCEPTRON CONVERGENCE THEOREM

Theorem. *Suppose the perceptron algorithm is run on a linearly separable data set* $\mathbf{D}$ *with margin* $\gamma > 0$. *Assume that* $||x|| \leq 1$ *for all* $x \in \mathbf{D}$. *Then the algorithm will converge after at most* $\frac{1}{\gamma^2}$ *updates.*

PERCEPTRON: ISSUES

- If data isn't linearly separable, no guarantees of convergence or training accuracy
- Even if training data is linearly separable, perceptron can overfit
- Averaged perceptron (average weight vectors across all iterations) is an algorithmic modification that helps both issues
- Other linear lassifiers
  - Support vector machine maximizes the margin
  - Logistic regression minimizes cross-entropy loss

FROM LINEAR TO NONLINEAR

- Feature mapping (kernel methods)
- Multiple perceptrons (neural networks)