

DS 4400

Machine Learning and Data Mining I  
Spring 2021

Alina Oprea  
Associate Professor  
Khoury College of Computer Science  
Northeastern University

March 11 2021

# Outline

- Boosting
  - AdaBoost
  - Properties of boosting
  - Bagging vs Boosting
- Review of linear models
  - Separating hyperplanes
- Support Vector Machines
  - Linearly separable data
    - Maximum margin classifier
  - Non-linearly separable data
    - Support vector classifier

# Ensemble Learning

Consider a set of classifiers  $h_1, \dots, h_L$

**Idea:** construct a classifier  $H(\mathbf{x})$  that combines the individual decisions of  $h_1, \dots, h_L$

- e.g., could have the member classifiers vote, or
- e.g., could use different members for different regions of the instance space

Successful ensembles require **diversity**

- Classifiers should make different mistakes
- Can have different types of base learners

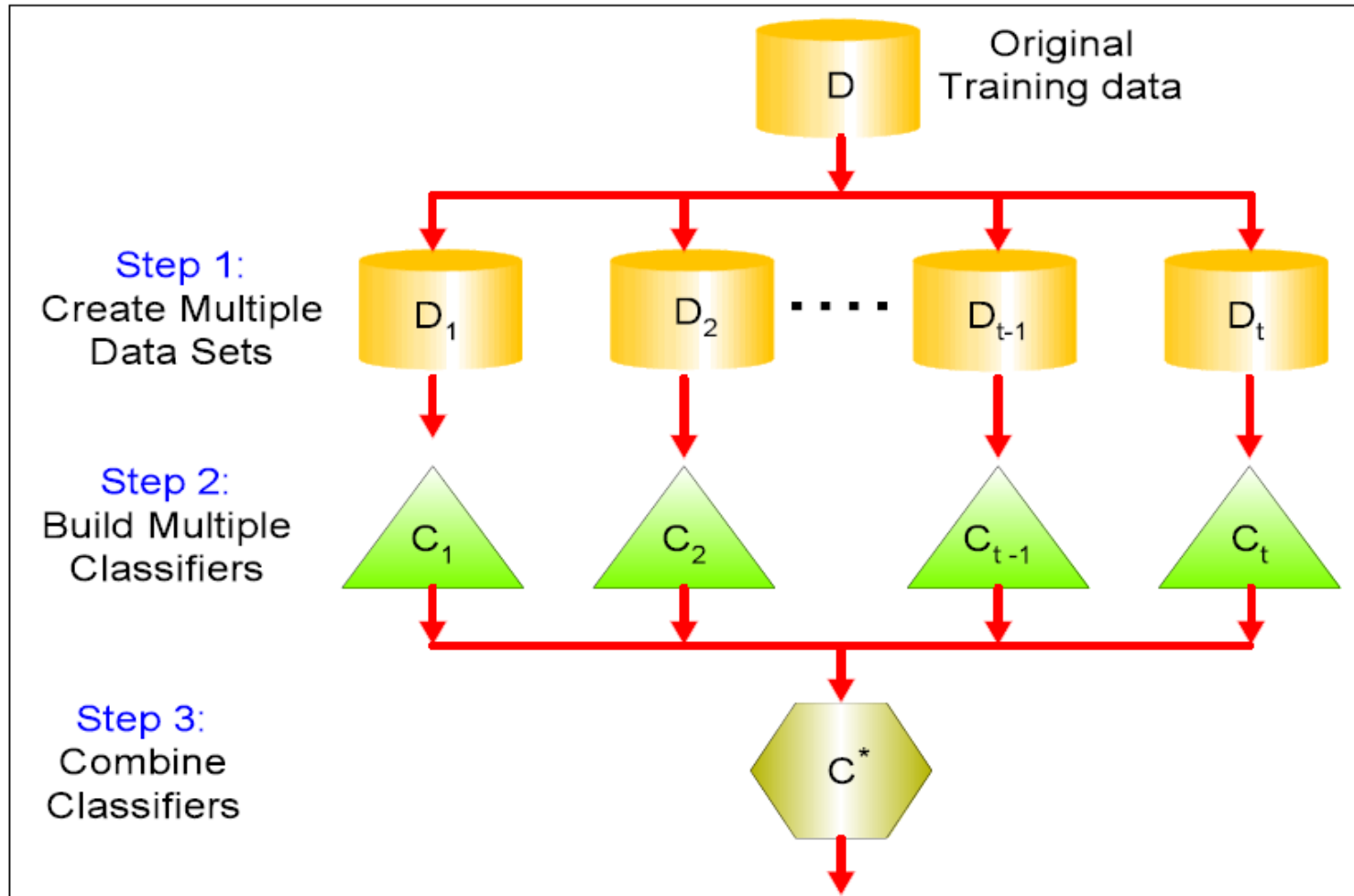
# How to Achieve Diversity

- Avoid overfitting
  - Vary the training data
- Features are noisy
  - Vary the set of features

Two main ensemble learning methods

- **Bagging** (e.g., Random Forests)
- **Boosting** (e.g., AdaBoost)

# Bagging



Majority Votes

# Overview of AdaBoost

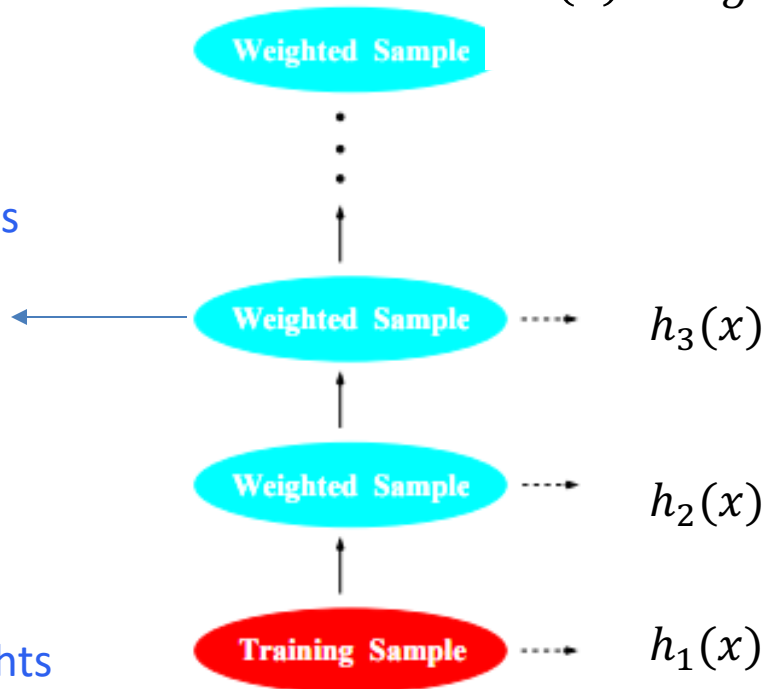
$$G(x) = \text{sign}\left(\sum_{i=1}^T \beta_i h_i(x)\right)$$

↓

Better classifiers will get  
higher weights

- Mis-classified examples get higher weights
- Correct examples get lower weights

Uniform weights

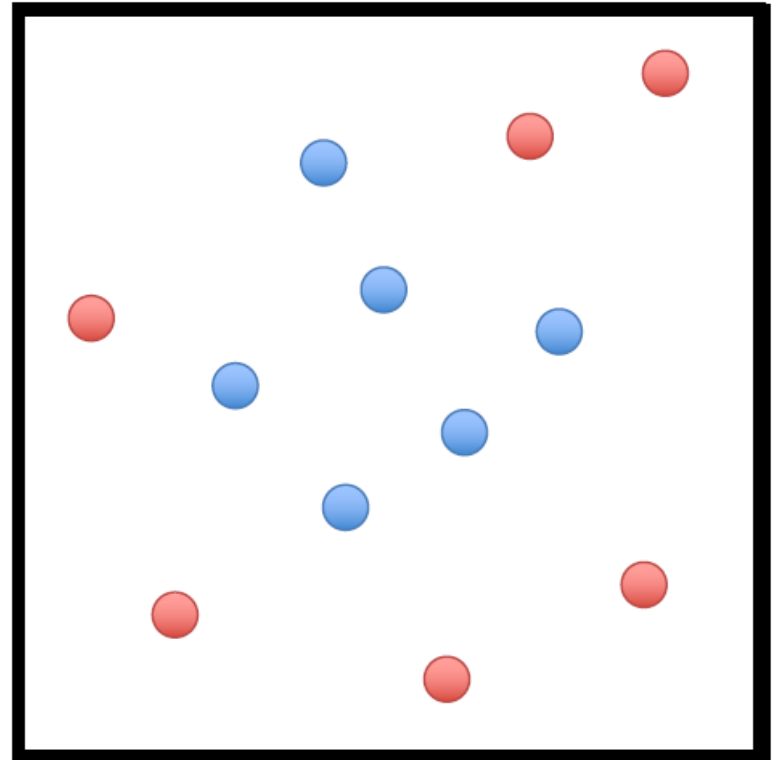


**FIGURE 10.1.** Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



- Size of point represents the instance's weight

# AdaBoost

$$y_i \in \{-1, 1\}$$

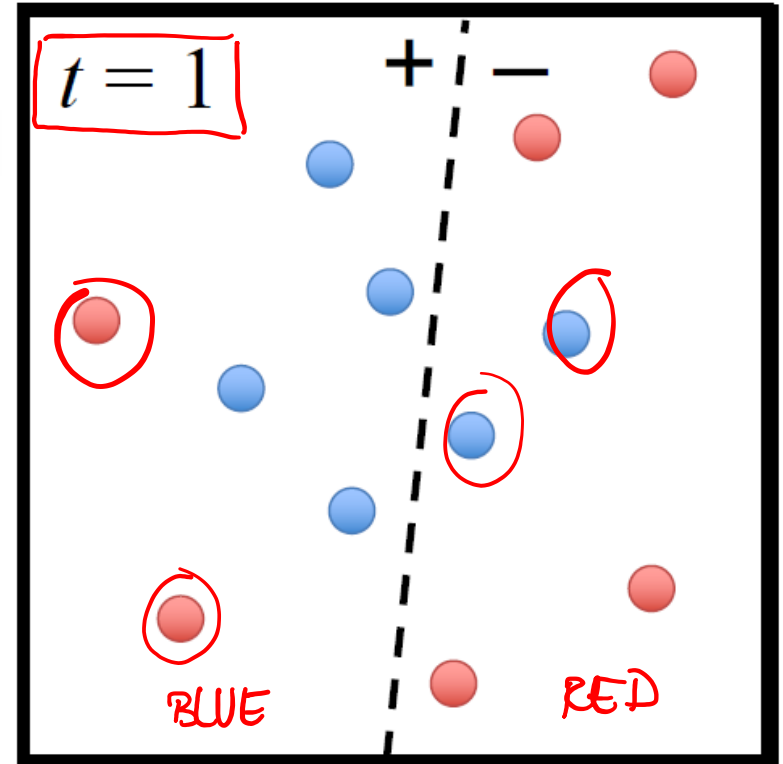
- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

1) If  $\epsilon_t > \frac{1}{2}$  STOP

2) If  $\epsilon_t \leq \frac{1}{2} \Rightarrow \frac{1 - \epsilon_t}{\epsilon_t} > 1 \Rightarrow \beta_t > 0$

$$\beta_t = \frac{1}{2} \ln \left( \frac{1}{\epsilon_t} - 1 \right)$$



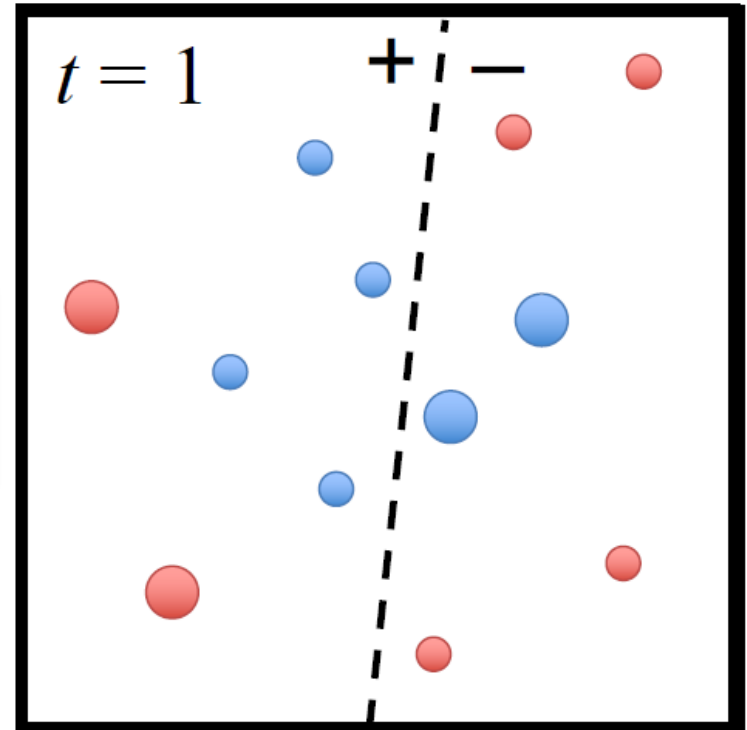
$$\epsilon_1 = \frac{4}{12} = \frac{1}{3}$$



# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:
- 7:    $\rightarrow w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



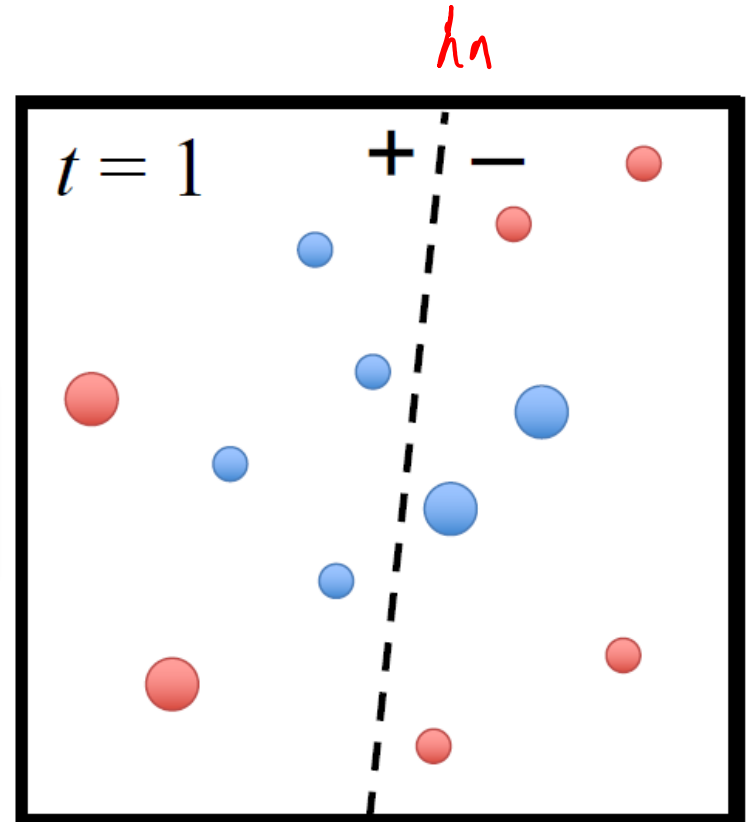
- 1) If  $x_i$  IS CORRECTLY CLASSIFIED  
 $\Rightarrow y_i \cdot h_t(x_i) = 1 \Rightarrow w_{t+1,i} = w_{t,i} \cdot \underbrace{e^{-\beta_t}}_{< 1} < w_{t,i}$
- 2) If  $x_i$  IS INCORRECTLY CLASSIFIED  
 $\Rightarrow y_i \cdot h_t(x_i) = -1 \Rightarrow w_{t+1,i} = w_{t,i} \cdot \underbrace{e^{\beta_t}}_{> 1} > w_{t,i}$

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

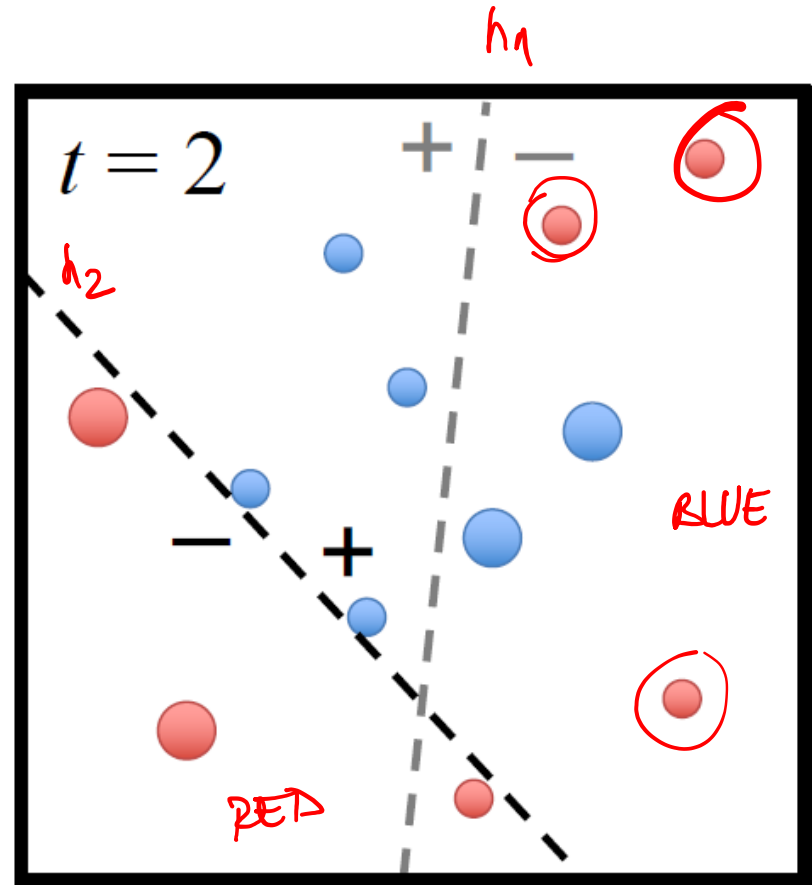
$$\sum_{i=1}^N w_{t+1,i} = 1$$



# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3: Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4: Compute the weighted training error of  $h_t$
- 5: Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6: Update all instance weights:  
 $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7: Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$



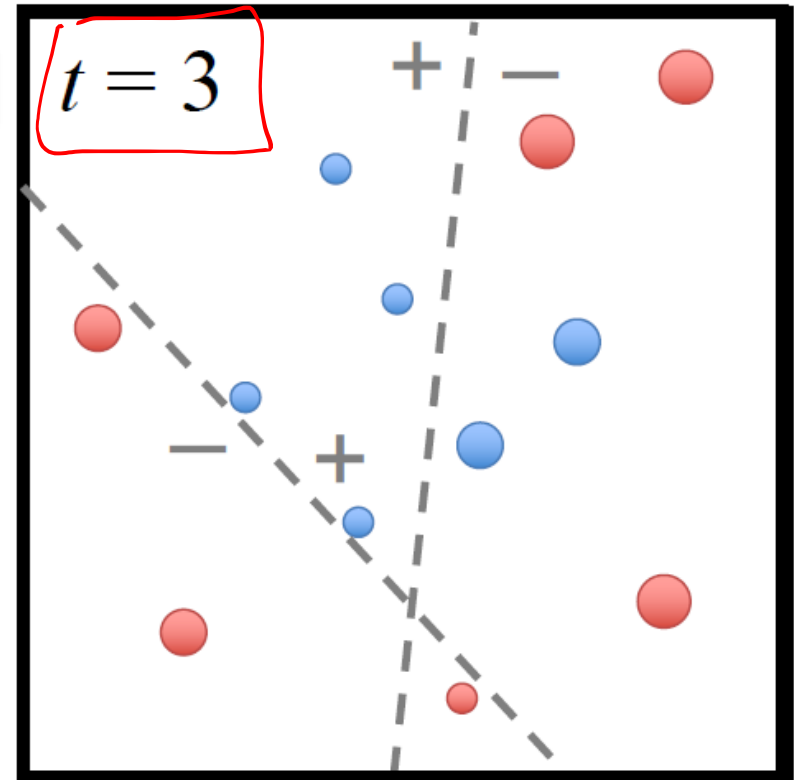
$$\epsilon_2 = \frac{3}{12} = \frac{1}{4}$$

$$\beta_2 = \frac{1}{2} \ln \left( \frac{1-\epsilon_2}{\epsilon_2} \right)$$

# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

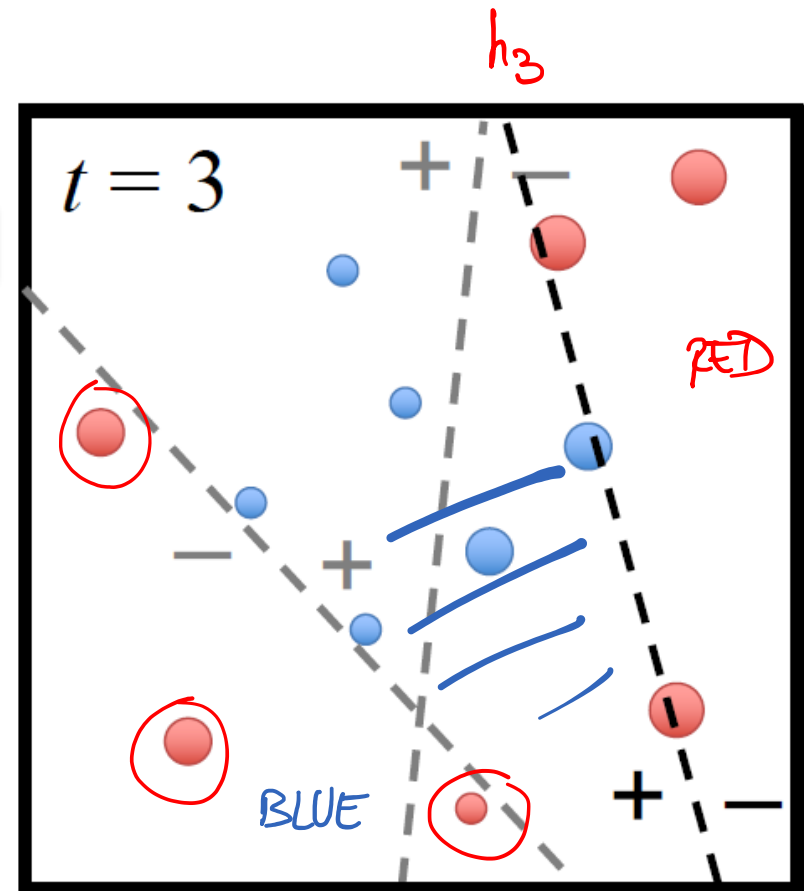


# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1 - \epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

- Compute importance of hypothesis  $\beta_t$
- Update weights  $w_t$

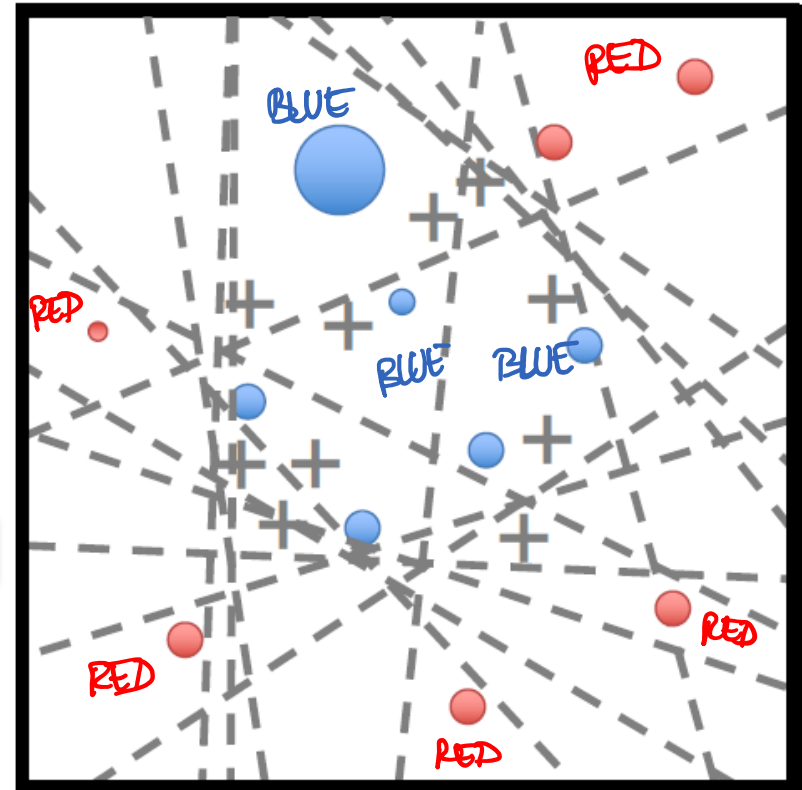


# AdaBoost

- 1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1$
- 2: **for**  $t = 1, \dots, T$
- 3:   Train model  $h_t$  on  $X, y$  with weights  $\mathbf{w}_t$
- 4:   Compute the weighted training error of  $h_t$
- 5:   Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$
- 6:   Update all instance weights:  
       $w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i))$
- 7:   Normalize  $\mathbf{w}_{t+1}$  to be a distribution
- 8: **end for**
- 9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

$t = T$



- Final model is a weighted combination of members
  - Each member weighted by its importance

# AdaBoost

**INPUT:** training data  $X, y = \{(\mathbf{x}_i, y_i)\}_{i=1}^n$ ,  
the number of iterations  $T$

1: Initialize a vector of  $n$  uniform weights  $\mathbf{w}_1 = [\frac{1}{n}, \dots, \frac{1}{n}]$

2: **for**  $t = 1, \dots, T$

3: Train model  $h_t$  on  $X, y$  with instance weights  $\mathbf{w}_t$

4: Compute the weighted training error rate of  $h_t$ :

$$\epsilon_t = \sum_{i: y_i \neq h_t(\mathbf{x}_i)} w_{t,i} \quad \text{ERROR}$$

5: Choose  $\beta_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$  IMPORTANCE OF  $h_t$

6: Update all instance weights:

$$w_{t+1,i} = w_{t,i} \exp(-\beta_t y_i h_t(\mathbf{x}_i)) \quad \forall i = 1, \dots, n$$

7: Normalize  $\mathbf{w}_{t+1}$  to be a distribution:

$$w_{t+1,i} = \frac{w_{t+1,i}}{\sum_{j=1}^n w_{t+1,j}} \quad \forall i = 1, \dots, n$$

8: **end for**

9: **Return** the hypothesis

$$H(\mathbf{x}) = \text{sign} \left( \sum_{t=1}^T \beta_t h_t(\mathbf{x}) \right)$$

$y_i \in \{-1, 1\}$

# Train with Weighted Instances

$x_i, w_i$  WEIGHT,  $y_i$  LABELS

1) LOGISTIC REGRESSION

$$L(x_i, y_i) = \sum_{i=1}^N \text{cost}(x_i, y_i) \cdot w_i \quad \text{(CROSS-ENTROPY)}$$

WEIGHT

2) DECISION TREES

SAMPLE FROM TRAINING DATA



# Train with Weighted Instances

- For algorithms like logistic regression, can simply incorporate weights  $w$  into the cost function
  - Essentially, weigh the cost of misclassification differently for each instance

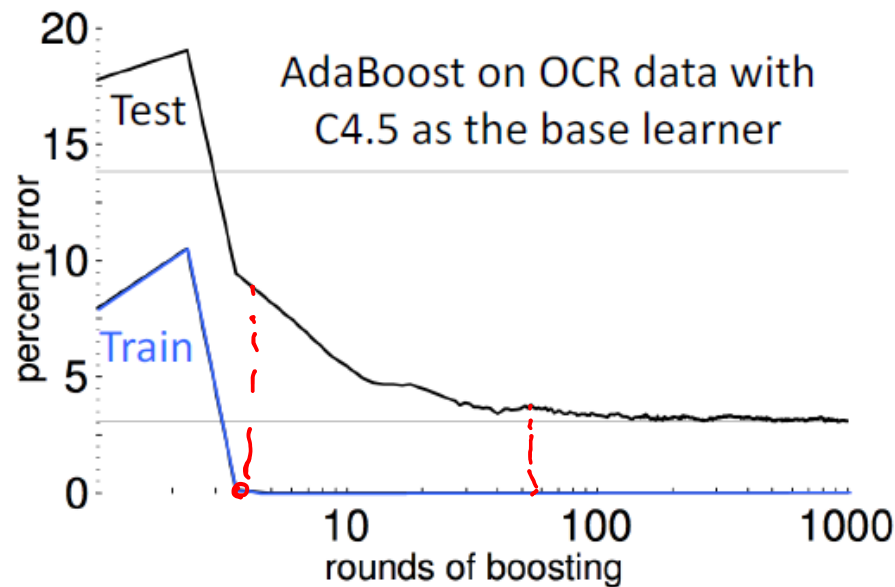
$$J_{\text{reg}}(\boldsymbol{\theta}) = - \sum_{i=1}^n w_i [y_i \log h_{\boldsymbol{\theta}}(\mathbf{x}_i) + (1 - y_i) \log (1 - h_{\boldsymbol{\theta}}(\mathbf{x}_i))] + \lambda \|\boldsymbol{\theta}_{[1:d]}\|_2^2$$

- For algorithms that don't directly support instance weights (e.g., ID3 decision trees, etc.), use weighted bootstrap sampling
  - Form training set by resampling instances with replacement according to  $w$

# Properties

- If a point is repeatedly misclassified
  - Its weight is increased every time
  - Eventually it will generate a hypothesis that correctly predicts it
- In practice AdaBoost does not typically overfit
- Does not use explicitly regularization

# Resilience to overfitting



- Empirically, boosting resists overfitting
- Note that it continues to drive down the test error even AFTER the training error reaches zero

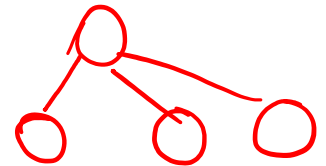
Increases confidence in prediction when adding more rounds

# Base Learner Requirements

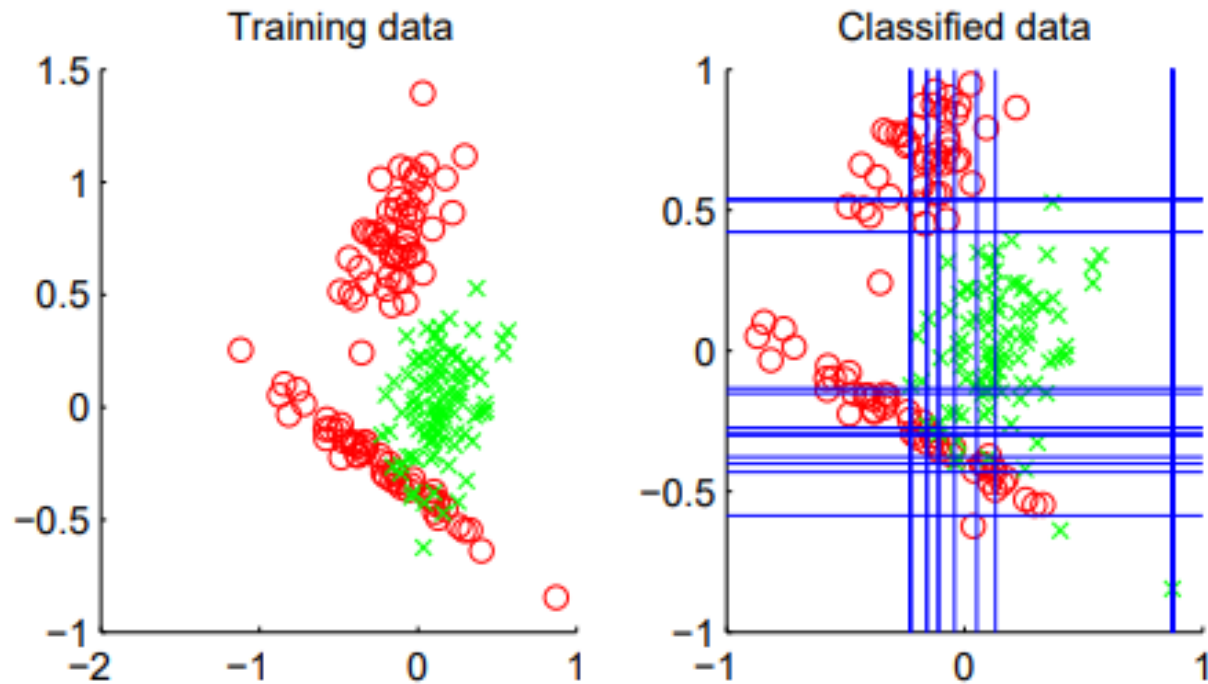
- AdaBoost works best with “weak” learners
  - Should not be complex
  - Typically high bias classifiers
  - Works even when weak learner has an error rate just slightly under 0.5 (i.e., just slightly better than random)
    - Can prove training error goes to 0 in  $O(\log n)$  iterations

- Examples:

- Decision stumps (1 level decision trees)
- Depth-limited decision trees
- Linear classifiers



# AdaBoost with Decision Stumps



# AdaBoost in Practice

## Strengths:

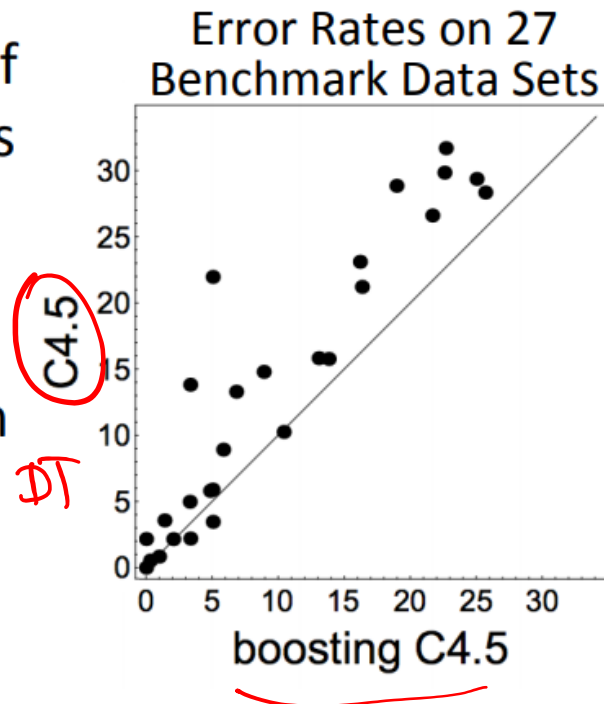
- Fast and simple to program
- No parameters to tune (besides T) **Learn with Cross-Validation**
- No assumptions on weak learner **Error less than  $\frac{1}{2}$**

## When boosting can fail:

- Given insufficient data
- Overly complex weak hypotheses
- Can be susceptible to noise
- When there are a large number of outliers

# Boosted Decision Trees

- Boosted decision trees are one of the best “off-the-shelf” classifiers
  - i.e., no parameter tuning
- Limit member hypothesis complexity by limiting tree depth
- Gradient boosting methods are typically used with trees in practice



“AdaBoost with trees is the best off-the-shelf classifier in the world” -Breiman, 1996  
(Also, see results by Caruana & Niculescu-Mizil, ICML 2006)

# Bagging vs Boosting

## BAGGING

- TRAINING: PARALLEL

DIVERSITY / BOOTSTRAP SAMPLES / RF FEATURE

- PREDICTION: SAME WEIGHT

- VAR ↓

## BOOSTING

- SEQUENTIAL

- WEIGHTS ON TRAINING

- WEIGHTED PRED.

- VAR ↓

- BIAS ↓



# Bagging vs Boosting

## Bagging

vs.

## Boosting

Resamples data points

Weight of each classifier is the same

Only variance reduction

Reweights data points (modifies their distribution)

Weight is dependent on classifier's accuracy

Both bias and variance reduced – learning rule becomes more complex with iterations

Applicable to complex models with low bias, high variance

Applicable to weak models with high bias, low variance

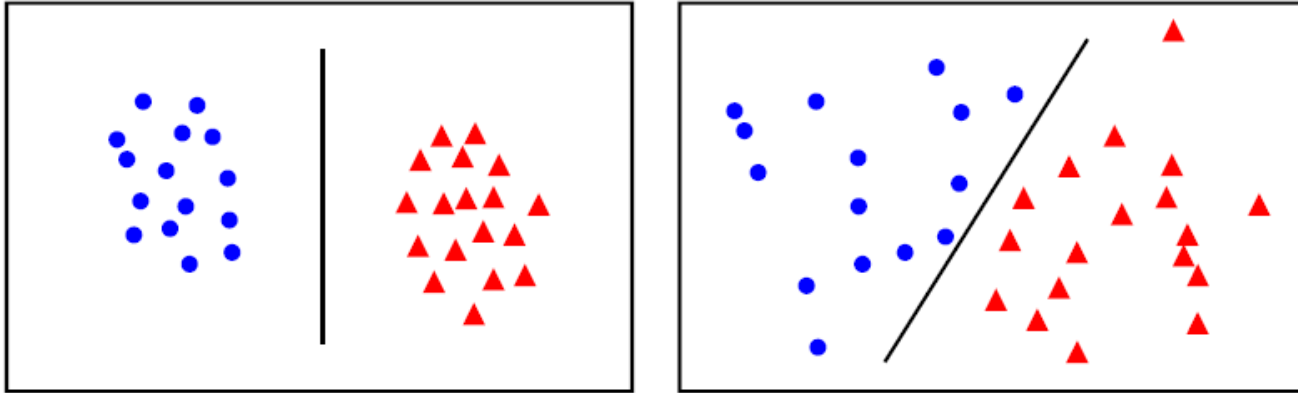
# Review

- Ensemble learning are powerful learning methods
  - Better accuracy than standard classifiers
- Bagging uses bootstrapping (with replacement), trains  $T$  models, and averages their prediction
  - Random forests vary training data and feature set at each split
- Boosting is an ensemble of  $T$  weak learners that emphasizes mis-predicted examples
  - AdaBoost has great theoretical and experimental performance
  - Can be used with linear models or simple decision trees (stumps, fixed-depth decision trees)

# Outline

- Boosting
  - AdaBoost
  - Properties of boosting
  - Bagging vs Boosting
- Review of linear models
  - Separating hyperplanes
- Support Vector Machines
  - Linearly separable data
    - Maximum margin classifier
  - Non-linearly separable data
    - Support vector classifier

# Linear models we've seen

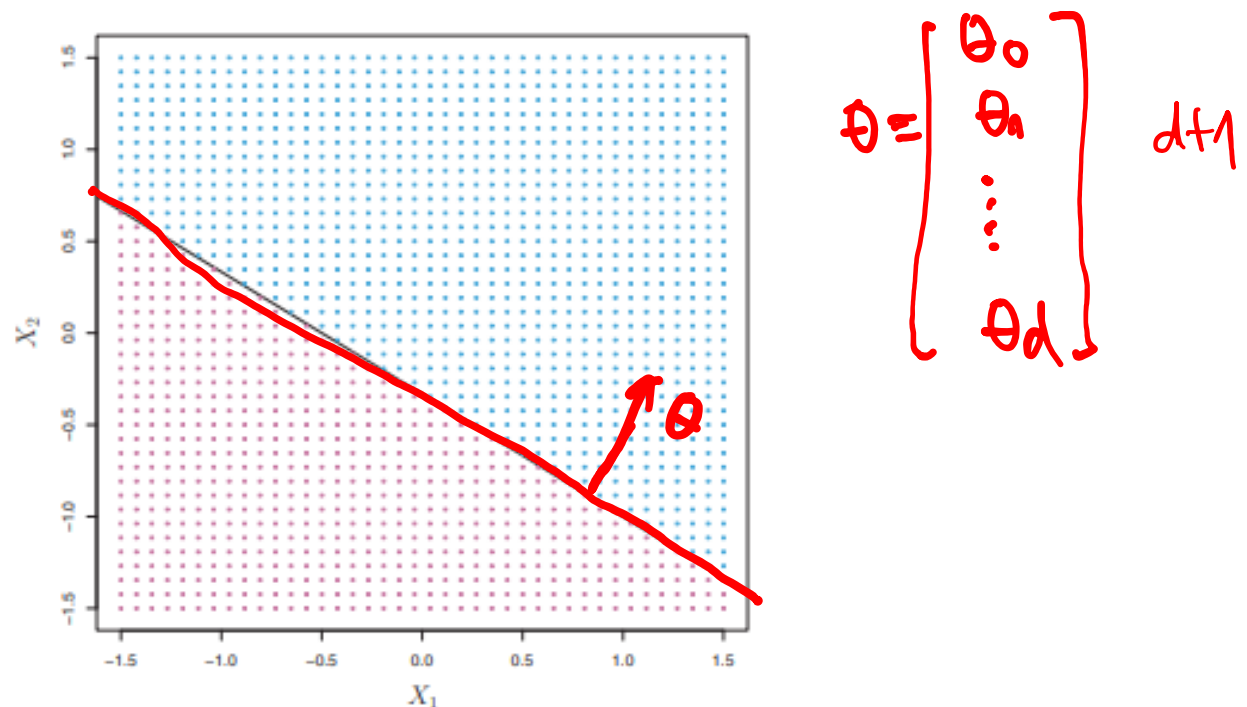


Classifiers with linear decision boundary:

- LOGISTIC REG
- LDA
- NB (SOME CASES)

# Hyperplane

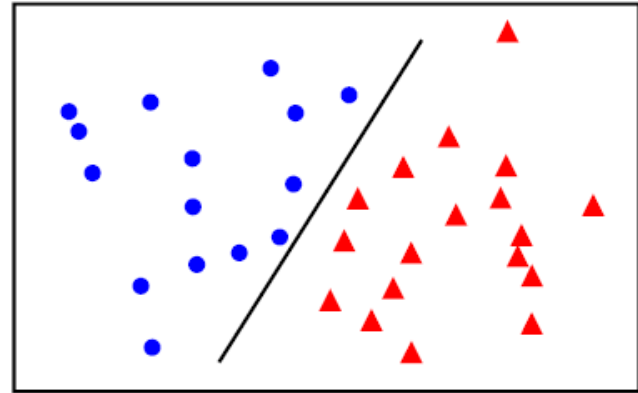
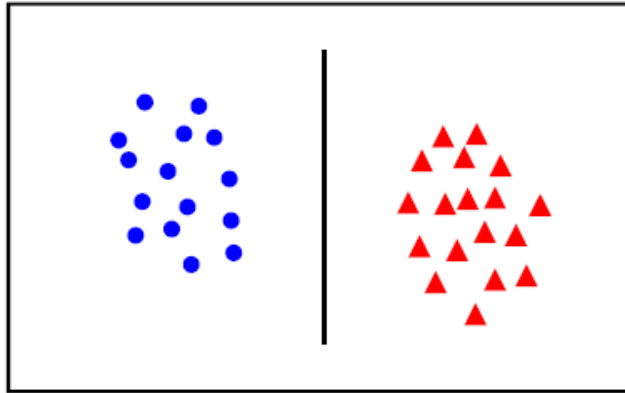
- Line (2-dimensions):  $\theta_0 + \theta_1 x_1 + \theta_2 x_2 = 0$
- Hyperplane (d-dimensions):  $\theta_0 + \theta_1 x_1 + \cdots \theta_d x_d = 0$



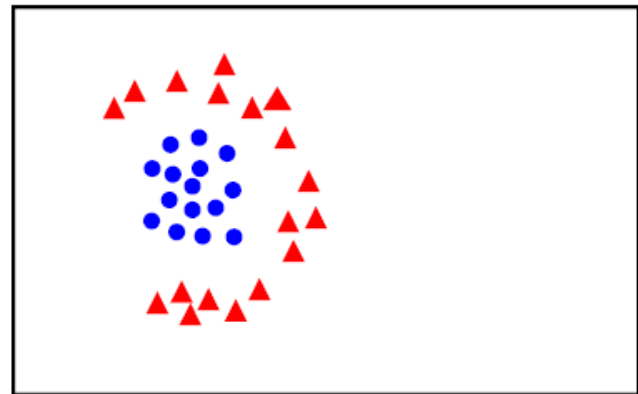
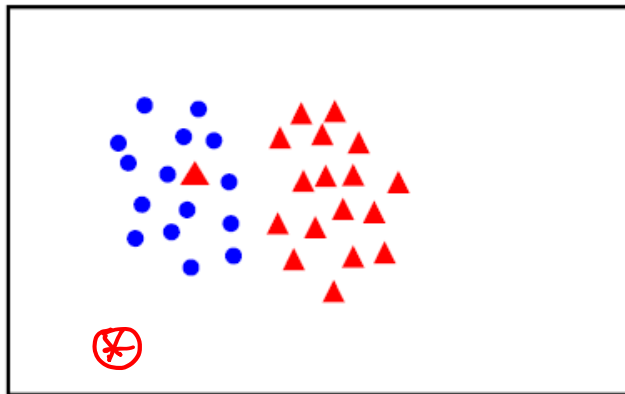
**FIGURE 9.1.** The hyperplane  $1 + 2X_1 + 3X_2 = 0$  is shown. The blue region is the set of points for which  $1 + 2X_1 + 3X_2 > 0$ , and the purple region is the set of points for which  $1 + 2X_1 + 3X_2 < 0$ .

# Linear separability

linearly  
separable



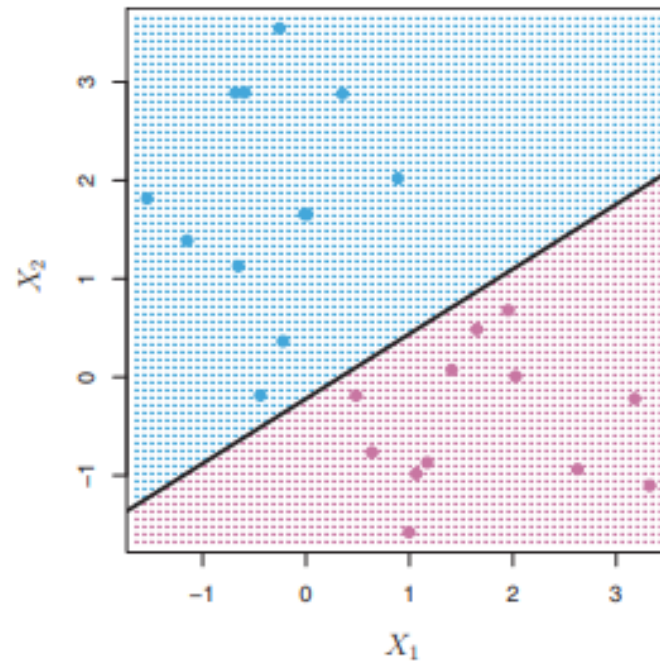
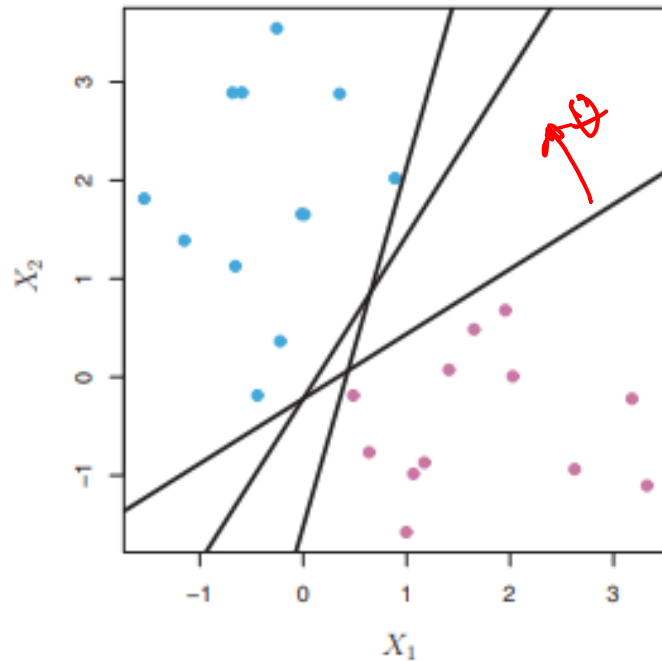
not  
linearly  
separable



# Notation

- Training data  $x_1, \dots, x_N$  with  $x_i = (\underbrace{x_{i1}, \dots, x_{id}}_{x_i})^T$
- Labels are from 2 classes:  $y_i \in \{-1, 1\}$
- Goal:
  - Build a model to classify training data
  - Test it on new vector  $x = (x_1, \dots, x_d)^T$  to predict label  $y$

# Separating hyperplane

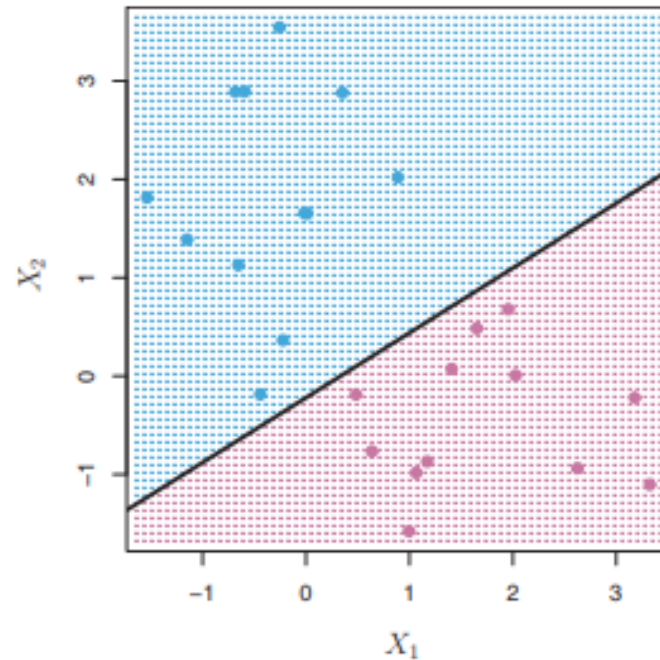
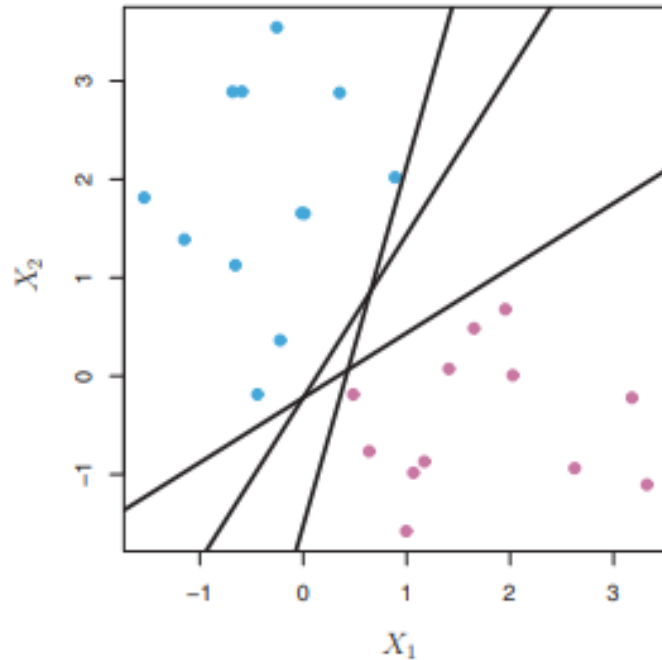


If  $y_i = 1 \Rightarrow \theta^T x_i > 0$   
 $y_i = -1 \Rightarrow \theta^T x_i < 0$

PERFECT SEPARATION



# Separating hyperplane



$$y_i \cdot (\theta^T x_i) > 0, \quad \forall i = 1, \dots, N$$

$$y_i (\theta_0 + \theta_1 x_{i1} + \dots + \theta_d x_{id}) > 0$$

$$y_i \in \{-1, 1\}$$

# From separating hyperplane to classifier

- Training data  $x_1, \dots, x_N$  with  $x_i = (x_{i1}, \dots, x_{id})^T$
- Labels are from 2 classes:  $y_i \in \{-1, 1\}$
- Let  $\theta_0, \dots, \theta_d$  (will be learned) such that:

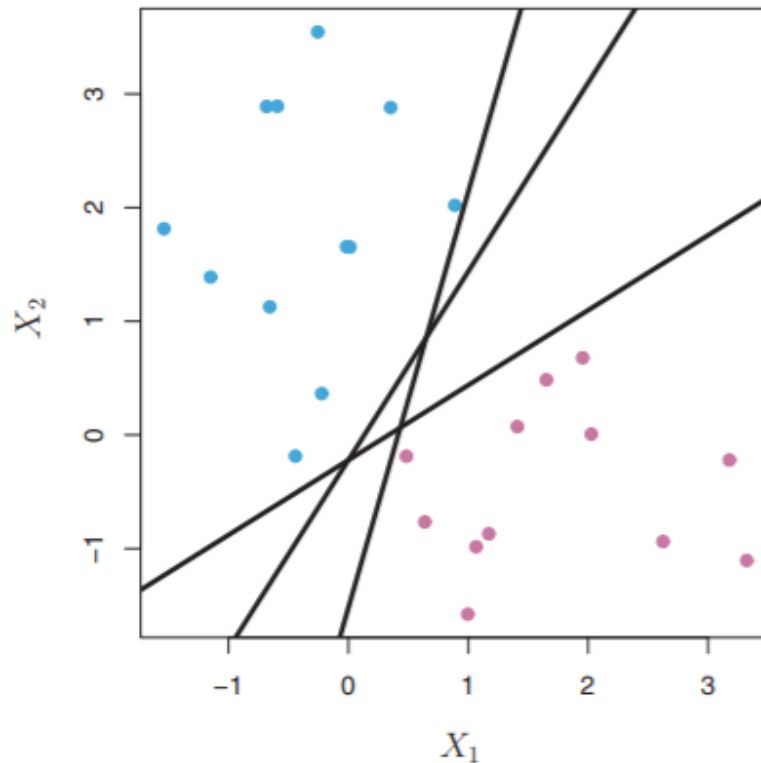
$$y_i(\theta_0 + \theta_1 x_{i1} + \dots + \theta_d x_{id}) > 0$$

- Classifier

$$f(z) = \text{sign}(\theta_0 + \theta_1 z_1 + \dots + \theta_d z_d) = \text{sign}(\theta^T z)$$

- Classify new test point  $x$ 
  - If  $f(x) > 0$  predict  $y = 1$
  - Otherwise predict  $y = -1$

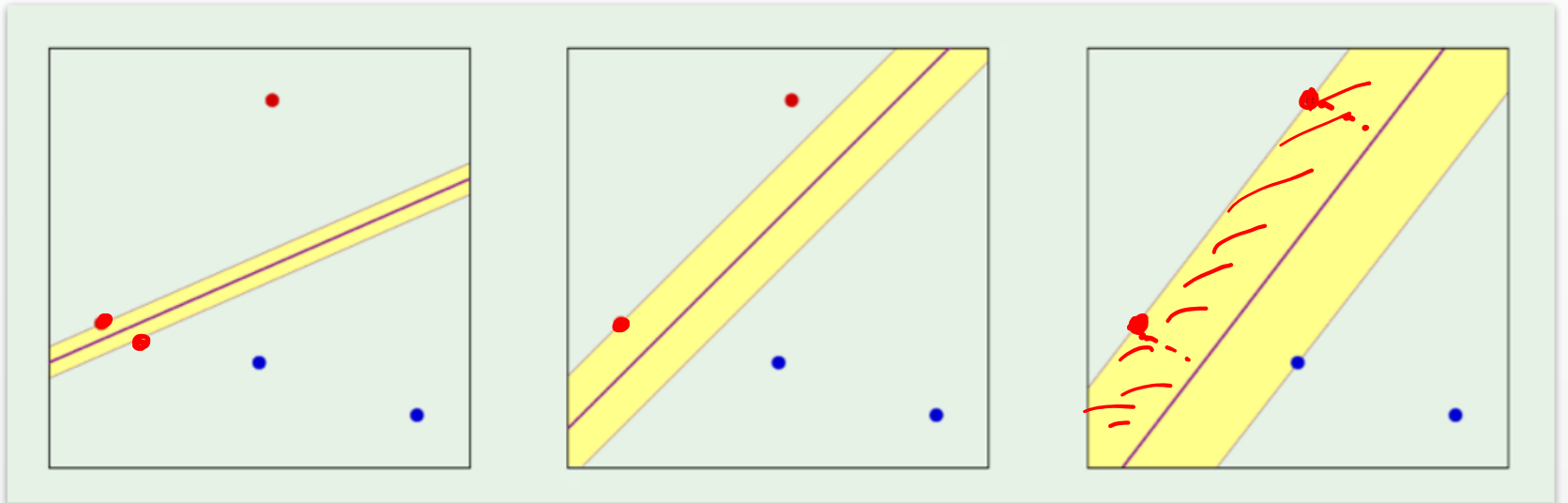
# Separating hyperplane



- If a separating hyperplane exists, there are infinitely many
- Which one should we choose?

# Intuition

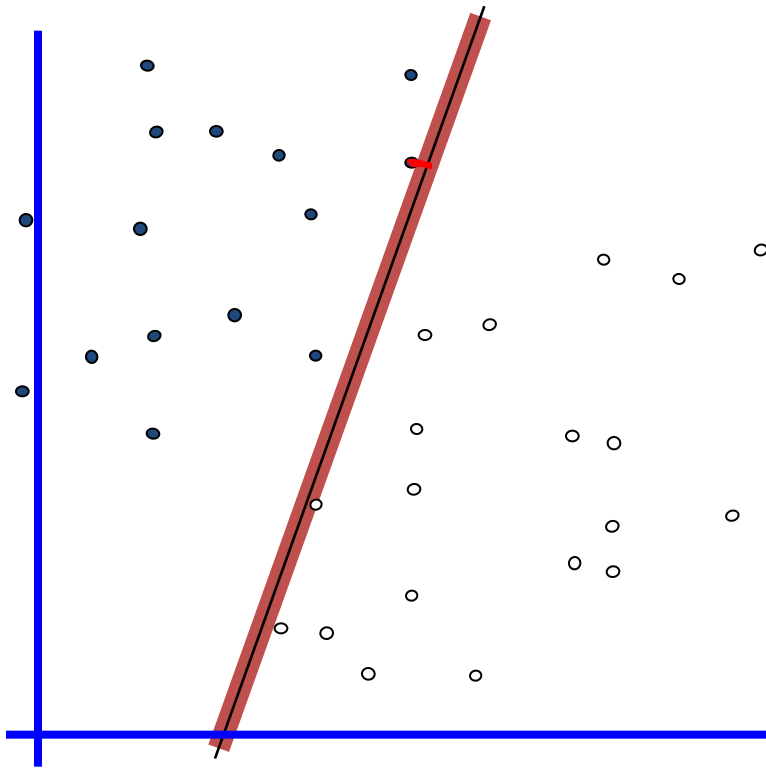
MAX SEPARATION



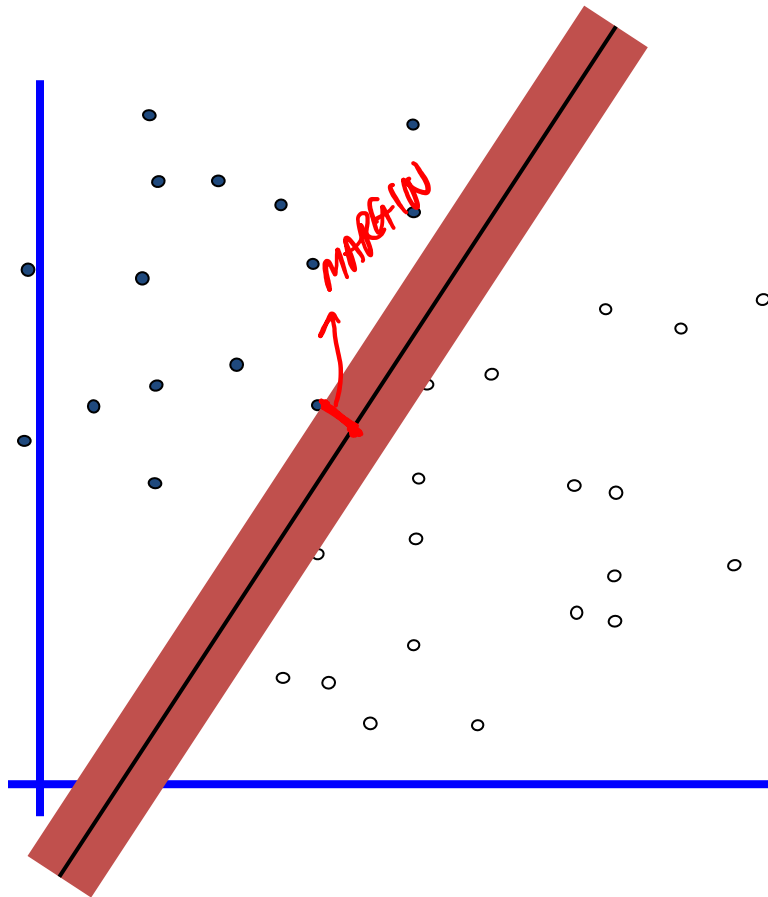
Which of these linear classifiers is the best?

# Classifier Margin

Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.



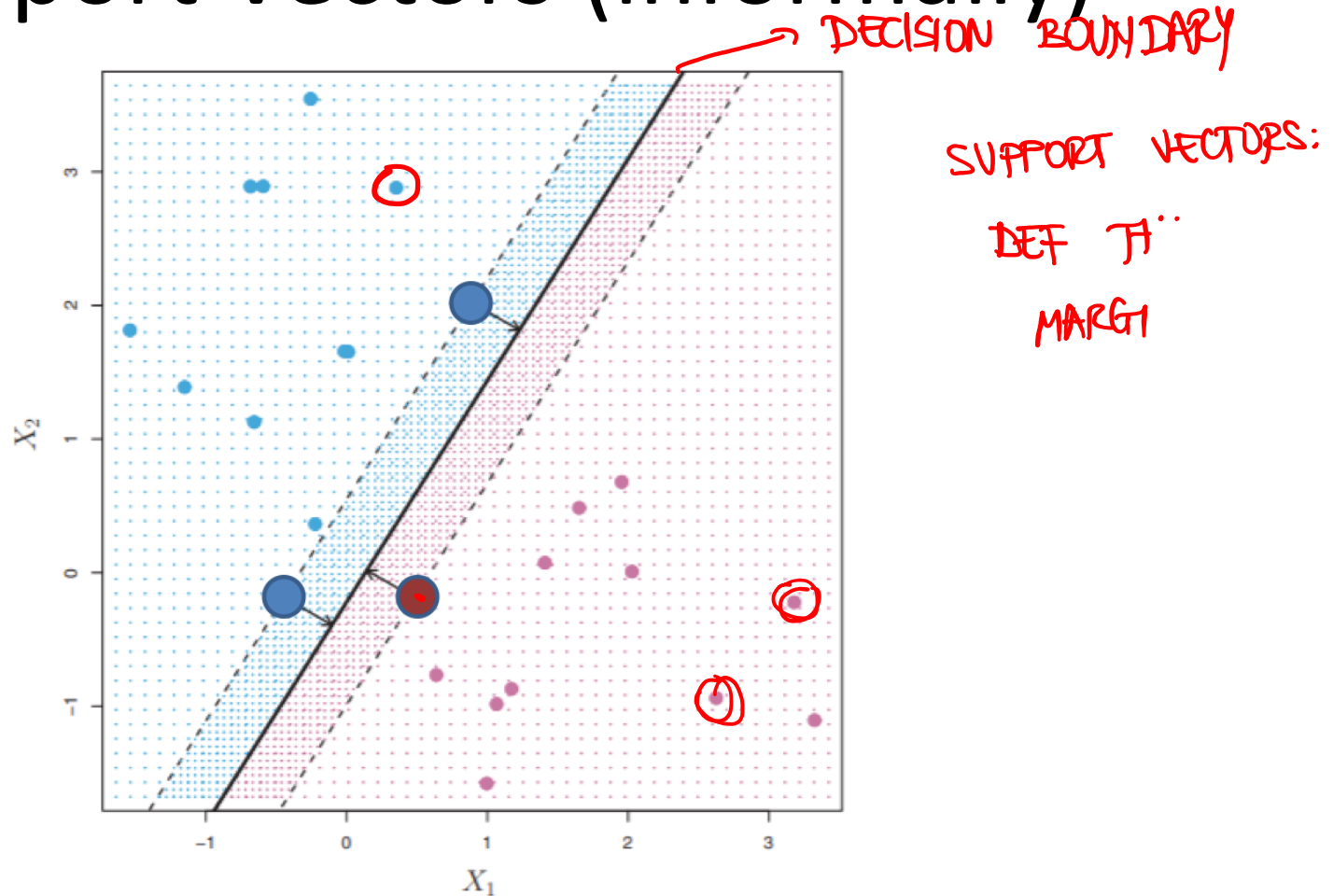
# Maximum Margin



Define the **margin** of a linear classifier as the width that the boundary could be increased by before hitting a data point.

Choose the **maximum margin linear classifier**: the linear classifier with the maximum margin.

# Support Vectors (informally)



- Support vectors = points “closest” to hyperplane
- If support vectors change, classifier changes
- If other points change, no effect on classifier

# Finding the maximum margin classifier

- Training data  $x_1, \dots, x_N$  with  $x_i = (x_{i1}, \dots, x_{id})^T$
- Labels are from 2 classes:  $y_i \in \{-1, 1\}$

MARGIN

maximize  $M$

$$y_i(\theta_0 + \theta_1 x_{i1} + \dots + \theta_d x_{id}) \geq M \quad \forall i=1, \dots, N.$$

$$\|\theta\|_2 = 1$$

Normalization constraint  
(to have unique solution)

Each point is on the  
right side of hyper-  
plane at distance  $\geq M$



# Properties of solution

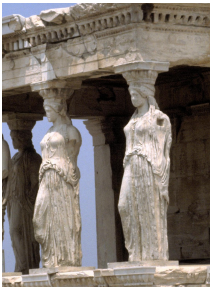
- The solution to the optimization provides a convenient way to rewrite the decision function using new variables  $\alpha_i$

– Originally:  $f(z) = \text{sign}(\theta_0 + \theta_1 z_1 + \cdots \theta_d z_d)$   
 $= \text{sign}(\theta^T z)$

→ Equivalent to:  $f(\mathbf{z}) = \theta_0 + \sum_i \alpha_i \langle \mathbf{z}, \mathbf{x}_i \rangle$

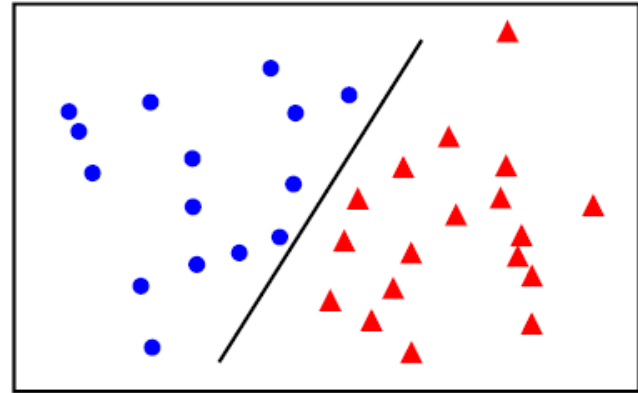
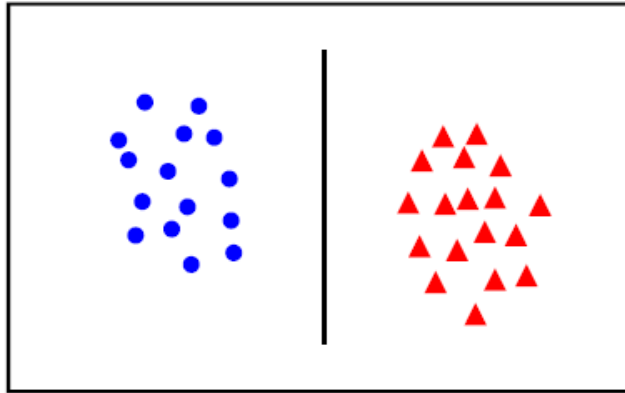
- For test point  $z$ , the inner product  $\langle z, x_i \rangle = z^T x_i$  with each training instance  $x_i$  in turn.
- And  $\alpha_i \neq 0$  only for support vectors! For all other training points  $\alpha_i = 0$ .

$\alpha_i = 0$  FOR ALL TR. POINTS THAT ARE  
NOT SUPPORT VECTORS

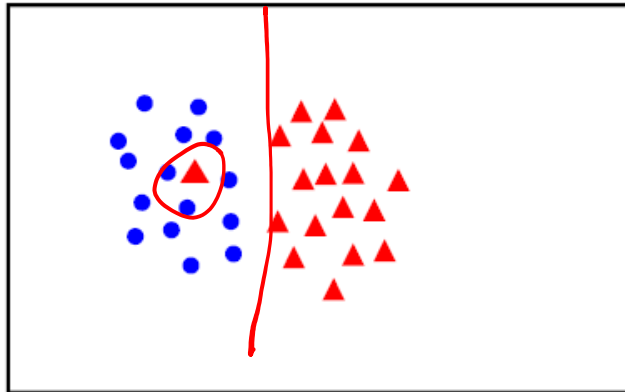


# Linear separability

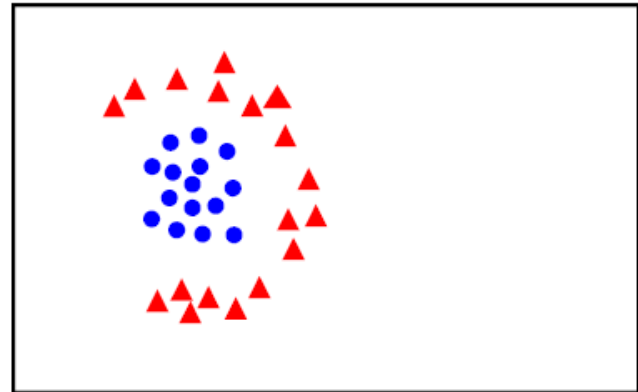
linearly  
separable



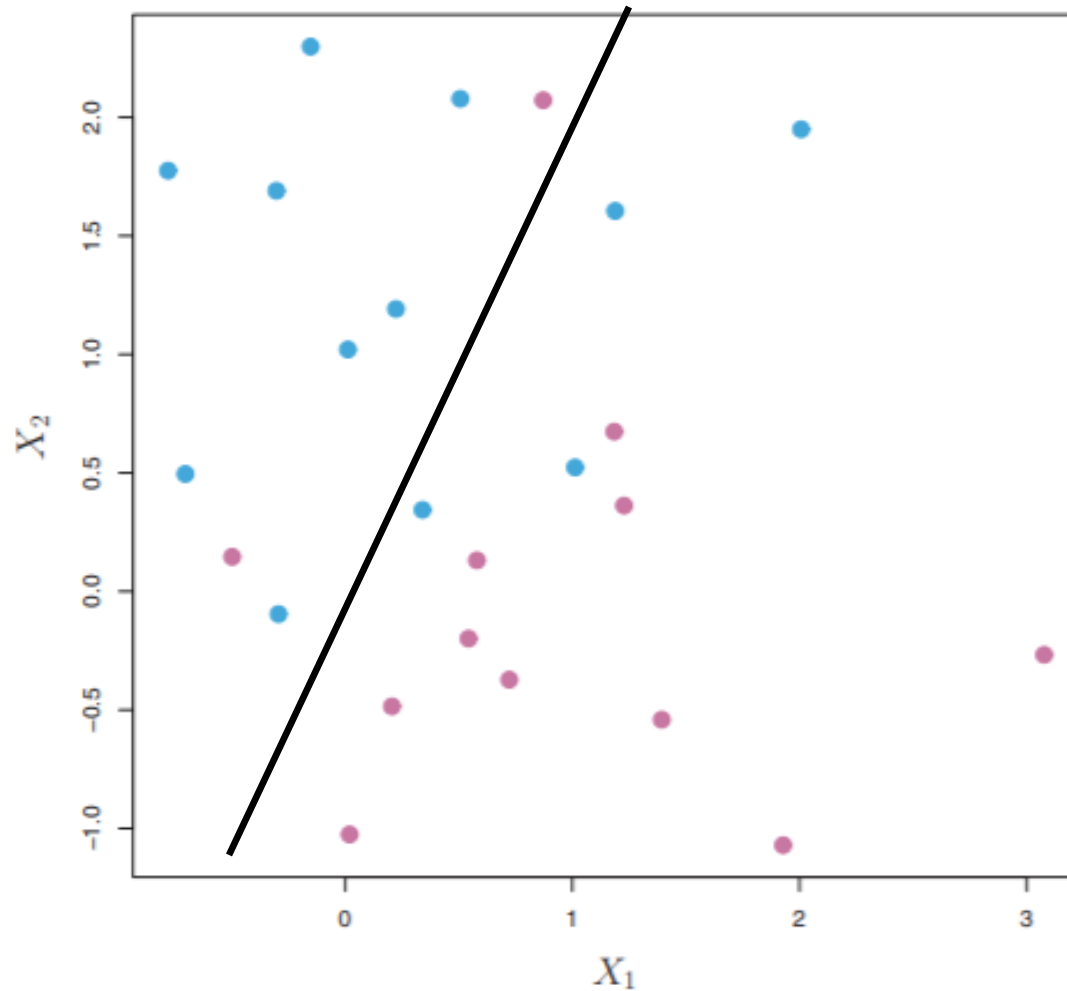
not  
linearly  
separable  
(but almost)



$$y_i \cdot (\theta^T x_i) < 0$$



# Non-separable case



Optimization problem has no solution!

# Support vector classifier

- Allow for small number of mistakes on training data
- Soft margin classifier

max  $M$

$$y_i(\theta_0 + \theta_1 x_{i1} + \cdots \theta_d x_{id}) \geq M(1 - \epsilon_i) \forall i$$

$$\|\theta\|_2 = 1$$

$$\epsilon_i \geq 0, \sum_i \epsilon_i \leq C$$

Slack

Error Budget (Hyper-parameter)

OPT PB  
SLACK

max M

$$y_i \cdot (\theta^T x_i)$$

$$y_i(\theta_0 + \theta_1 x_{i1} + \dots + \theta_d x_{id}) \geq M(1 - \epsilon_i) \quad \forall i$$

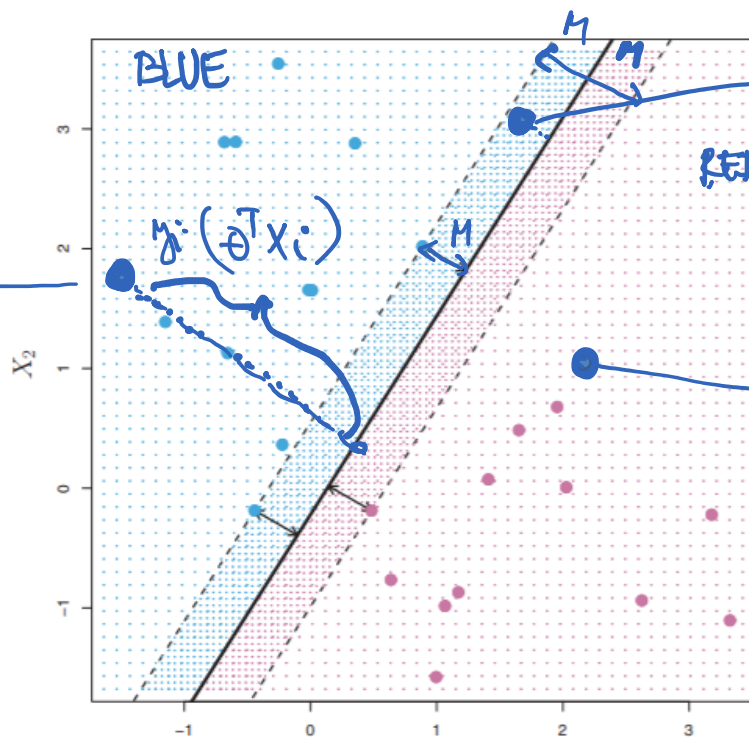
$$\|\theta\|_2 = 1$$

$$\epsilon_i \geq 0, \sum_i \epsilon_i \leq C$$

Error  
Budget

Slack

$\epsilon_i = 0$   
CORRECT SIDE  
MARGIN  $> M$

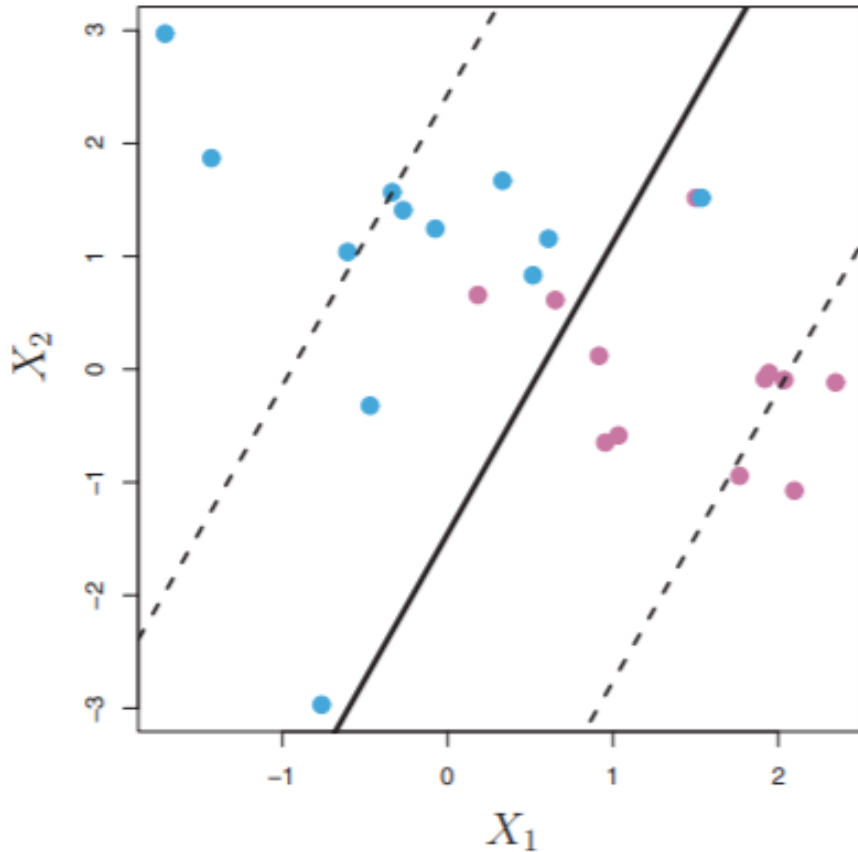


CORRECT SIDE  
DIST  $< M$   
VIOLATES MARGIN  
 $0 < \epsilon_i < 1$

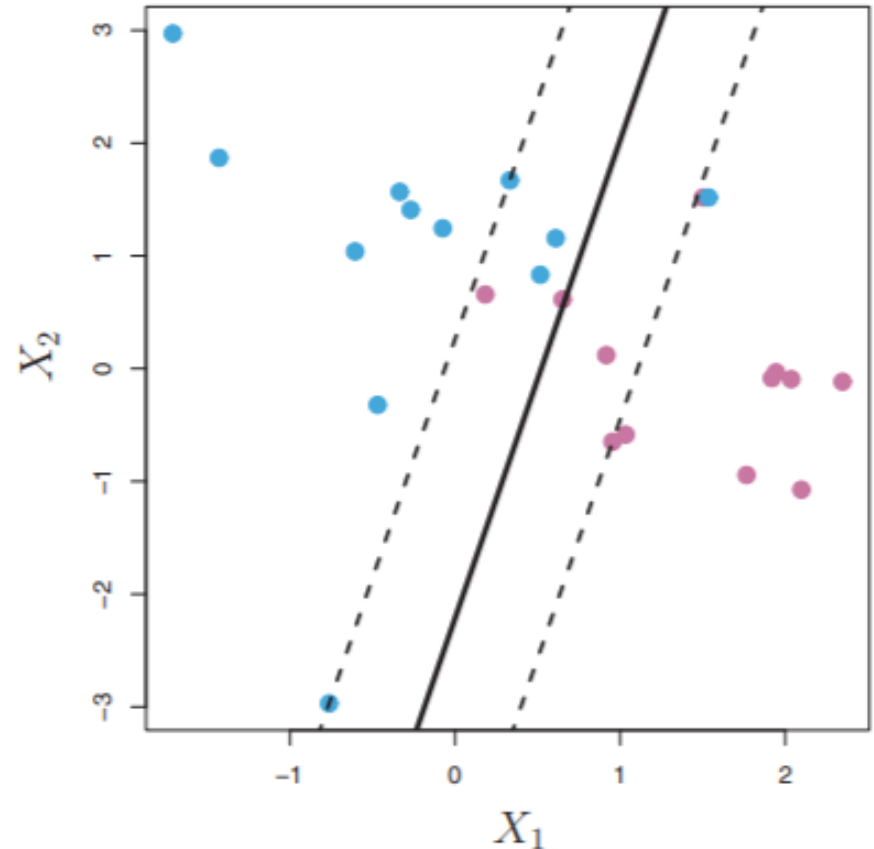
INCORRECT LABEL  
 $y_i \cdot (\theta^T x_i) < 0$   
 $\epsilon_i > 1$

AT MOST C ERRORS

# Error Budget and Margin



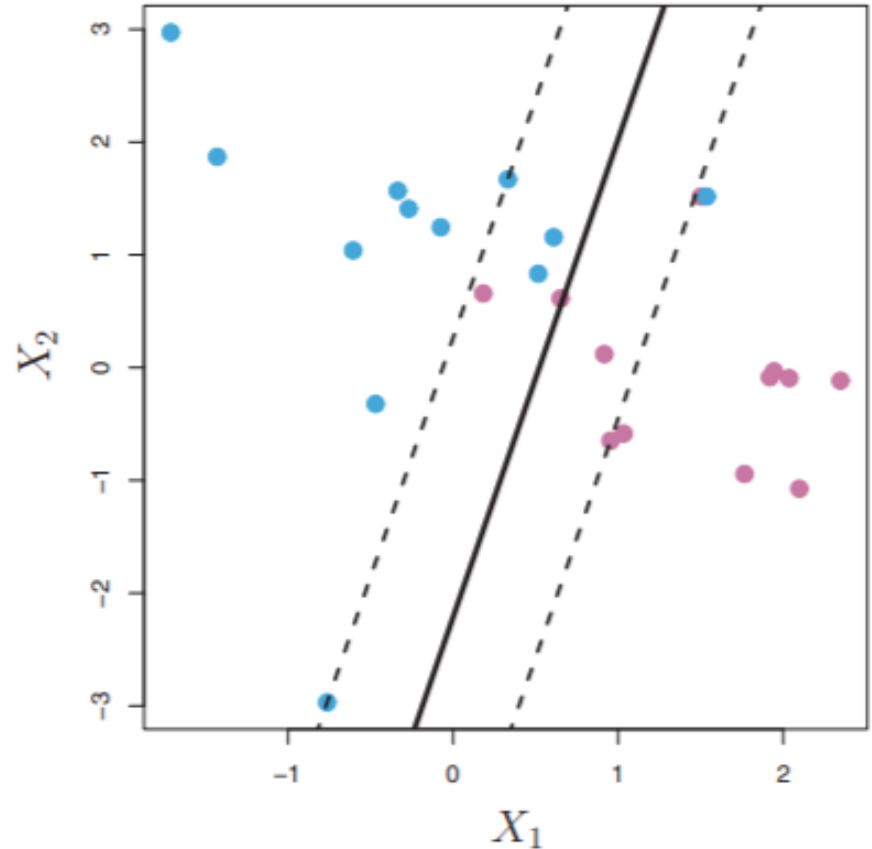
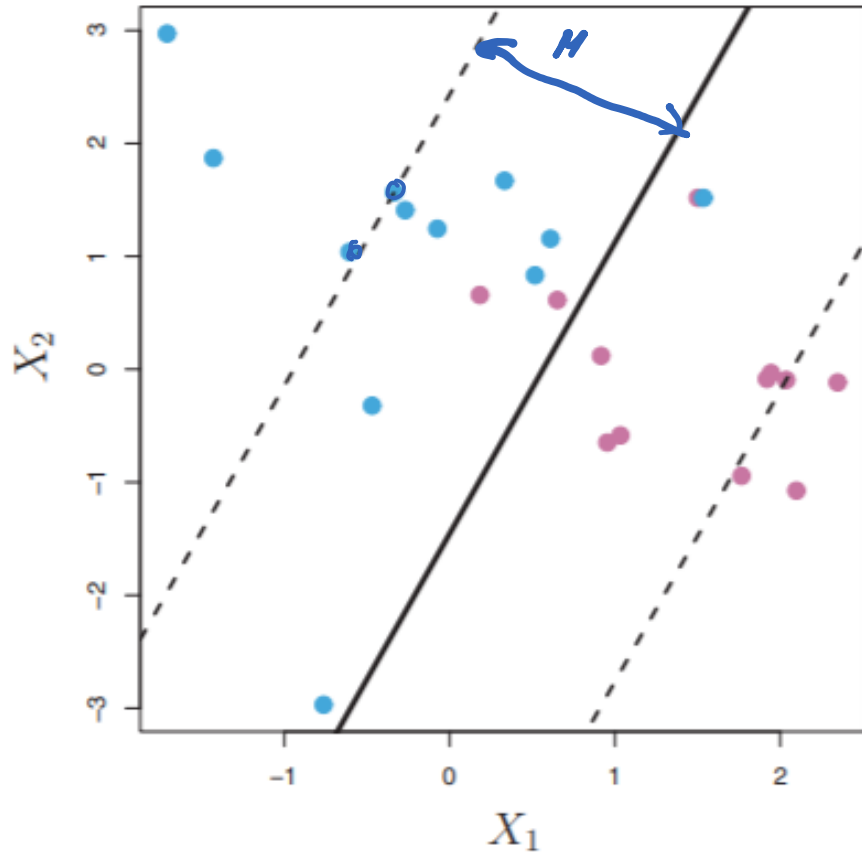
Larger  $C$   
LOW VAR.



Smaller  $C$   
HIGH VAR  
OVERFIT

Find best hyper-parameter  $C$  by cross-validation

# Support vectors



Support vectors: all points within the margin of the classifier

# Support vector classifier

- Just like in separable case, gives solution of the form:


$$f(z) = \theta_0 + \sum_i \alpha_i \langle z, x_i \rangle$$

Where  $\alpha_i \neq 0$  for support vectors (and  $\alpha_i = 0$  for all other training points)

- This model is called
  - Support Vector Classifier
  - Linear SVM
  - Soft-margin classifier



# Properties

- **Maximum margin classifier**
  - Classifier of maximum margin
  - For linearly separable data
- **Support vector classifier** / Linear SVM
  - Allows some slack and sets a total error budget (hyper-parameter) 
- For both, final classifier on a point is a linear combination of inner product of point with support vectors
  - Efficient to evaluate

# Acknowledgements

- Slides made using resources from:
  - Andrew Ng
  - Eric Eaton
  - David Sontag
  - Andrew Moore
- Thanks!