# Machine Learning

## CS 165B

Prof. Matthew Turk

Monday, May 23, 2016

Today

- Model ensembles (Ch. 11)
  - Bagging and boosting
- Machine learning experiments (Ch. 12)

# Notes

## Remaining schedule

- Main topics:
  - Model ensembles
  - Machine learning experiments
  - Neural networks
  - What's hot in machine learning
- For grades:
  - Homework assignment #5
    - Due Friday, June 3
  - Final exam
    - Wednesday, June 8 (8-11am!)
- Next week
  - No class next Monday (Memorial Day holiday)
  - Final exam review… when?
    - More info soon

# Model ensembles

Chapter 11 in the textbook

# Model ensembles

- We've seen how combining features can be beneficial
- We can also combine models to increase performance
  - Combinations of models are known as model ensembles
  - Potential for better performance at the cost of increased complexity
- General approach to model ensembles:
  - Construct multiple different models from adapted versions of the training data (e.g., reweighted or resampled)
  - Combine the predictions of these models in some way (averaging, voting, weighted voting, etc.)
- Two of the best-known ensemble methods are bagging and boosting
- These are "meta" methods – i.e., they are independent of the particular learning method (linear classifier, SVM, etc.)

# Bagging

- Bagging = "bootstrap aggregation"
  - Create $T$ models on different random samples of the training data set
  - Each sample is a set of training data called a bootstrap sample
    - Could be any size – often $|D|$ is used, the size of the training set
- Bootstrap samples: Sample the data set with replacement (i.e., a sample can be chosen more than once in a bootstrap sample)
  - For $j = 1$ to $|D|$, choose with uniform probability from training data points $D = \{ d_1, d_2, \ldots, d_{|D|} \}$
  - Gather these into the bootstrap sample $D_i$
- Use $\{ D_1, D_2, \ldots, D_T \}$ to train models $\{ M_1, M_2, \ldots, M_T \}$, then combine the predictions of the $T$ models
- Differences between the $T$ bootstrap samples create diversity among the models in the ensemble

# Bagging

**Algorithm** Bagging($D, T, \mathscr{A}$) – train an ensemble of models from bootstrap samples.

**Input** : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.

**Output** : ensemble of models whose predictions are to be combined by voting or averaging.

**for** $t = 1$ to $T$ **do**

  build a bootstrap sample $D_t$ from $D$ by sampling $|D|$ data points with replacement;

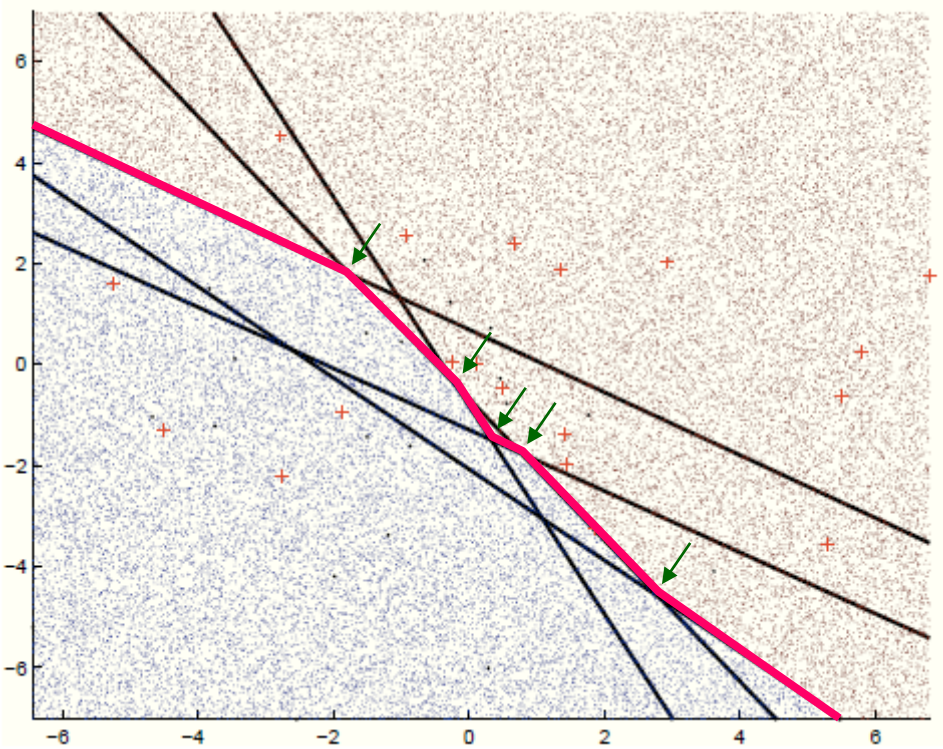
  run $\mathscr{A}$ on $D_t$ to produce a model $M_t$;

**end**

**return** $\{M_t | 1 \leq t \leq T\}$

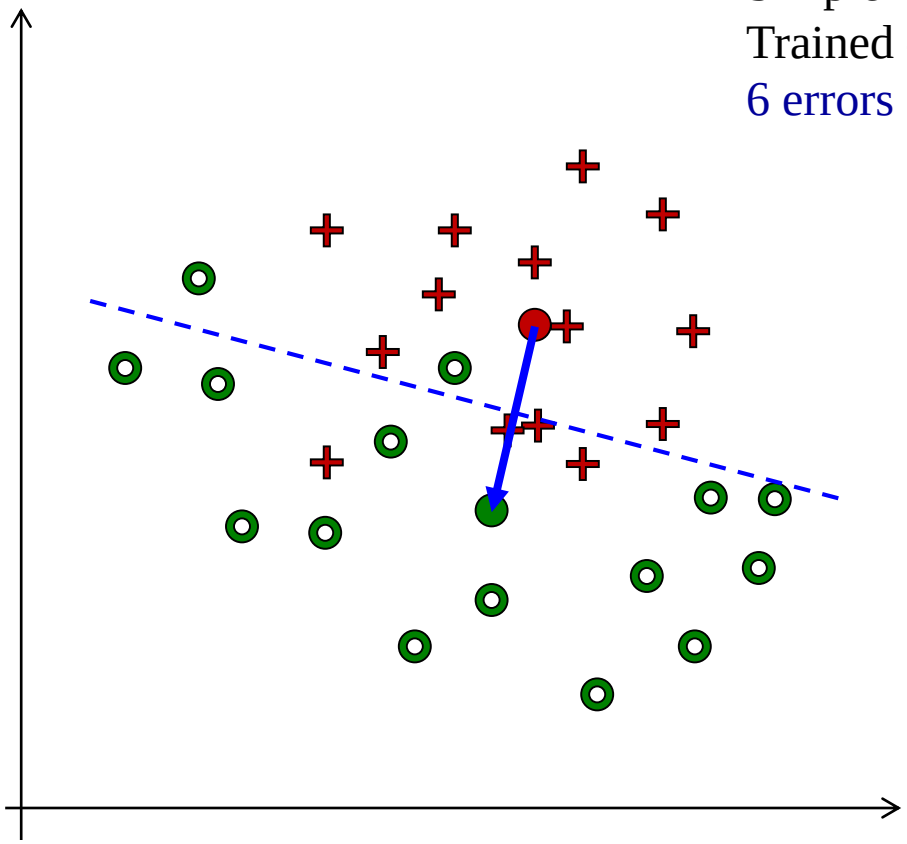
# Bagging example 1

- Train 5 binary linear classifiers by bagging
- Use majority vote (for example) to determine classification output for a new $\boldsymbol{x}$
- The decision boundary is piecewise linear



# Bagging example 2

|D| = 29

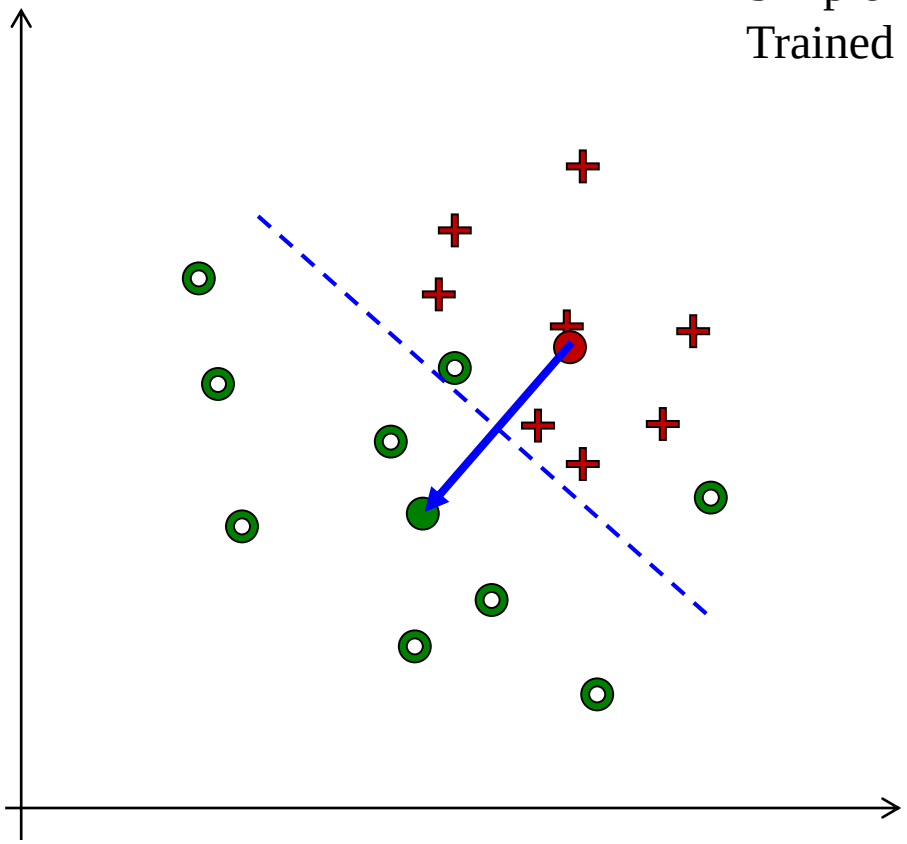


Simple linear classifier
Trained on all data
6 errors

# Bagging example 2

Simple linear classifier
Trained on bootstrap sample set 1

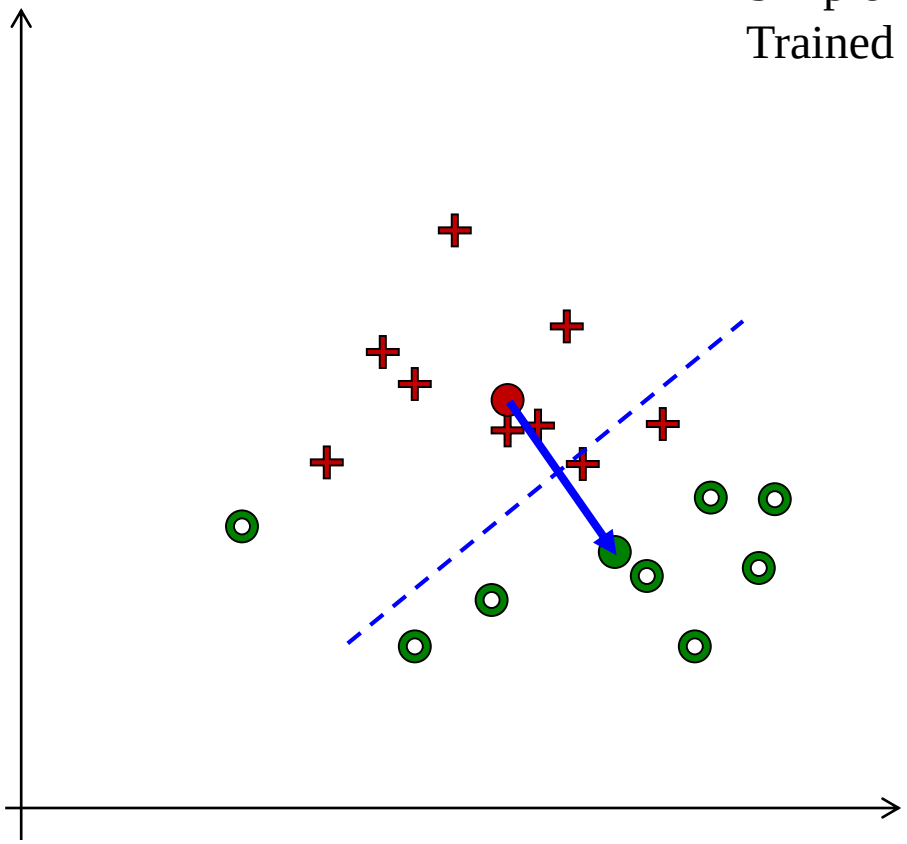
$|D_1| = 17$



# Bagging example 2

Simple linear classifier
Trained on bootstrap sample set 2

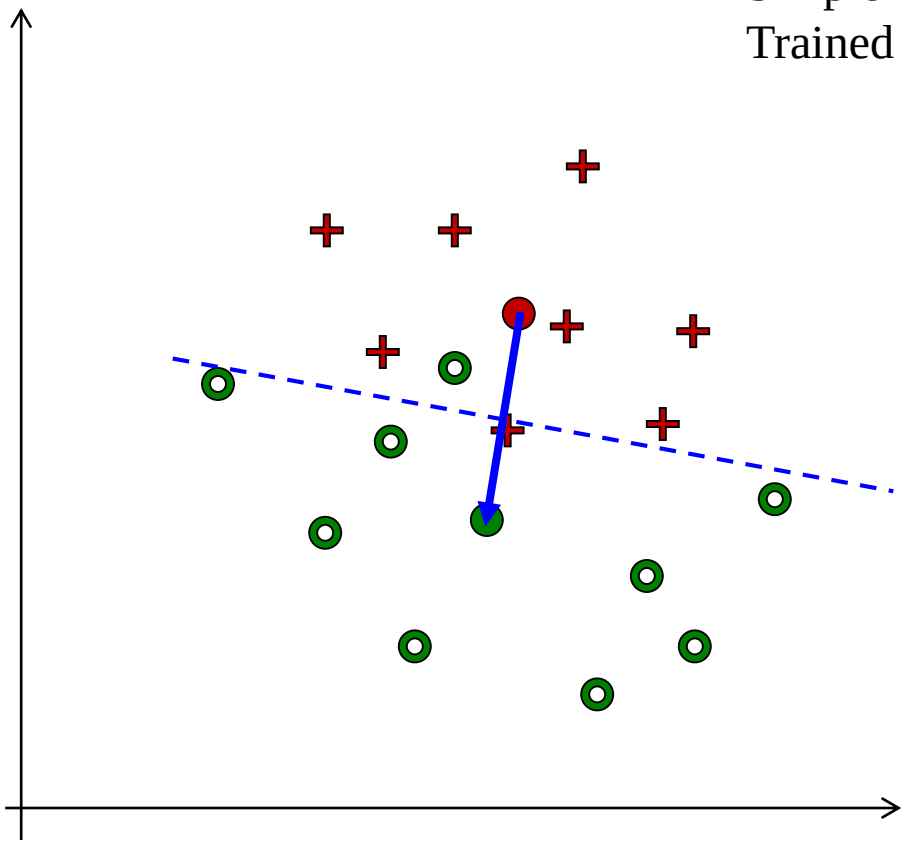
$|D_2| = 17$



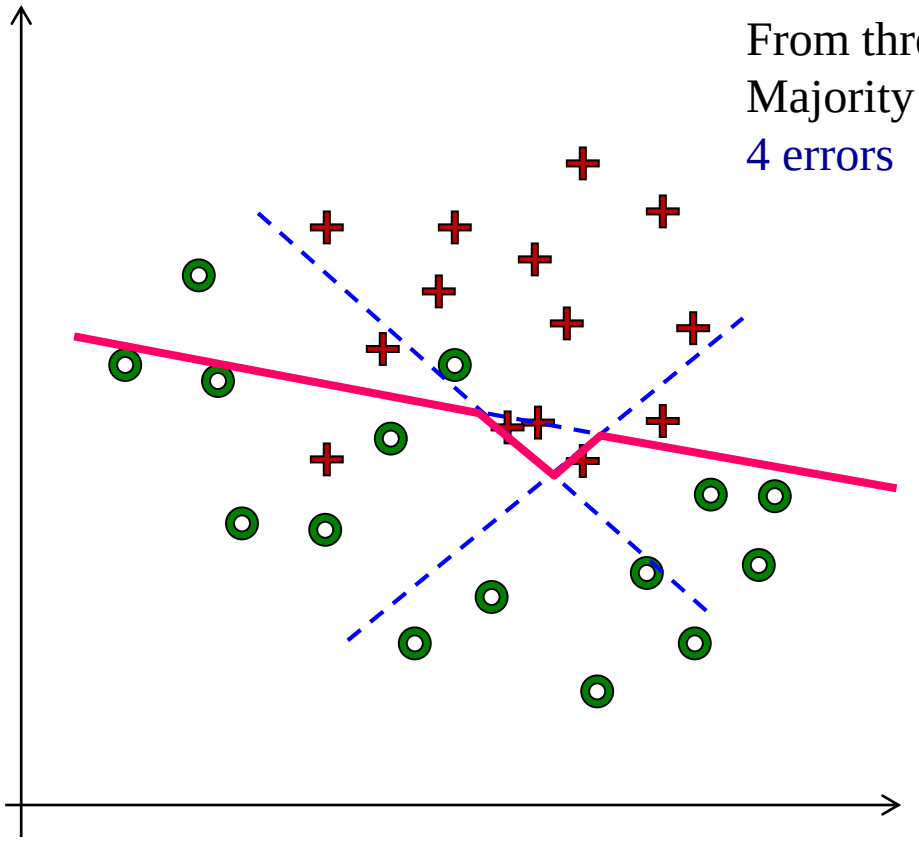
# Bagging example 2

Simple linear classifier
Trained on bootstrap sample set 3

$|D_3| = 17$



# Bagging example 2



Ensemble linear classifier
From three boosting data sets
Majority vote
4 errors

# Bagging with tree models

- Bagging is particularly useful for tree models (e.g., decision trees), which can be very sensitive to variations in the training data
  - E.g., build *T* decision trees, produce *T* output labels for an input, then vote to determine the classifier output

Also:

- Subspace sampling: Build each decision tree from a different random subset of the **features**

A bunch of trees

Bagging + subspace sampling = *random forests* method

# Bagging with tree models

**Algorithm** RandomForest($D, T, d$) – train an ensemble of tree models from bootstrap samples and random subspaces.

**Input** : data set $D$; ensemble size $T$; subspace dimension $d$.

**Output** : ensemble of tree models whose predictions are to be combined by voting or averaging.

**for** $t = 1$ to $T$ **do**

  build a bootstrap sample $D_t$ from $D$ by sampling $|D|$ data points with replacement;

  select $d$ features at random and reduce dimensionality of $D_t$ accordingly;

  train a tree model $M_t$ on $D_t$ without pruning;

**end**

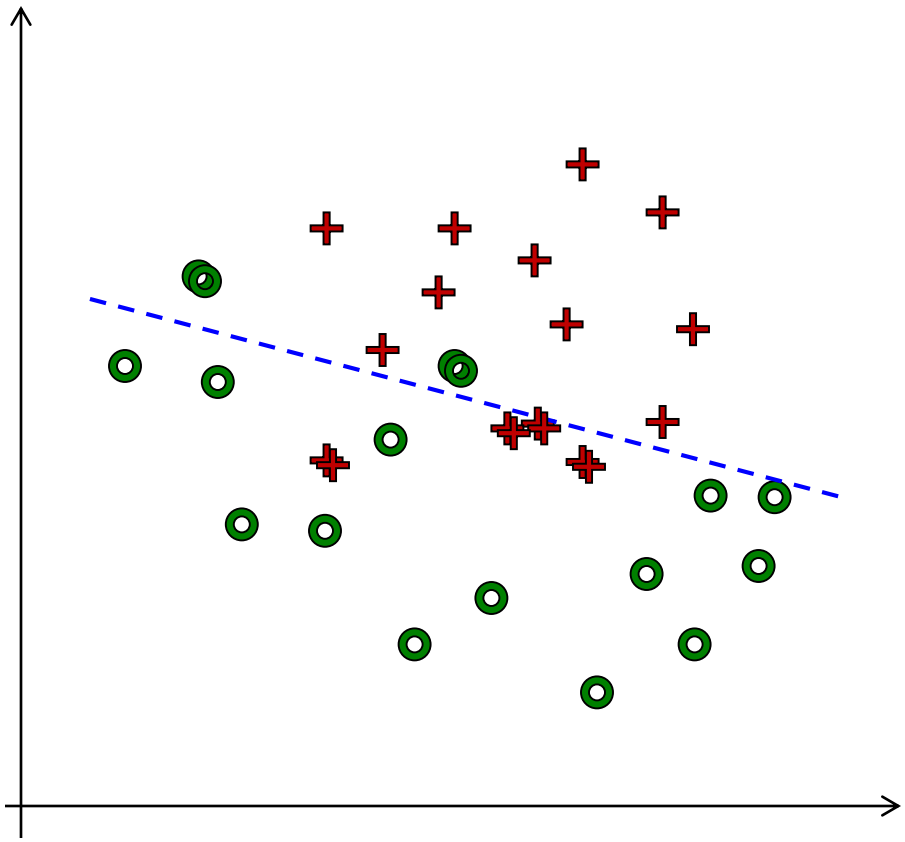
**return** $\{M_t | 1 \leq t \leq T\}$

# Boosting

- Boosting is similar to bagging, but it uses a more sophisticated technique to create diverse training sets
- The focus is on adding classifiers that do better on the misclassifications from earlier (Boolean) classifiers
  - By giving them a higher weight
- As with bagging, boosting attempts to create a *strong classifier* (*learner*) from a set of *weak classifiers* (*learners*)
  - The resulting ensemble model is less susceptible to overfitting
- Adaboost ("adaptive boosting")
  - As long as the performance of each classifier is better than chance ($\epsilon < 0.5$), it is guaranteed to converge to a **better** classifier
- Can be extended to multi-class classification

# Boosting intuition

Training:

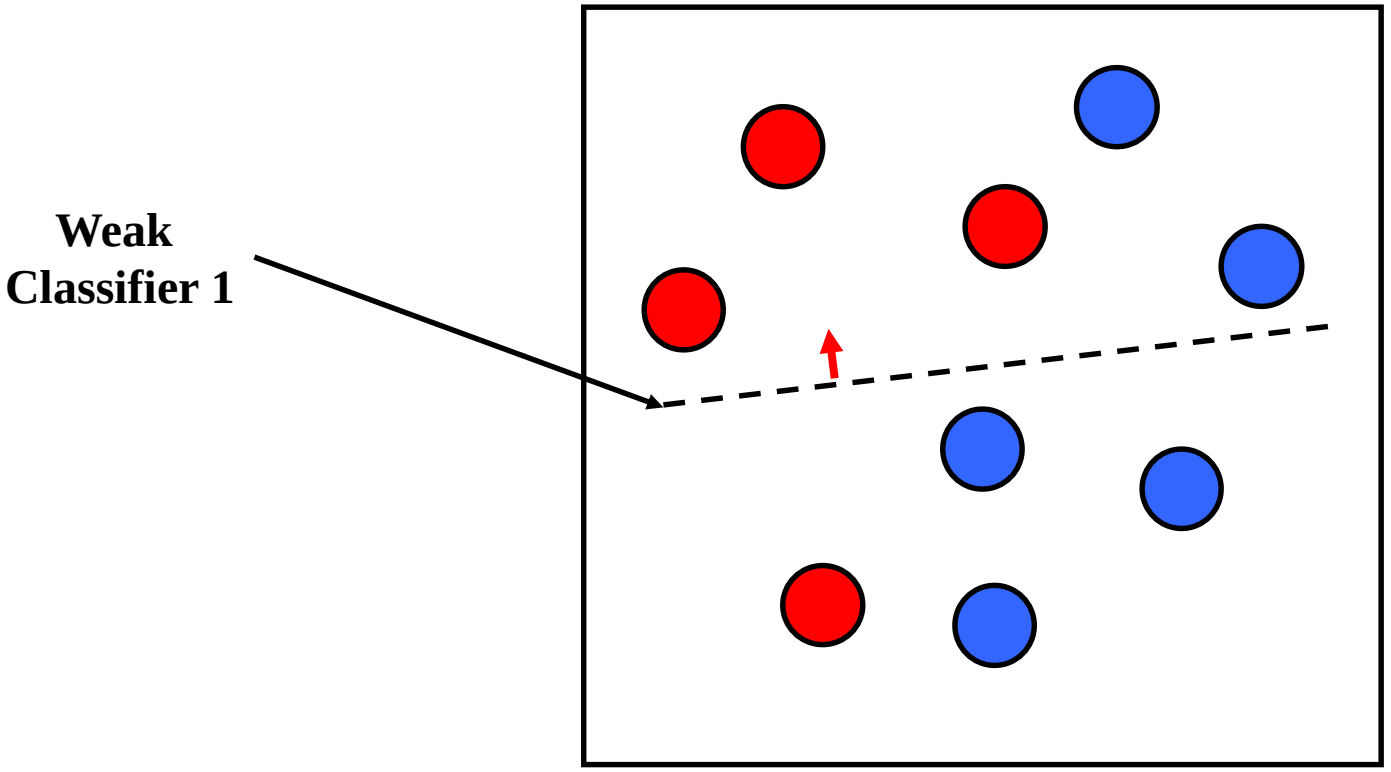


To increase the influence of the misclassified points, duplicate them (i.e., increase their relative weights by 100%)

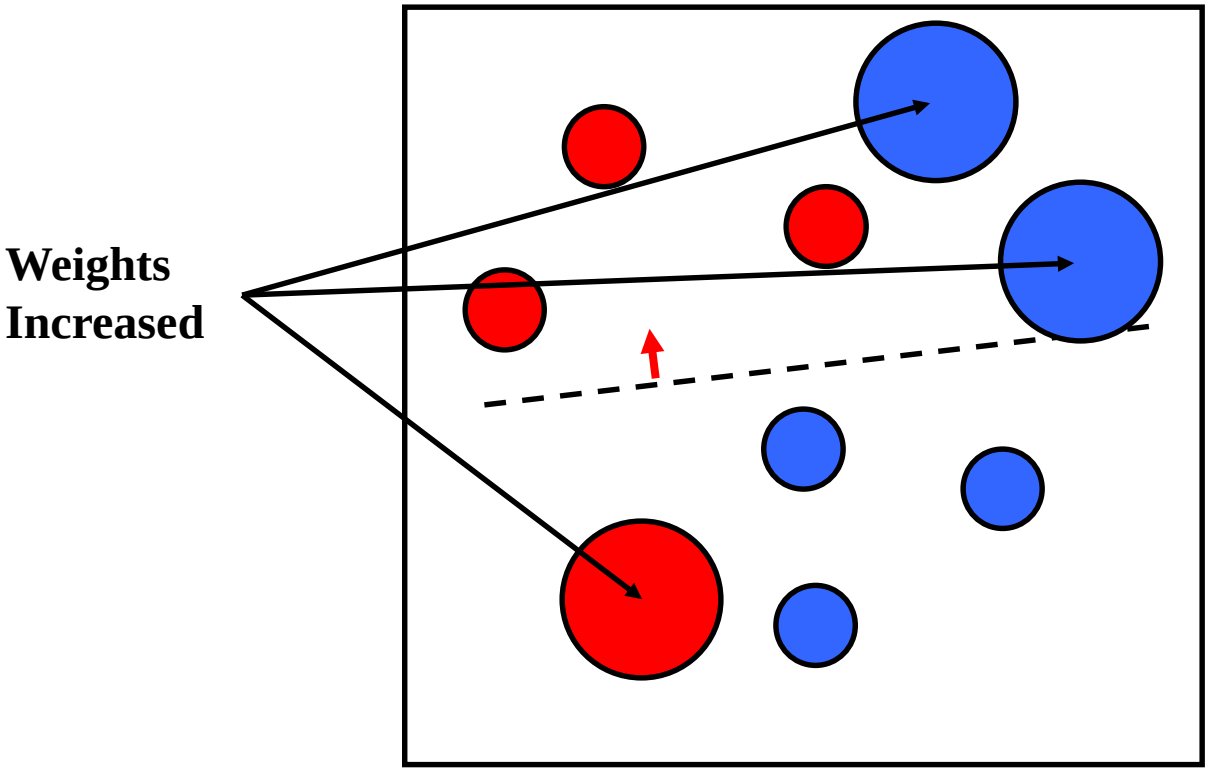
Or we could increase their weights by some other amount.

Assumption: we have a classifier model that takes into account the **weights** of data points

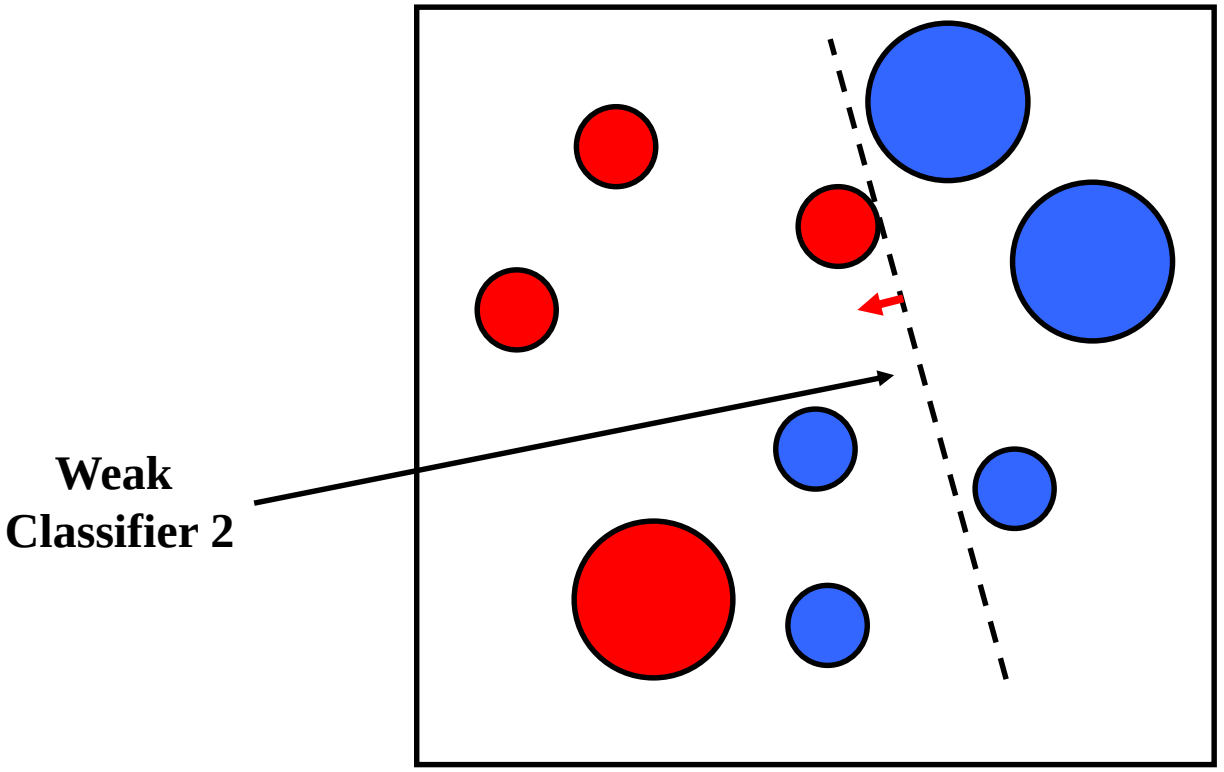
# Boosting illustration



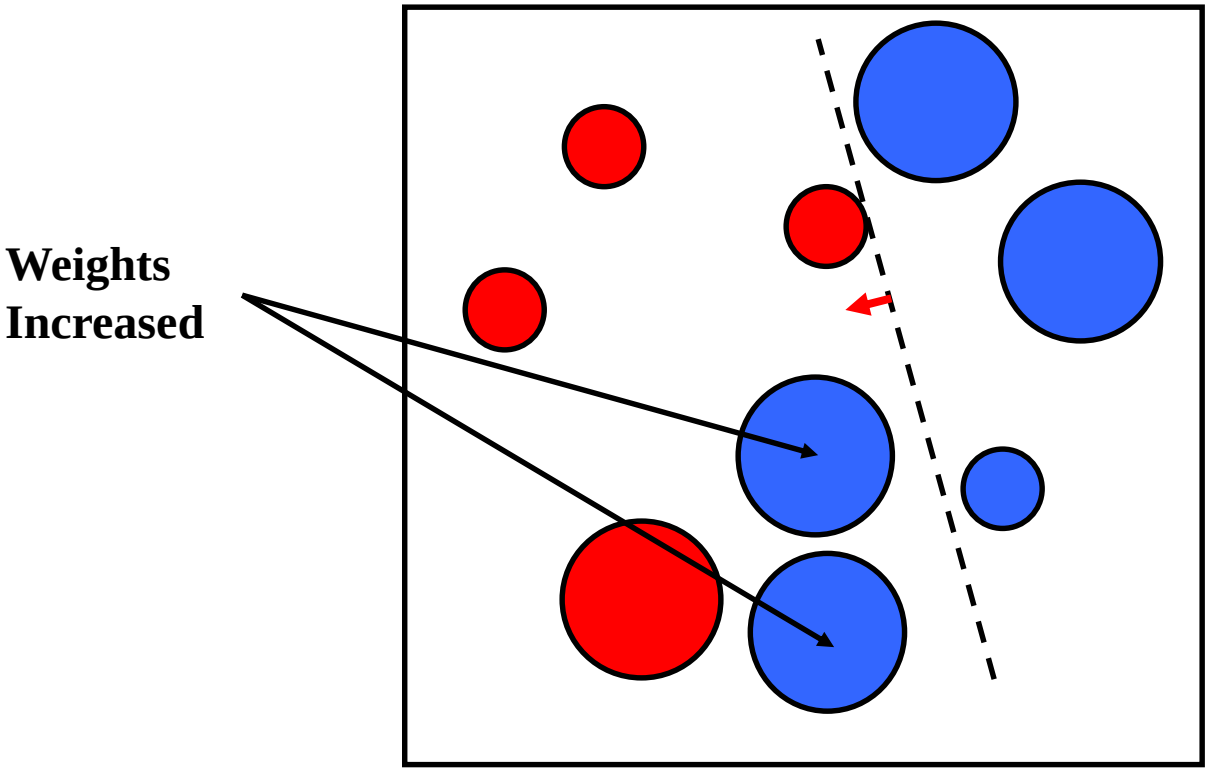
# Boosting illustration



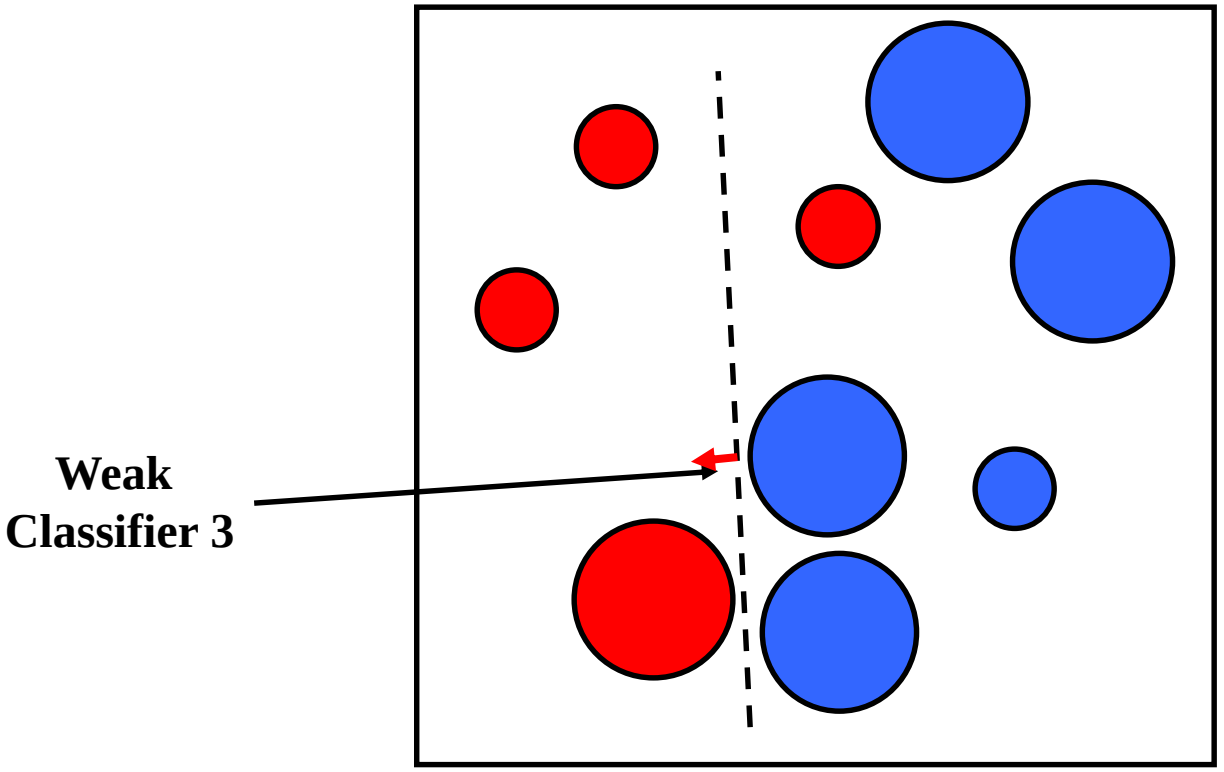
# Boosting illustration



# Boosting illustration

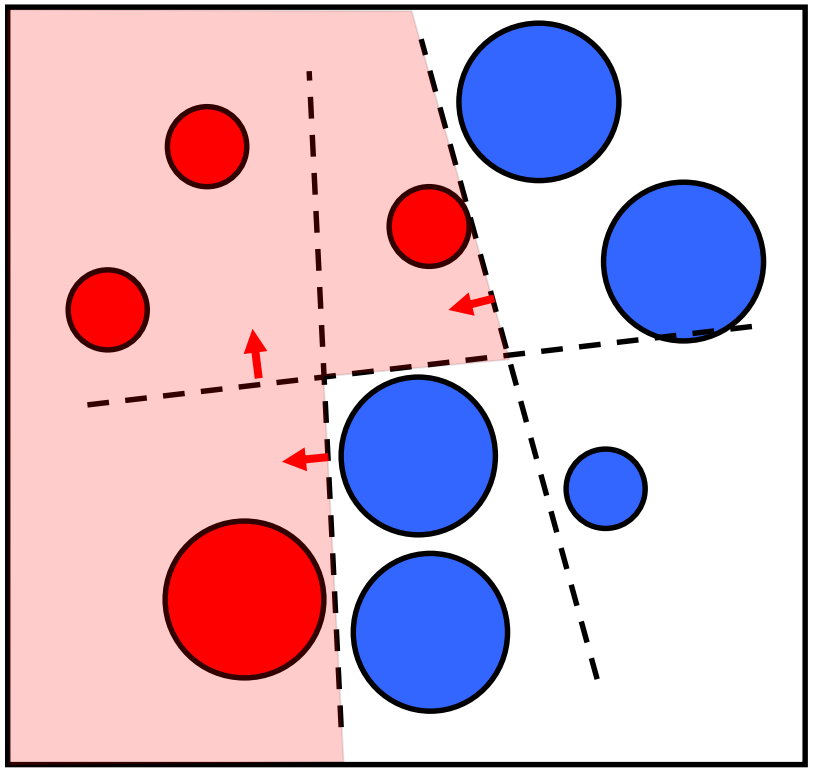


# Boosting illustration



# Boosting illustration

**Final classifier is a weighted combination of weak classifiers**

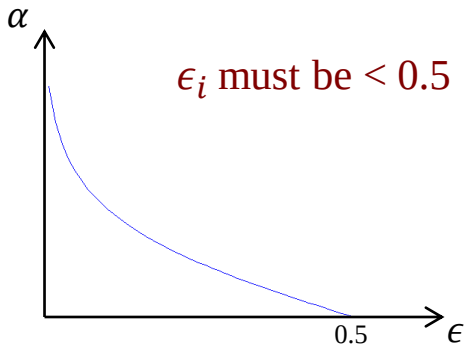


Majority vote of classifiers shown; however, weighted average is typically used

# Boosting

- Procedure:
  - Train a classifier; assign it a confidence factor $\alpha_i$ based on the error rate $\epsilon_i$

$$\alpha_i = \frac{1}{2}\ln\frac{1-\epsilon_i}{\epsilon_i}$$




$\epsilon_i$ must be < 0.5

  - Give misclassified instances a higher weight
    - Assign half of the total weight to the misclassified examples

Misclassified point

$$w' = \frac{w}{2\epsilon_i}$$

Correctly classified point

$$w' = \frac{w}{2(1-\epsilon_i)}$$

Weights will sum to 0.5+0.5=1

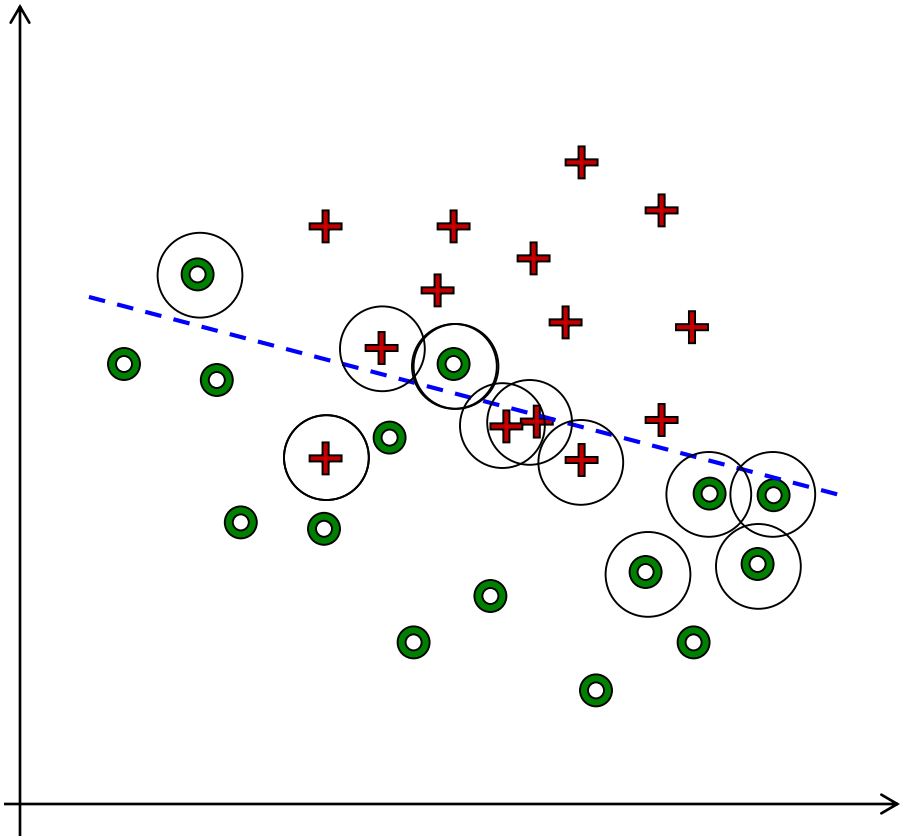
  - Repeat for *T* classifiers
  - The ensemble predictor is a weighted average of the models (rather than majority vote)

$$M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$$

  - Threshold for binary output

# Boosting example

29 points
Initial weights $w = 1/29 = 0.034$



Round 1:
2 + 4 = 6 errors

$$\epsilon_1 = \frac{6}{29} = 0.21$$

$$\alpha_1 = \frac{1}{2}\ln\frac{1-\epsilon_1}{\epsilon_1} = 0.67$$

$w' = \{\ 2.42w,\ 0.63w\ \}$

Round 2:
2 + 5 = 7 errors

$$\epsilon_2 = \frac{7}{29} = 0.24$$

$$\alpha_2 = \frac{1}{2}\ln\frac{1-\epsilon_2}{\epsilon_2} = 0.58$$

$w' = \{\ 2.07w,\ 0.66w\ \}$

Continue for $T$ iterations or until $\epsilon \geq 0.5$

# Boosting

Computational complexity: $O(TDK)$
(ensemble size x data size x dimensionality)

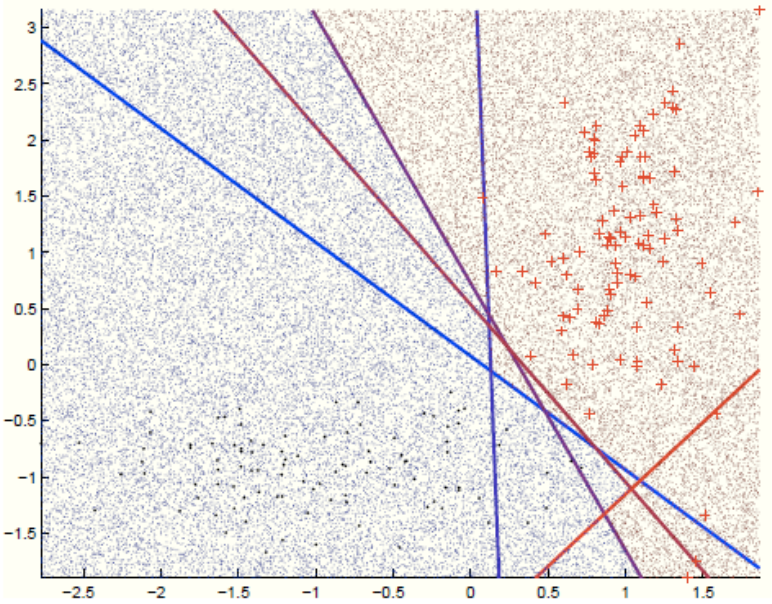
**Algorithm** Boosting$(D, T, \mathscr{A})$ – train an ensemble of binary classifiers from reweighted training sets.

**Input** : data set $D$; ensemble size $T$; learning algorithm $\mathscr{A}$.
**Output** : weighted ensemble of models.

$w_{1i} \leftarrow 1/|D|$ for all $x_i \in D$ ; // start with uniform weights
**for** $t = 1$ to $T$ **do**
  run $\mathscr{A}$ on $D$ with weights $w_{ti}$ to produce a model $M_t$;
  calculate weighted error $\epsilon_t$;
  **if** $\epsilon_t \geq 1/2$ **then**
    set $T \leftarrow t-1$ and break
  **end**
  $\alpha_t \leftarrow \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ ; // confidence for this model
  $w_{(t+1)i} \leftarrow \frac{w_{ti}}{2\epsilon_t}$ for misclassified instances $x_i \in D$ ; // increase weight
  $w_{(t+1)j} \leftarrow \frac{w_{tj}}{2(1-\epsilon_t)}$ for correctly classified instances $x_j \in D$ ; // decrease
**end**
**return** $M(x) = \sum_{t=1}^{T} \alpha_t M_t(x)$

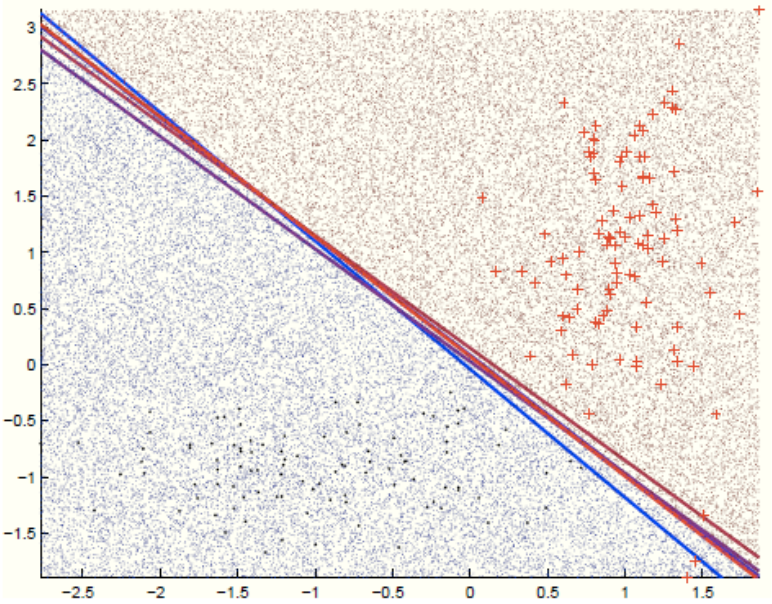
# Boosting vs. bagging

Boosting




Can achieve zero training error by focusing on the misclassifications

Bagging




With relatively large bootstrap sample sets, there tends to be little diversity in the learned models

# Ensemble methods: summary

- Model ensemble methods are meta-methods: they are ways to combine ML models via supervised learning, but they are agnostic to the kind of model being used
  - Simple linear classifier, Perceptron, SVM, neural network, etc. (weighted versions)
- They provide an opportunity to improve performance and (mostly) avoid over-fitting to the training data
  - Typically using randomization of some sort
- Bagging is essentially a variance-reduction technique (increase consistency), while boosting is essentially a bias reduction technique (increase accuracy)
- There are many other ensemble approaches in machine learning

# Machine learning experiments

# Machine learning experiments

- While there is a lot of important theory in machine learning (e.g., algorithms, proving convergence, proving performance bounds, establishing computability, etc.), it is a practical field, applying algorithms to data for practical use
- ML experiments are critical for confirming the various assumptions in a ML problem and establishing realistic expectations for performance
- Some key questions for ML experiments in a domain *D*:
  - How does a specific model perform on data from *D*?
  - Which of a set of models has the best performance on *D*?
  - How do models from learning algorithm *A* perform on *D*?
  - Which learning model *A* produces the best model(s) for domain *D*?
- How do we define performance?

False positive rate (FPR) = $\frac{FP}{N}$ = α

False negative rate (FNR) = $\frac{FN}{P}$ = β

True positive rate (TPR) = $\frac{TP}{P}$ = Sensitivity = Recall = 1 - β

True negative rate (TNR) = $\frac{TN}{N}$ = Specificity = 1 - α

Accuracy = $\frac{TP+TN}{P+N} = \left(\frac{P}{P+N}\right) TPR + \left(\frac{N}{P+N}\right) TNR$

Error rate = $\frac{FP+FN}{P+N}$

Precision = $\frac{TP}{\hat{P}}$

Accuracy + error rate = 1

Average recall = $\frac{TPR+TNR}{2}$

Contingency table:

| Predicted class $\hat{C}$ \ Actual class $C$ | 1 | 0 | |
|---|---|---|---|
| 1 | TP | FP | *Estimated positive* $\hat{P}$ |
| 0 | FN | TN | *Estimated negative* $\hat{N}$ |
| | *Positives* $P$ | *Negatives* $N$ | TOTAL |

# Example

| $\hat{c}$ \ $c$ | 1 | 0 |
|---|---|---|
| 1 | 50 | 0 |
| 0 | 50 | 10 |
| | P=100 | N=10 |

Accuracy = 60/110 = 0.55
Error rate = 1 – 0.55 = 0.45

Average recall = (0.5+1.0)/2 = 0.75

| $\hat{c}$ \ $c$ | 1 | 0 |
|---|---|---|
| 1 | 50 | 0 |
| 0 | 50 | 100 |
| | P=100 | N=100 |

Accuracy = 150/200 = 0.75
Error rate = 1 – 0.75 = 0.25

Average recall = (0.5+1.0)/2 = 0.75

# Performance measure

- If we choose accuracy (or error rate) as the measure of performance, we're making an assumption that the class distribution of the test set is representative of the real problem domain
- If we choose average recall, we're making an assumption that the real problem consists of uniform class distributions (equally likely)
- In general, when running an experiment, we should keep a record of a sufficient set of measurements to enable reproduction of the contingency table – in order to compute additional values later on if needed
  - Need four of the independent values (e.g., FP, FN, P, N)

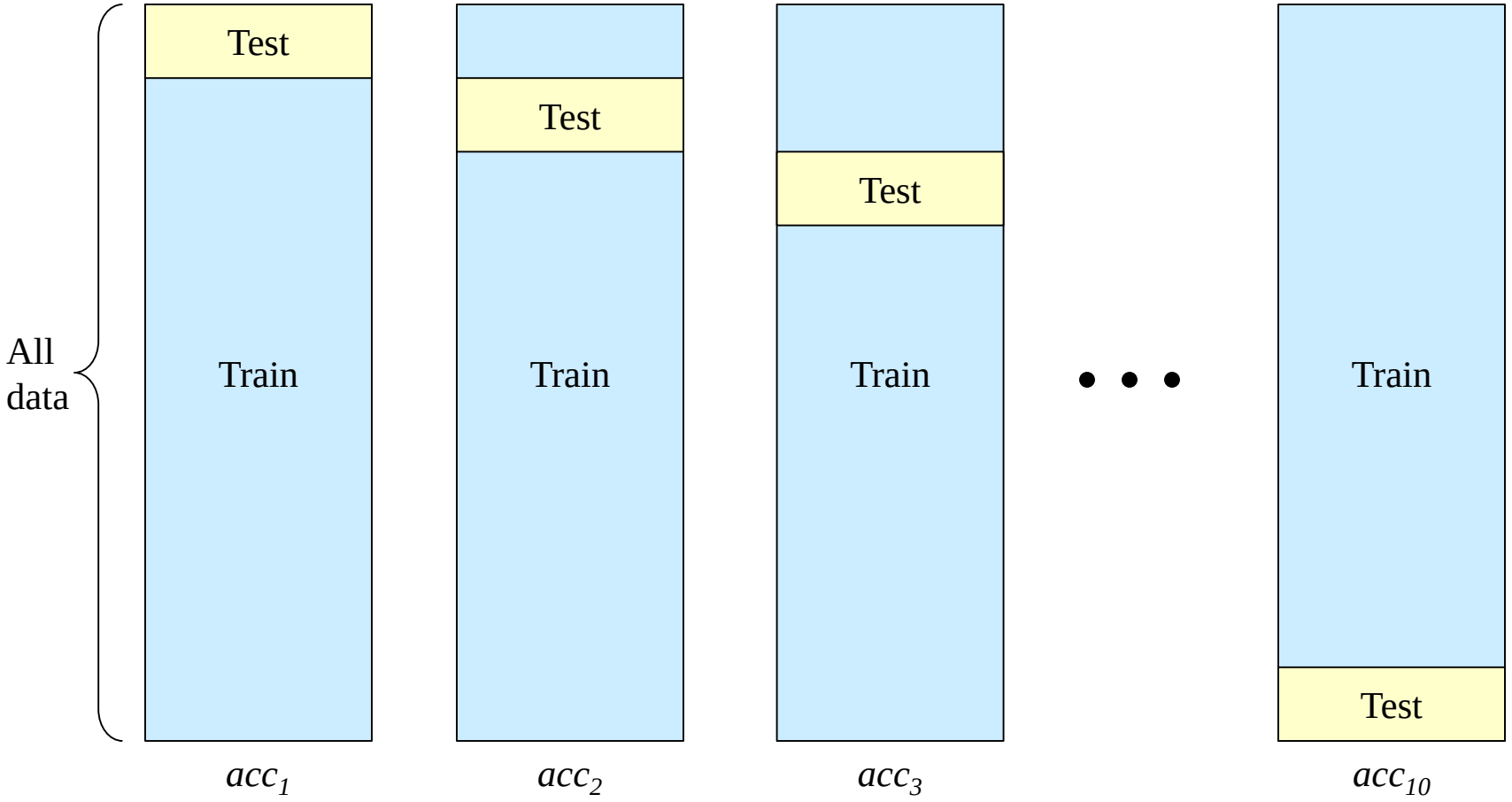
# How to measure performance

- So we decide on a performance measure (e.g., accuracy) and we evaluate it on test data, resulting in *acc*
  - How confident are we that our classifier really has an accuracy of *acc*?
  - I.e., if we repeated it on (different sets of) representative data an infinite number of times, what would the variance be?
    - We don't know, but we can estimate it by assuming certain things about the true distribution of the variable *acc*
- If we can estimate the performance measure *k* times, we reduce the sample variance by a factor of $1/\sqrt{k}$
- This is typically done in the experimental process of *cross-validation*

# Cross-validation

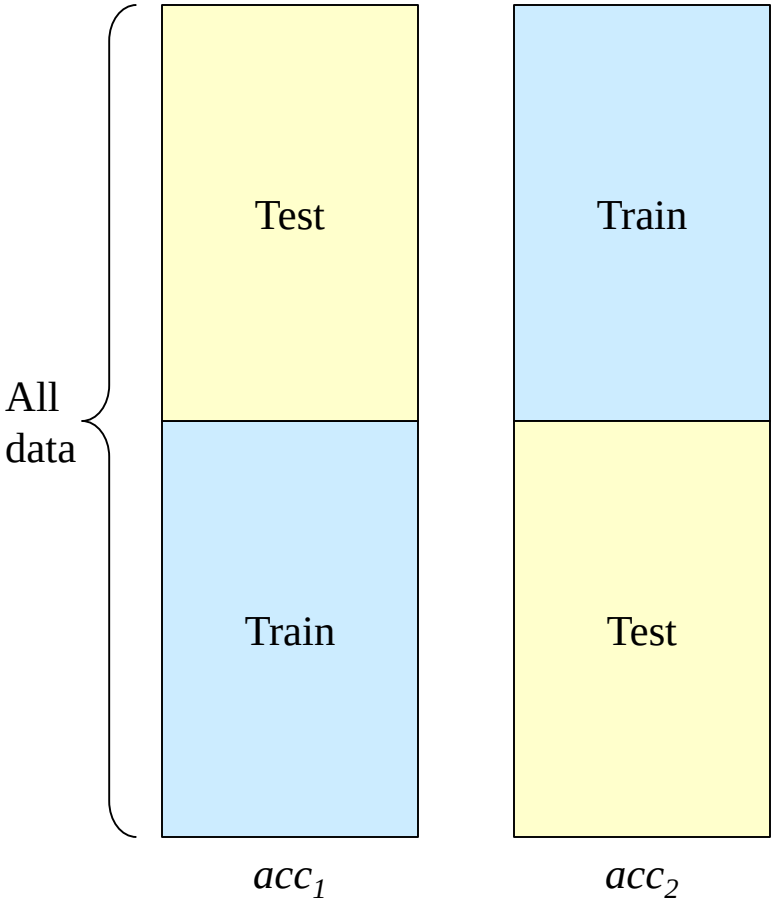
- The cross-validation process for experimental evaluation:
  - Randomly partition the experimental data into *k* parts ("folds")
  - Train the model on *k*–1 folds
  - Evaluate it on the remaining test fold
  - Repeat for a total of *k* times, evaluating on each test fold
  - Average the performance measure (e.g., accuracy), over the *k* trials
- This is *k*-fold cross-validation
  - Typically, $k = 10$, but other values are also used
    - $k = 2$ is quite common
  - Rule of thumb: folds should contain at last 30 instances
    - Implying $\geq 300$ instances in the data set for $k = 10$
  - If $k = n$ (the number of data points), this is leave-one-out cross-validation (a.k.a. the jackknife)

# Example: 10-fold cross-validation




$$acc = \text{Average}(acc_i)$$

# Example: 2-fold cross-validation




$$acc = ½ (acc_1 + acc_2)$$