Computer Science

CSC580: Principles of Data Science

Ensemble Methods

Jason Pacheco

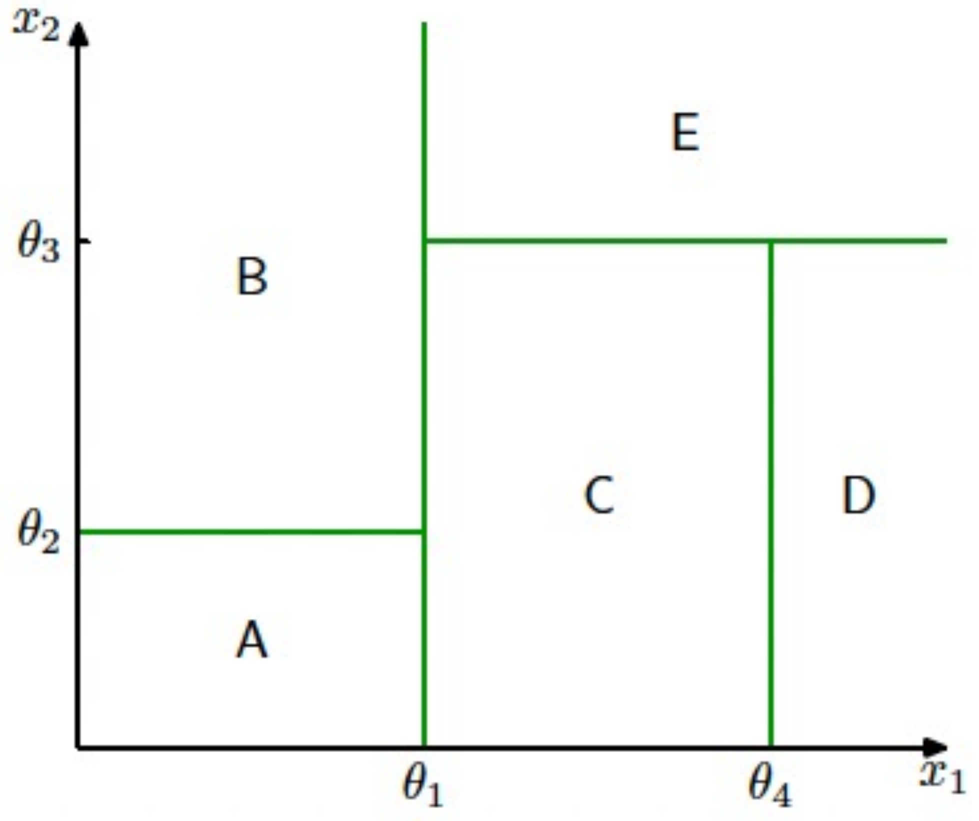
Ensemble Methods

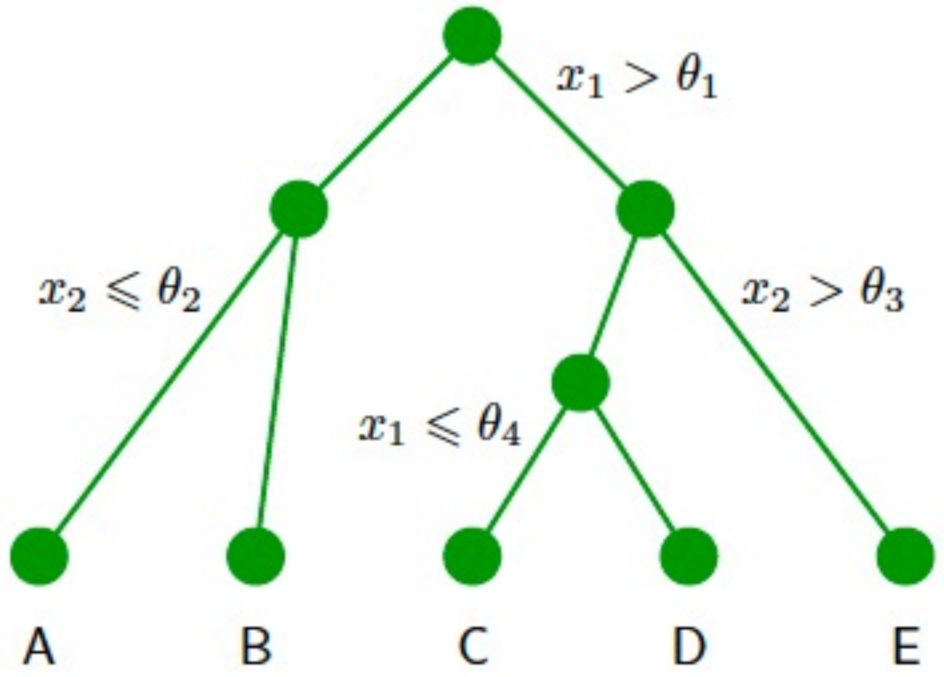
Ensemble Methods combine several base models to produce a single model with better predictive accuracy.

Motivation

- Groups of people often make better decisions than a single individual
- Combines opinions of multiple “learners” (models)
- Different models tend to make different (uncorrelated) errors
- Combining models averages out individual errors
- Difference in methods is in how *base learners* are combined into an *ensemble*

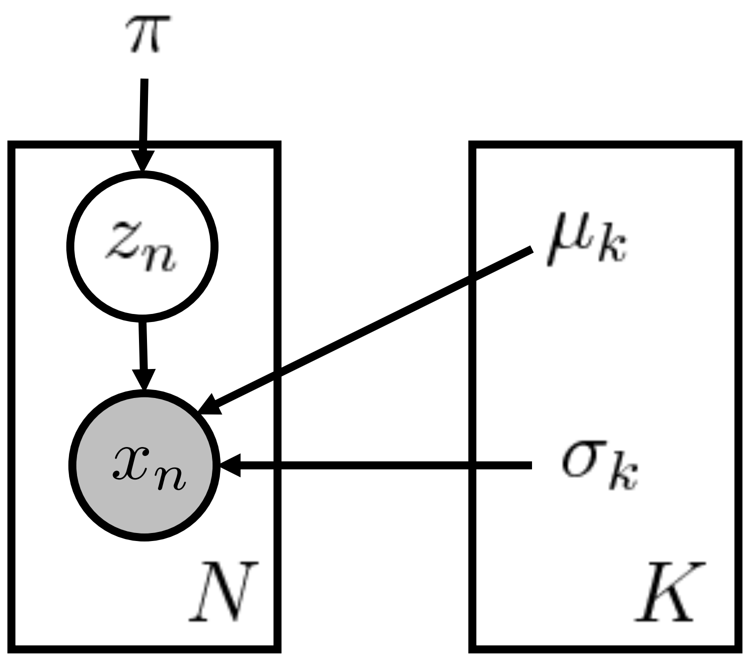
Tree-Based Models as Ensembles





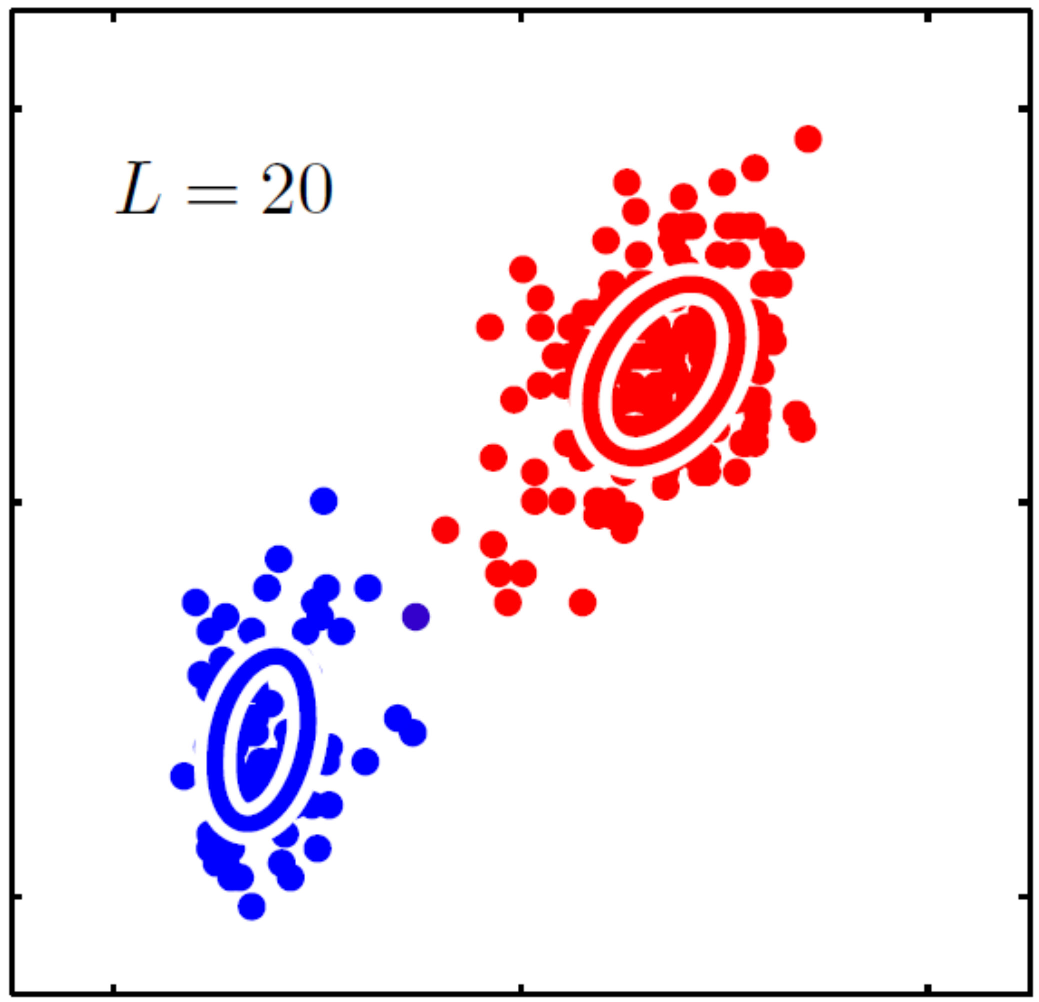
Assigns simple (constant prediction) model in regions

Example: Gaussian Mixture Model



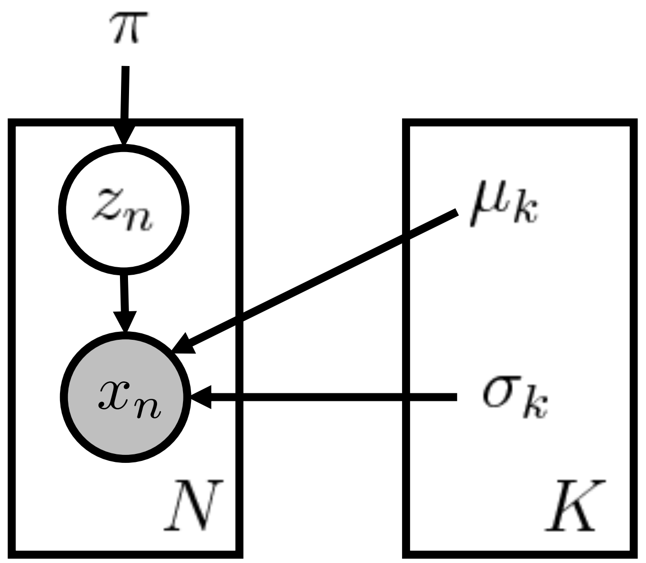
GMM

Can think of GMM as "soft" partitioning of model ensemble



Source: Bishop, PRML

GMM as Model Combination



GMM

The model distribution is given by,

$$p(\mathbf{x}) = \sum_{\mathbf{z}} p(\mathbf{x}, \mathbf{z}) = \sum_{k=1}^{K} \pi_k \mathcal{N}(\mathbf{x}|\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$$

For $\mathbf{X} = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ the probability of the data is,

$$p(\mathbf{X}) = \prod_{n=1}^{N} p(\mathbf{x}_n) = \prod_{n=1}^{N} \left[\sum_{\mathbf{z}_n} p(\mathbf{x}_n, \mathbf{z}_n) \right].$$

- Each datapoint is generated from a different (Gaussian) model
- Compare to *Bayesian model averaging*

Bayesian Model Averaging

- Consider a set of models $h = 1, \ldots, H$
- Each distribution is given by $p(\mathbf{X}|h)$ (e.g. Gaussian, Student-t, etc.)
- The distribution over the data is given by,

$$p(\mathbf{X}) = \sum_{h=1}^{H} p(\mathbf{X}|h)p(h).$$

- All data is generated from *one model*
- As size of dataset increases, uncertainty reduces and $p(h|\mathbf{X})$ concentrates
- Similar considerations apply for predictive distribution $p(x|X)$

Committee Methods

- Most models you have seen are *deterministic*
- If you train on the same data you get the same model
- Voting requires differing models
- Two ways to get difference in models
  - Change the learning algorithm
  - Change the training dataset

Voting Example

Train multiple classifiers (KNN, decision tree, etc.) call them,

$$f_1, f_2, \ldots, f_m$$

At test time, compute predictions,

$$\hat{y}_1 = f_1, \hat{y}_2 = f_2, \ldots, \hat{y}_m = f_m$$

- Assume binary labels $\hat{y} \in \{0, 1\}$
- Count number of +1's among y's
- If there are more +1's then vote +1
- Otherwise vote -1

Voting Classifiers

Very unlikely that all classifiers will make the same mistakes

As long as each error is made by a *minority of models* then you will **achieve an optimal classifier!**

Unfortunately, inductive biases of different learning algorithms are highly correlated…

…but ensembles can still be helpful for *reducing variance*

Voting Methods

Naturally extends to multi-class classification

Voting doesn’t make sense in:

- Regression
- Ranking
- Etc.

You will rarely see the same output from multiple models

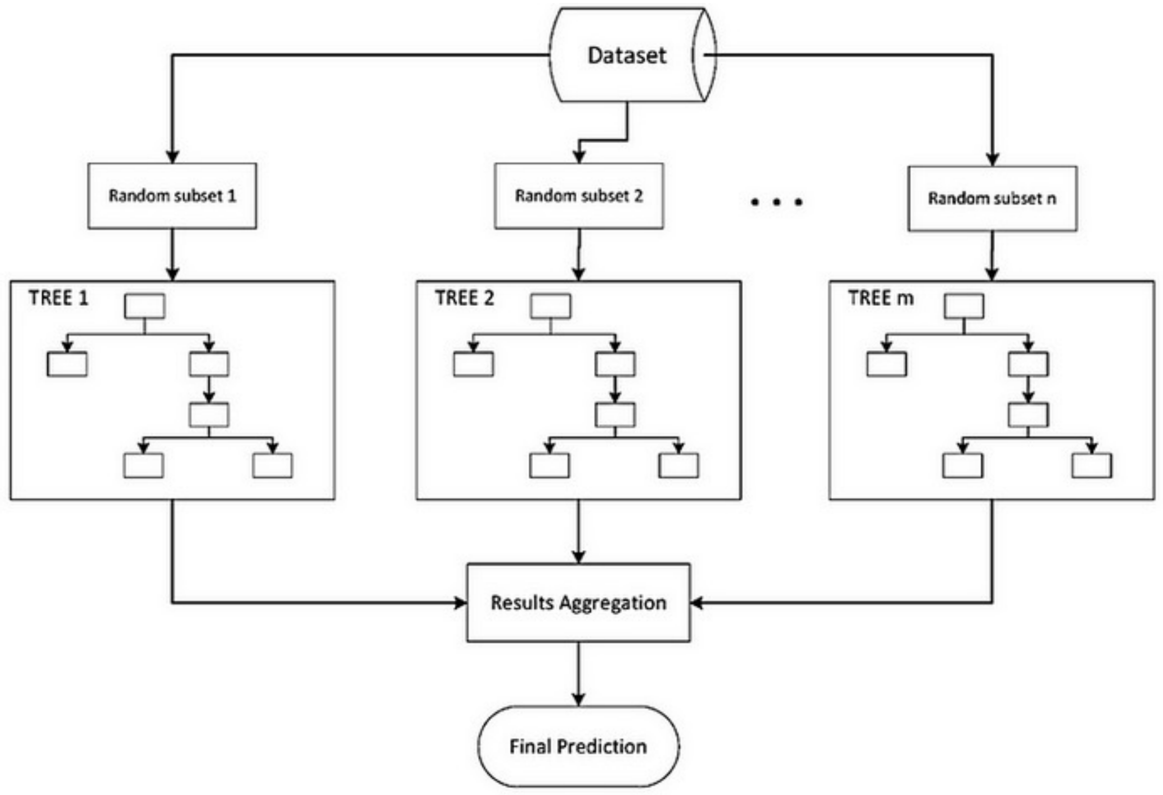
- E.g. two different regression models
- For regression: Take the mean / median

Bagging

Voting methods combine multiple models to produce randomness

Instead of multiple models, use a single model trained on different datasets

Bagging = "Bootstrap Aggregating" uses *bootstrap resampling* to produce multiple datasets from a single training set



Recall: Bootstrap

Suppose we observe data $X_1, X_2, \ldots, X_n \sim P(X;\theta)$:

1. Sample new "dataset" $X_1^*, \ldots, X_m^*$ uniformly from $X_1, \ldots, X_n$ **with replacement**

2. Compute estimate $\hat{\theta}_m(X_1^*, \ldots, X_m^*)$

2. Repeat B times to get set of estimators $\hat{\theta}_{m,1}, \hat{\theta}_{m,2}, \ldots, \hat{\theta}_{m,B}$

3. Compute sample mean and sample variance of estimators,

$$\bar{\theta}_{\text{boot}} = \frac{1}{B}\sum_{b=1}^{B} \hat{\theta}_{m,b} \qquad \sigma_{\text{boot}}^2 = \frac{1}{B}\sum_{b=1}^{B}(\hat{\theta}_{m,b} - \bar{\theta}_{\text{boot}})^2$$

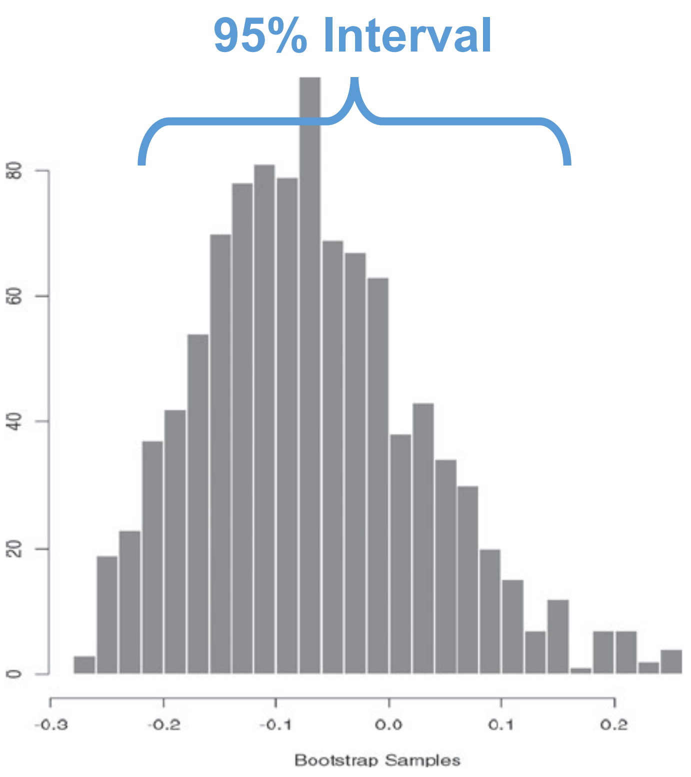
3. 95% Confidence Interval: $\bar{\theta}_{\text{boot}} \pm 2\sigma_{\text{boot}}$

Assumes Normally-distributed estimates $\hat{\theta}_m$**.**

Bootstrap Example

Eight subjects who used medical patches to infuse a hormone into the blood using three treatments: placebo, old-patch, new-patch

subject	placebo	old	new	old − placebo	new − old
1	9243	17649	16449	8406	-1200
2	9671	12013	14614	2342	2601
3	11792	19979	17274	8187	-2705
4	13357	21816	23798	8459	1982
5	9055	13850	12560	4795	-1290
6	6290	9806	10157	3516	351
7	12412	17208	16570	4796	-638
8	18806	29044	26325	10238	-2719



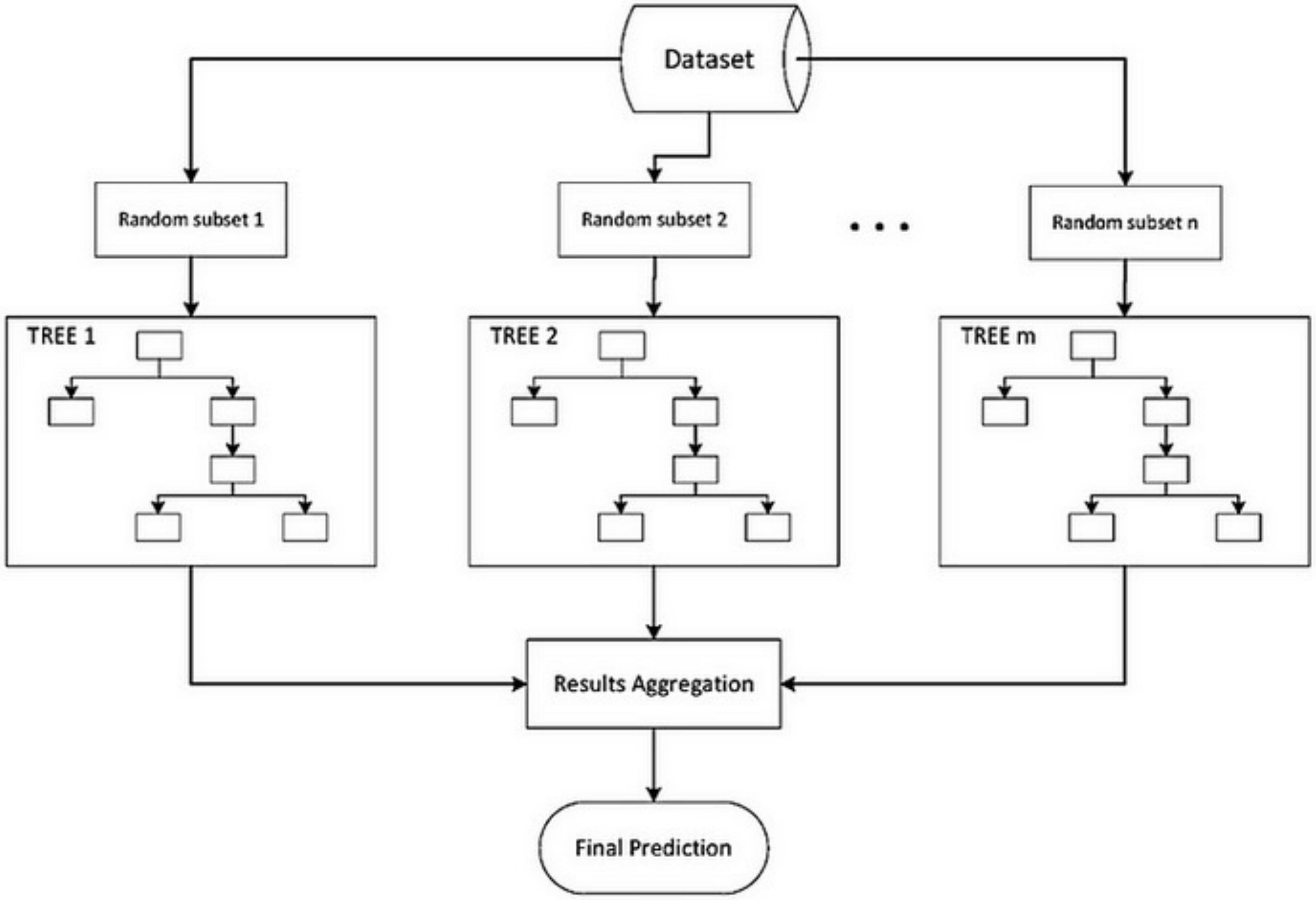
Estimate whether relative efficacy is the same under new drug,

$$\theta = \frac{\mathbf{E}[\text{new} - \text{old}]}{\mathbf{E}[\text{old} - \text{placebo}]}$$

Bootstrap B=1,000 samples yields 95% confidence interval,

$$\theta \in (-0.24, 0.15)$$

Bagging



https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ec3f9fef5f

Boosting Error Analysis

Let *M* models be trained on bootstrap data with committee predictions,

$$y_{\mathrm{COM}}(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} y_m(\mathbf{x}).$$

Suppose *h*(*x*) is the true regression we wish to predict and each model,

$$y_m(\mathbf{x}) = h(\mathbf{x}) + \epsilon_m(\mathbf{x}).$$

Average sum-of-squares error then takes the form,

$$\mathbb{E}_{\mathbf{x}}\left[\{y_m(\mathbf{x}) - h(\mathbf{x})\}^2\right] = \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})^2\right]$$

Where expectation is with respect to the input data *x*

Boosting Error Analysis

Average error of individual models is therefore,

$$E_{\mathrm{AV}} = \frac{1}{M} \sum_{m=1}^{M} \mathbb{E}_{\mathbf{x}} \left[\epsilon_m(\mathbf{x})^2 \right]$$

Average error from the committee is given by,

$$\begin{aligned} E_{\mathrm{COM}} &= \mathbb{E}_{\mathbf{x}} \left[\left\{ \frac{1}{M} \sum_{m=1}^{M} y_m(\mathbf{x}) - h(\mathbf{x}) \right\}^2 \right] \\ &= \mathbb{E}_{\mathbf{x}} \left[\left\{ \frac{1}{M} \sum_{m=1}^{M} \epsilon_m(\mathbf{x}) \right\}^2 \right] \end{aligned}$$

Boosting Error Analysis

Let’s assume errors are zero-mean and uncorrelated,

$$\begin{aligned} \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\right] &= 0 \\ \mathbb{E}_{\mathbf{x}}\left[\epsilon_m(\mathbf{x})\epsilon_l(\mathbf{x})\right] &= 0, \qquad m \neq l \end{aligned}$$

Then we obtain,

$$E_{\text{COM}} = \frac{1}{M} E_{\text{AV}}.$$

- Committee reduces error by a factor of M
- Relies on errors being uncorrelated, but errors are often correlated
- Even with correlated errors we can still show,

$$E_{\text{COM}} \leqslant E_{\text{AV}}.$$

Boosting

- Taking a **weak learner** and producing a **strong learner**
- Start with a crummy learning algorithm (weak learner)
- Retrain it and upweight examples that it makes errors on
- Do this again…
- …and again…

Boosting

Define a **strong learning algorithm** $\mathcal{L}$ as:

- Given a desired error rate ϵ
- A failure probability δ
- And “enough” training data
- With high probability (at least $1 - \delta$)
- $\mathcal{L}$ learns a classifier f that has error at most ϵ

- Known as *probably almost correct (PAC)* learning
- But directly building a strong algorithm can be hard
- Instead, build a weak learner $\mathcal{W}$ and *boost* it

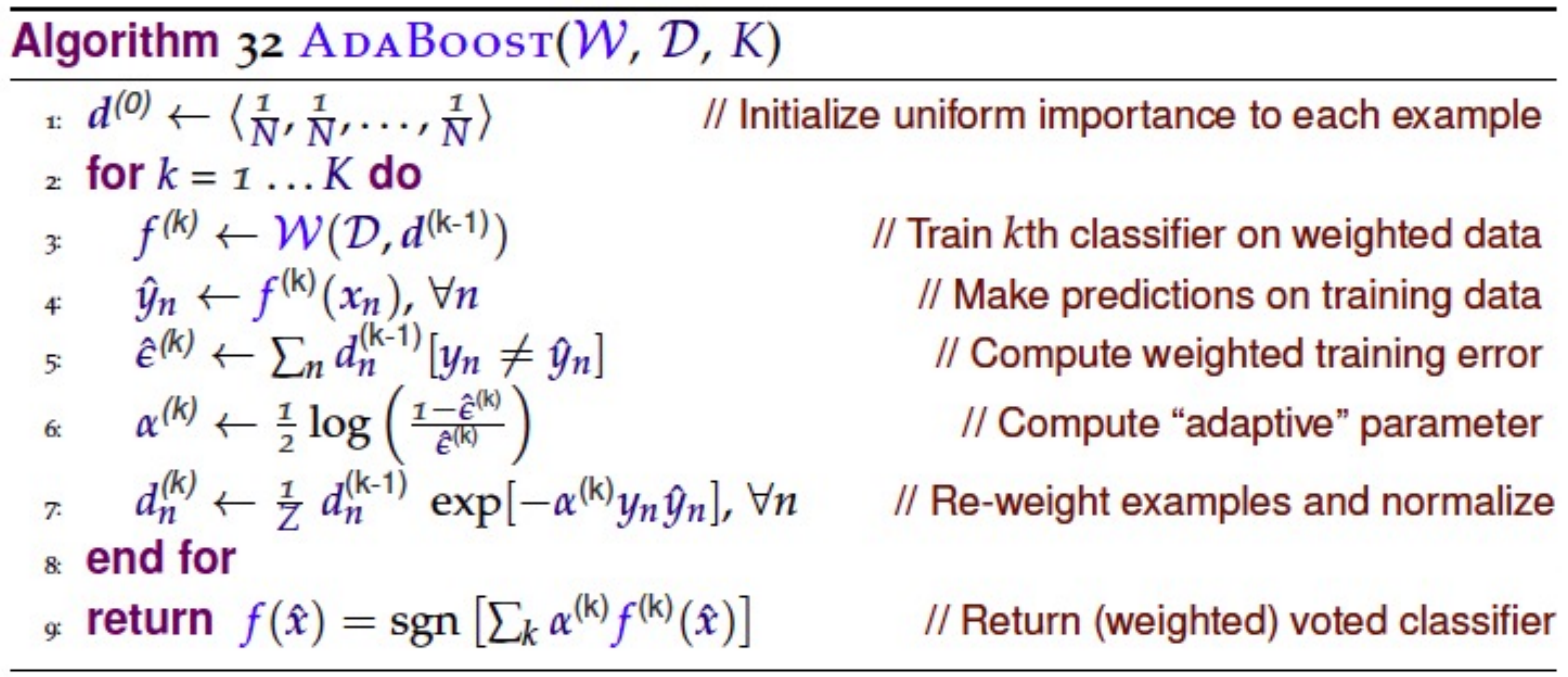
AdaBoost

- Short for “Adaptive Boosting”
- Runs in polynomial time
- Does not have a large number of hyperparameters
- Typically *adapts* to the data that you give it

Intuition Study for an exam using a past exam.

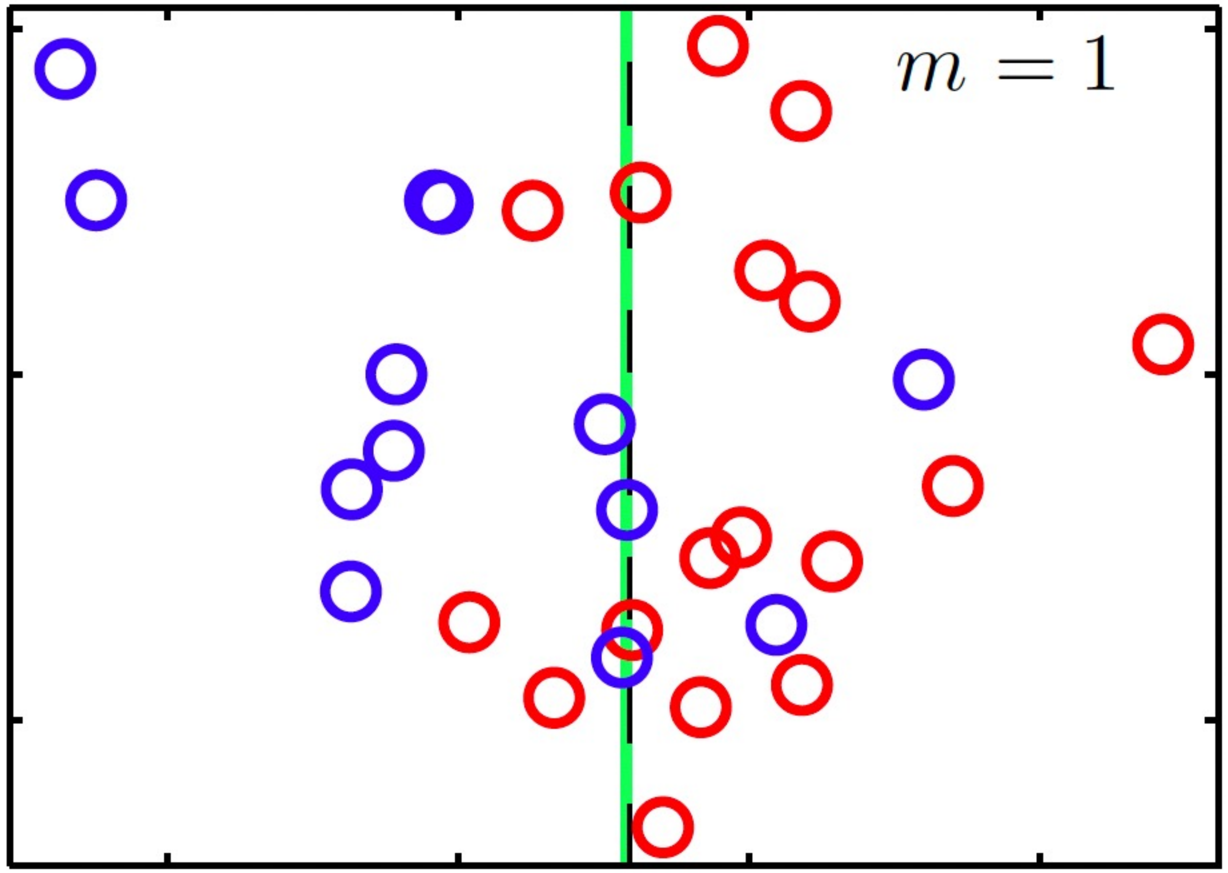
- Grade your past exam
- Retake exam and pay less attention to questions you got right
- Pay more attention to questions that you got wrong
- Regrade and repeat, and repeat, and repeat…

AdaBoost

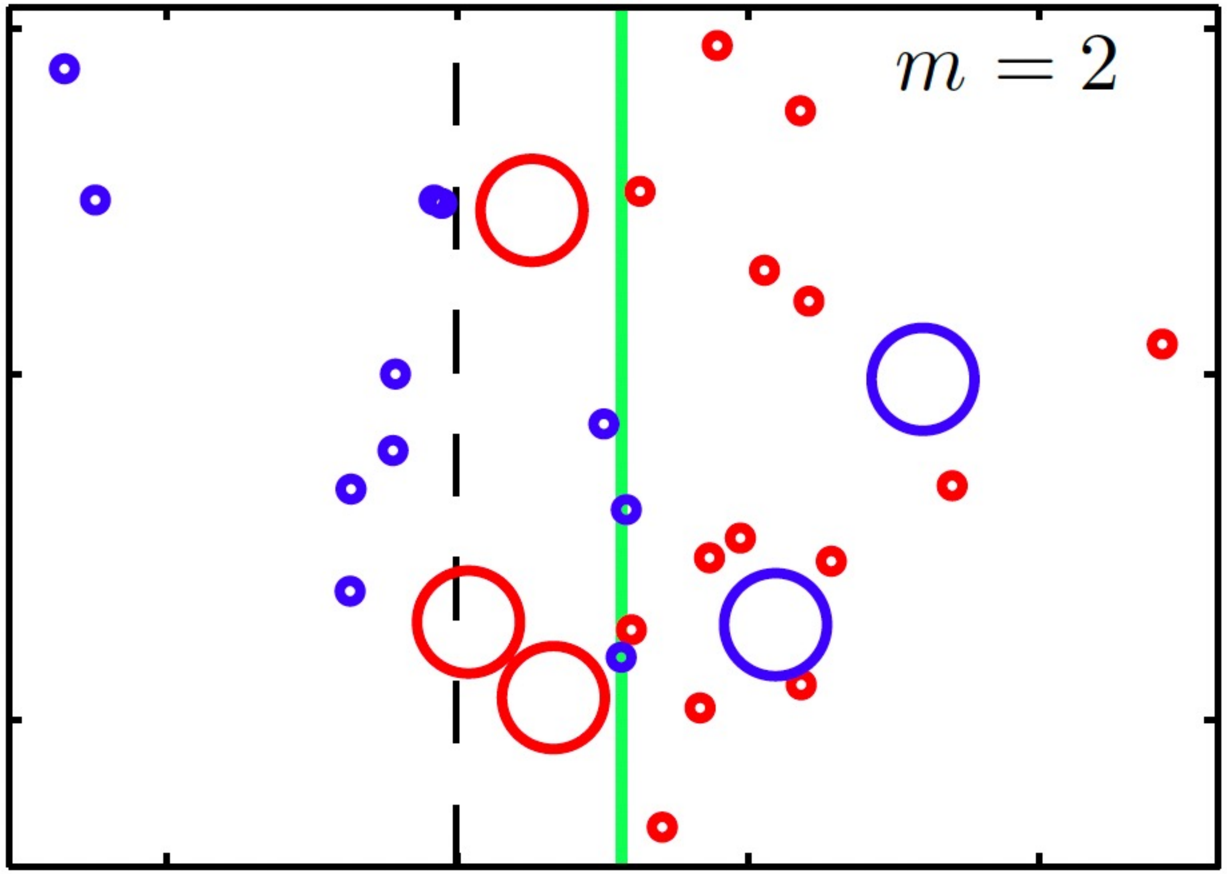


```
Algorithm 32 AdaBoost($\mathcal{W}$, $\mathcal{D}$, $K$)
$d^{(0)} \leftarrow \langle \frac{1}{N}, \frac{1}{N}, \ldots, \frac{1}{N} \rangle$        // Initialize uniform importance to each example
for $k = 1 \ldots K$ do
    $f^{(k)} \leftarrow \mathcal{W}(\mathcal{D}, d^{(k-1)})$        // Train $k$th classifier on weighted data
    $\hat{y}_n \leftarrow f^{(k)}(x_n), \forall n$        // Make predictions on training data
    $\hat{\epsilon}^{(k)} \leftarrow \sum_n d_n^{(k-1)}[y_n \neq \hat{y}_n]$        // Compute weighted training error
    $\alpha^{(k)} \leftarrow \frac{1}{2} \log\left(\frac{1-\hat{\epsilon}^{(k)}}{\hat{\epsilon}^{(k)}}\right)$        // Compute "adaptive" parameter
    $d_n^{(k)} \leftarrow \frac{1}{Z} d_n^{(k-1)} \exp[-\alpha^{(k)} y_n \hat{y}_n], \forall n$        // Re-weight examples and normalize
end for
return $f(\hat{x}) = \mathrm{sgn}\left[\sum_k \alpha^{(k)} f^{(k)}(\hat{x})\right]$        // Return (weighted) voted classifier
```

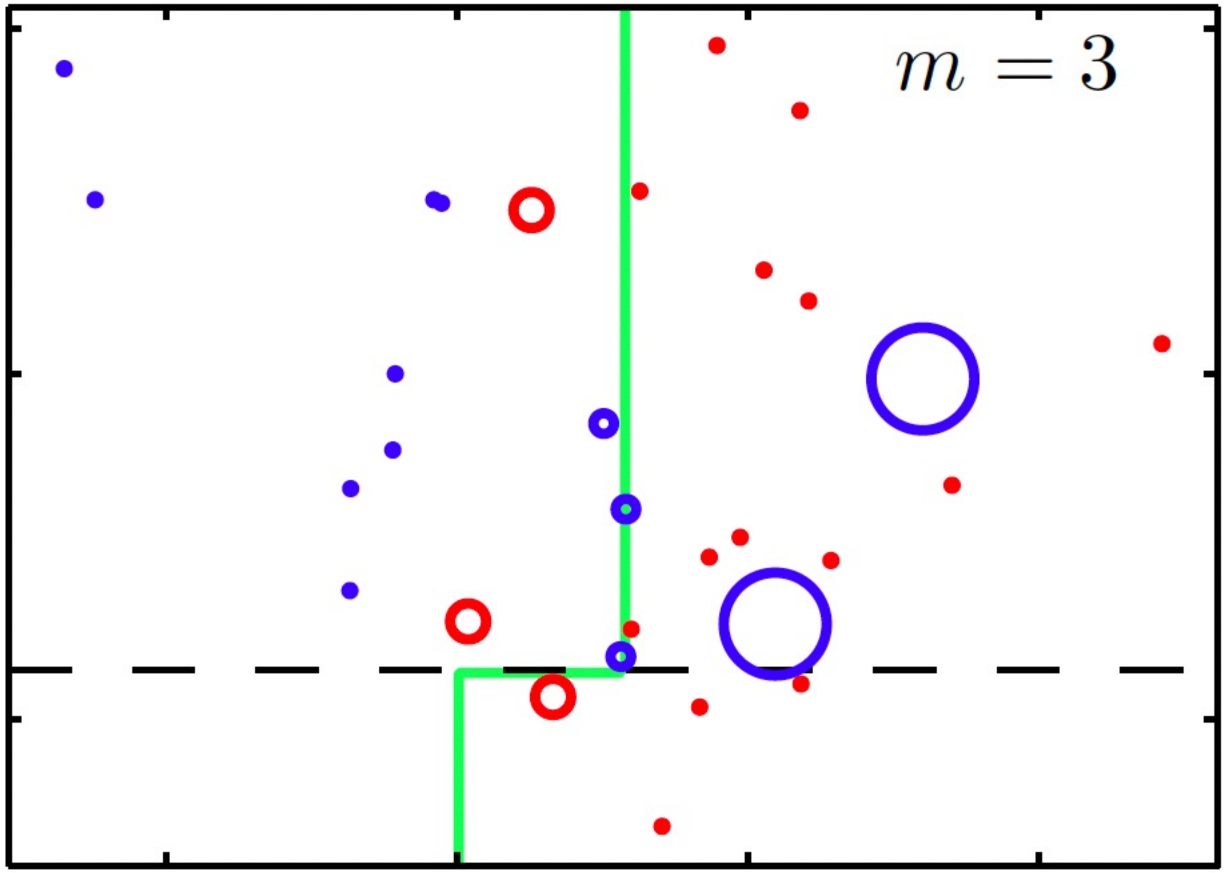
Boosting



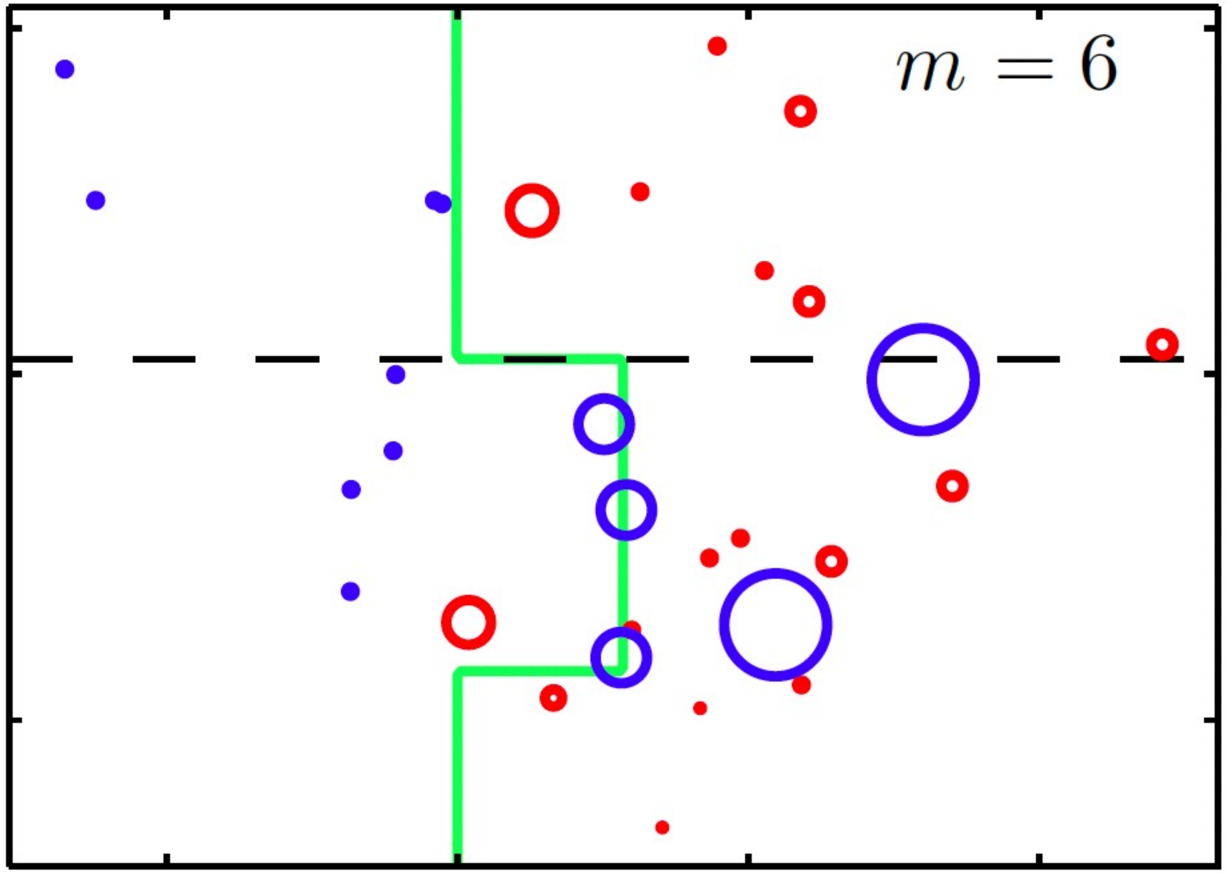
Boosting



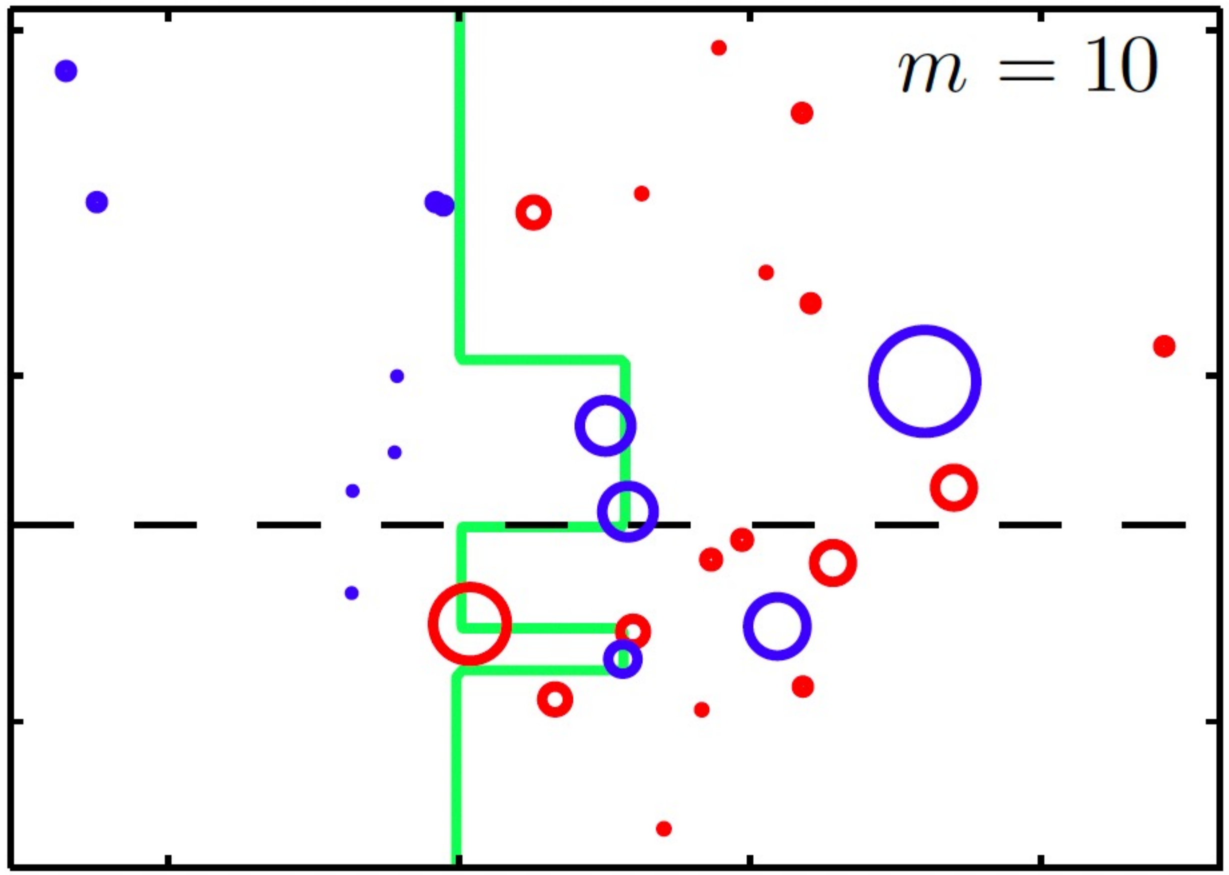
Boosting



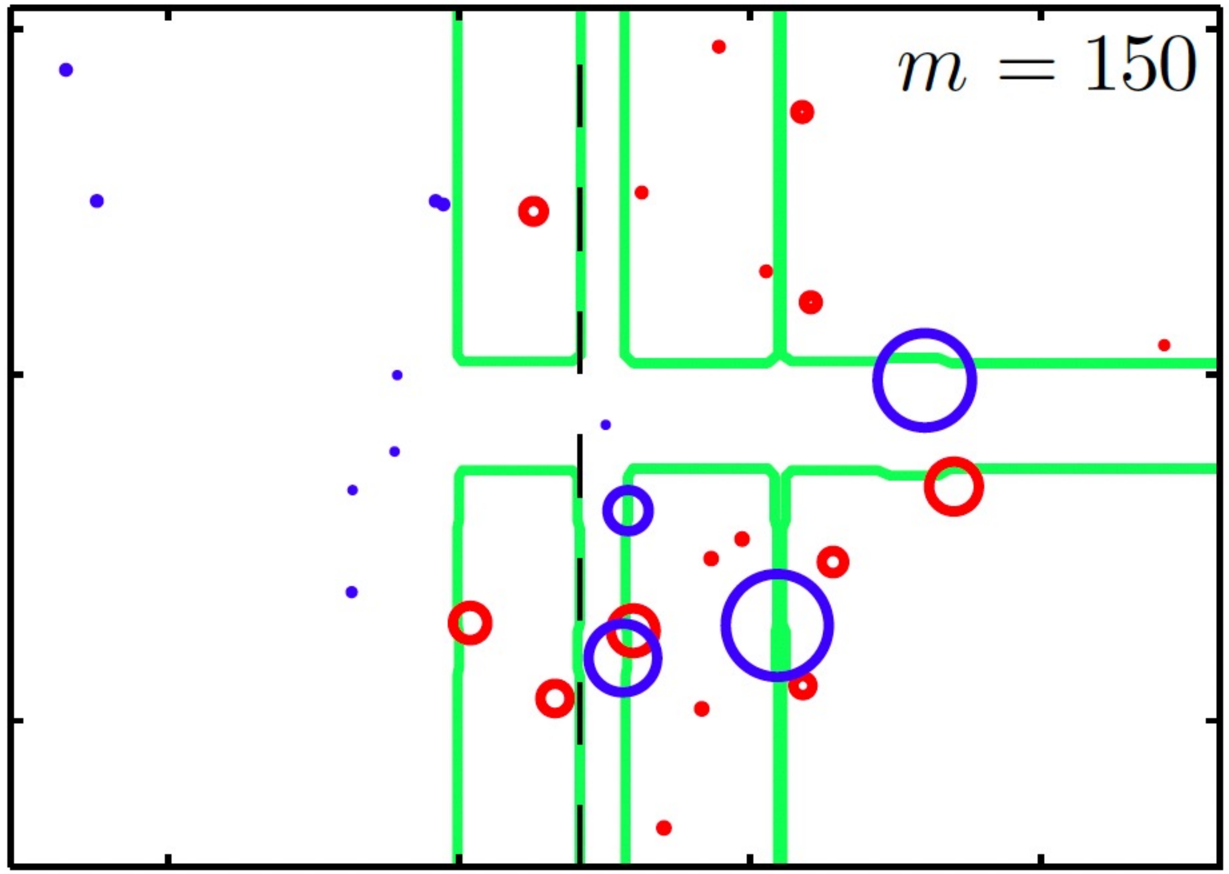
Boosting



Boosting



Boosting



AdaBoost

Consider the exponential error function,

$$E = \sum_{n=1}^{N} \exp\{-y_n \hat{y}_n\}$$

Where $\hat{y}_n$ is a linear combination of base classifiers,

$$\hat{y}_n = \frac{1}{2} \sum_{k=1}^{m} \alpha_k f_\ell(x_n)$$

We want to minimize *E* with respect to base classifiers $f_\ell(x)$ and weights α_ℓ ,

$$E = \sum_{n=1}^{N} \exp\left\{-y_n f_{k-1}(x_n) - \frac{1}{2}\alpha_k y_n \hat{y}_k\right\}$$

$$= \sum_{n=1}^{N} d_n^k \exp\left\{-\frac{1}{2}\alpha_k y_n \hat{y}_k\right\}$$

AdaBoost

So the error function is given by,

Weights d_n^k can be viewed as constants in optimization

$$E = \sum_{n=1}^{N} d_n^k \exp\left\{-\frac{1}{2}\alpha_k y_n \hat{y}_k\right\}$$

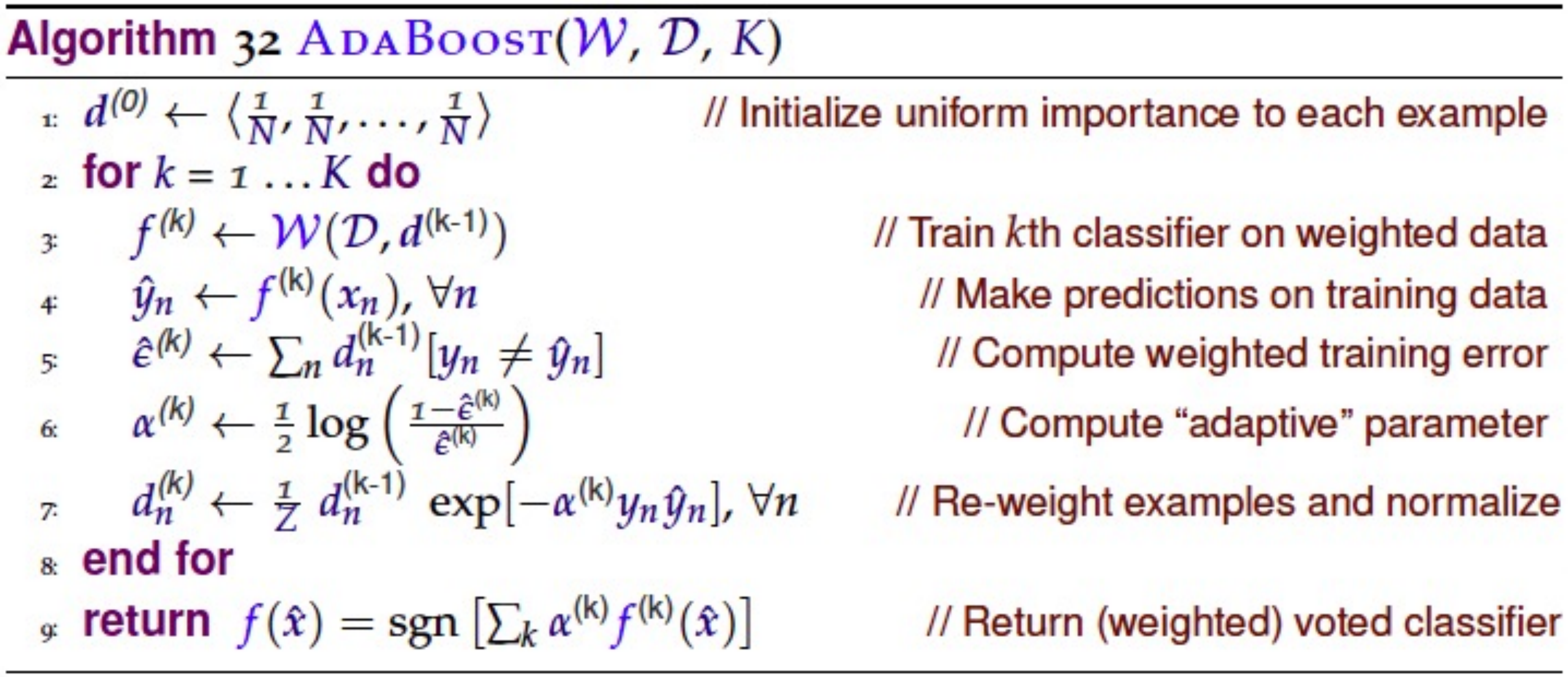
Given $\hat{y}_k$ and α_k the weights can be updated sequentially as,

$$d_n^{k+1} = d_n^k \exp\left\{-\frac{1}{2}\alpha_k y_n \hat{y}_k\right\}$$

- This recovers the weight update in AdaBoost
- Similar analysis can be used to recover updates for α_k
- This shows that AdaBoost minimizes exponential error

$$E = \sum_{n=1}^{N} \exp\{-y_n \hat{y}_n\}$$

AdaBoost



Algorithm 32 AdaBoost($\mathcal{W}$, $\mathcal{D}$, K)

```
1: $d^{(0)} \leftarrow \langle \frac{1}{N}, \frac{1}{N}, \dots, \frac{1}{N} \rangle$    // Initialize uniform importance to each example
2: for $k = 1 \dots K$ do
3:     $f^{(k)} \leftarrow \mathcal{W}(\mathcal{D}, d^{(k-1)})$    // Train $k$th classifier on weighted data
4:     $\hat{y}_n \leftarrow f^{(k)}(x_n), \forall n$    // Make predictions on training data
5:     $\hat{\epsilon}^{(k)} \leftarrow \sum_n d_n^{(k-1)}[y_n \neq \hat{y}_n]$    // Compute weighted training error
6:     $\alpha^{(k)} \leftarrow \frac{1}{2} \log \left( \frac{1-\hat{\epsilon}^{(k)}}{\hat{\epsilon}^{(k)}} \right)$    // Compute "adaptive" parameter
7:     $d_n^{(k)} \leftarrow \frac{1}{Z} d_n^{(k-1)} \exp[-\alpha^{(k)} y_n \hat{y}_n], \forall n$    // Re-weight examples and normalize
8: end for
9: return $f(\hat{x}) = \mathrm{sgn}\left[\sum_k \alpha^{(k)} f^{(k)}(\hat{x})\right]$    // Return (weighted) voted classifier
```

Example: AdaBoost Decision Stump

Decision Stump Tree with a single question (e.g. ”is feature x on?”)

All week decision stumps must have the form,

$$f(\boldsymbol{x}) = s(2x_d - 1) \quad \text{where} \quad s \in \{\pm 1\}$$

Now let f_k single feature and s_k its sign, then:

$$f(\boldsymbol{x}) = \operatorname{sgn}\left[\sum_k \alpha_k f^{(\mathrm{k})}(\boldsymbol{x})\right] = \operatorname{sgn}\left[\sum_k \alpha_k s_k (2x_{f_k} - 1)\right]$$

$$= \operatorname{sgn}\left[\sum_k 2\alpha_k s_k x_{f_k} - \sum_k \alpha_k s_k\right]$$

Example: AdaBoost Decision Stump

$$f(\boldsymbol{x}) = \operatorname{sgn}\left[\boldsymbol{w} \cdot \boldsymbol{x} + b\right]$$

$$\text{where } w_d = \sum_{k:f_k=d} 2\alpha_k s_k \quad \text{and} \quad b = -\sum_k \alpha_k s_k$$

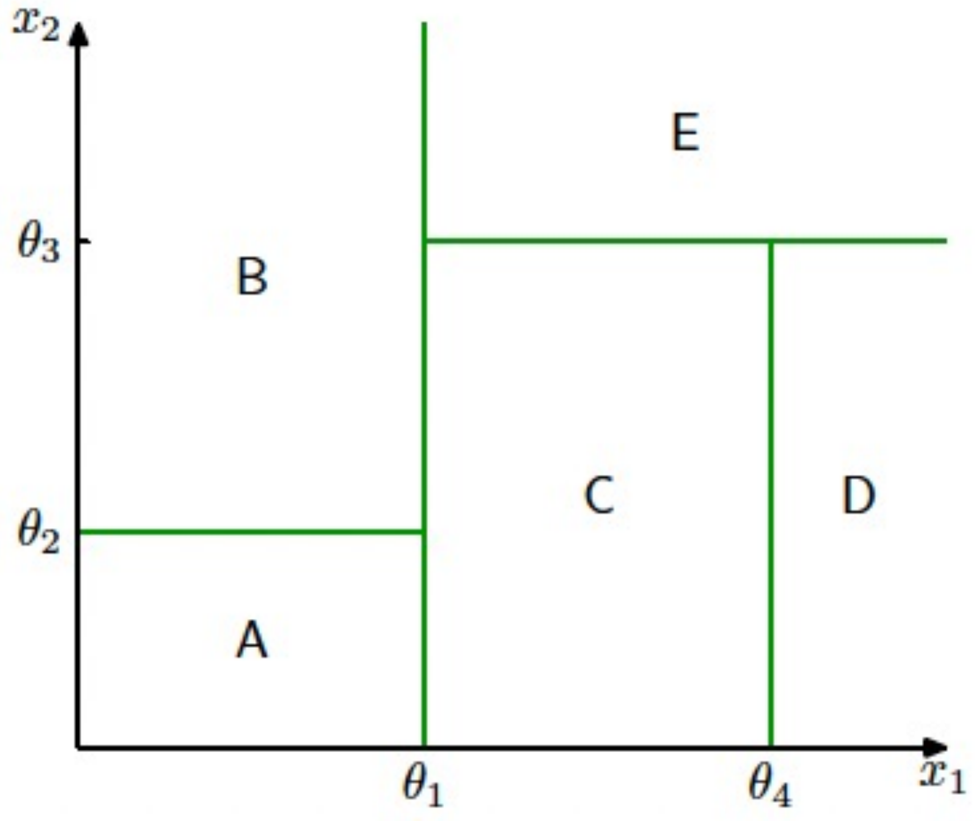
Thus, AdaBoost with decision stumps is a **linear classifier!**

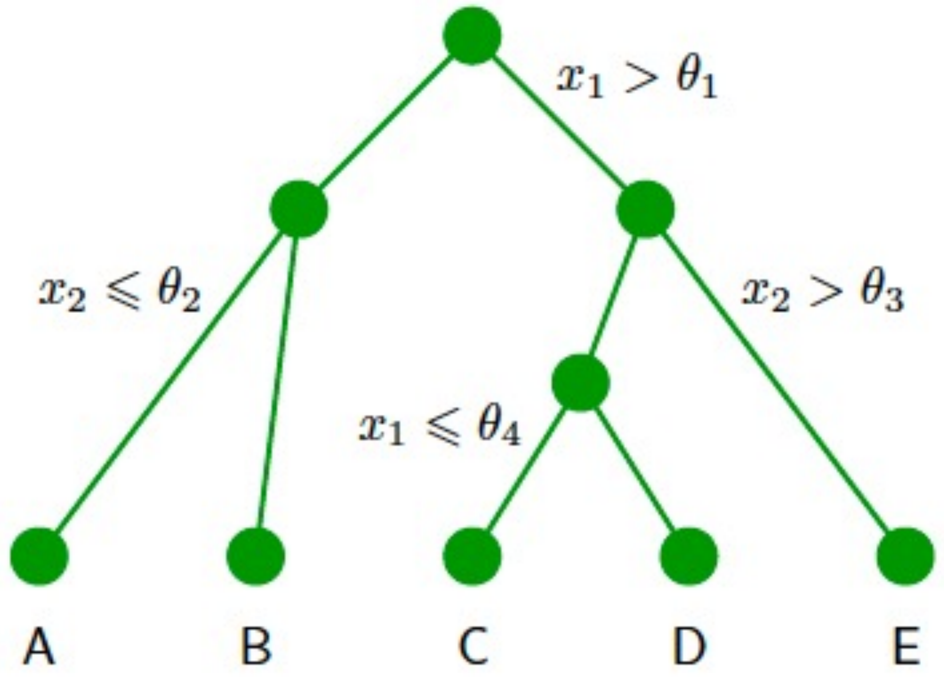
Consider boosting a linear classifier,

$$f(\boldsymbol{x}) = \operatorname{sgn}\left[\sum_k \alpha_k \operatorname{sgn}\left(\boldsymbol{w}^{(\mathrm{k})} \cdot \boldsymbol{x} + b^{(\mathrm{k})}\right)\right]$$

What type of model does this look like?

Tree-Based Models as Ensembles



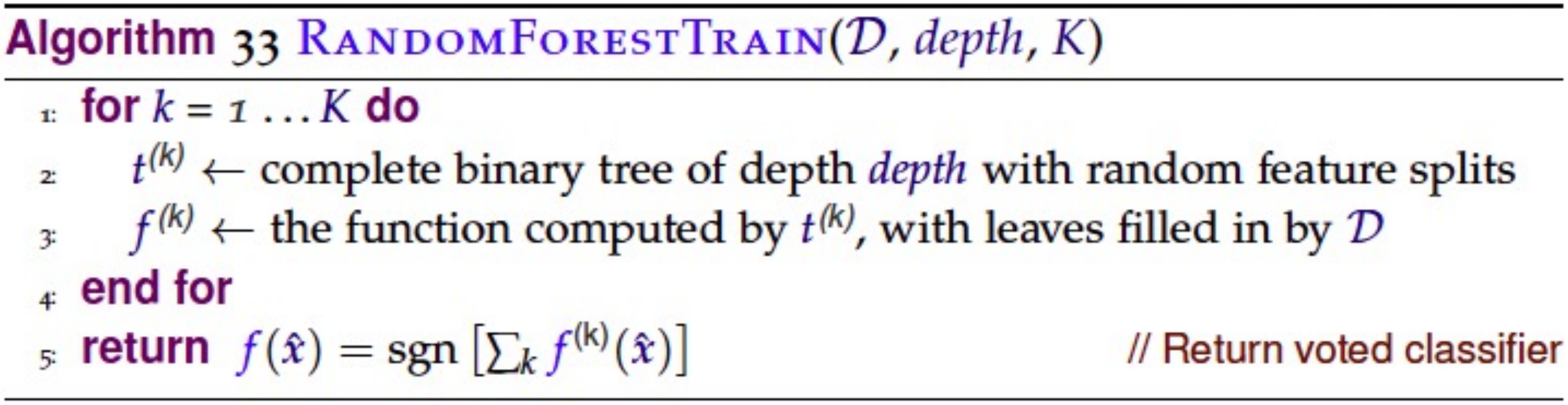


Assigns simple (constant prediction) model in regions

Random Forest Classifiers

- Training decision trees is expensive
    - Expensive part is choosing tree structure
    - Filling in leaves is cheap
- **Idea** Use random tree structures and just fill in leaves
    - This is a *random decision tree*
    - A collection of random trees is a ***random forest***
- **Approach**
    - Generate K (full) binary trees with random features
    - Use training data to assign leaves (classification decisions)

Random Forest



Algorithm 33 RandomForestTrain($\mathcal{D}$, *depth*, K)

```
1: for $k = 1 \dots K$ do
2:     $t^{(k)} \leftarrow$ complete binary tree of depth $depth$ with random feature splits
3:     $f^{(k)} \leftarrow$ the function computed by $t^{(k)}$, with leaves filled in by $\mathcal{D}$
4: end for
5: return $f(\hat{x}) = \text{sgn}\left[\sum_k f^{(k)}(\hat{x})\right]$        // Return voted classifier
```

- K trees can be generated in parallel
- Features are selected randomly **with replacement**
- May have duplicate features, even in single path
- Data is *only* needed to assign leaves

Random Forest : Why does it work?

- Some trees will query on *useless* features
  - These trees will make essentially random predictions
- But some trees will query on *good* features
  - These trees will make good predictions
  - Because leaves are estimated based on training data
- If you have enough trees…
  - Random ones will wash out as noise
  - Only good trees affect final classification