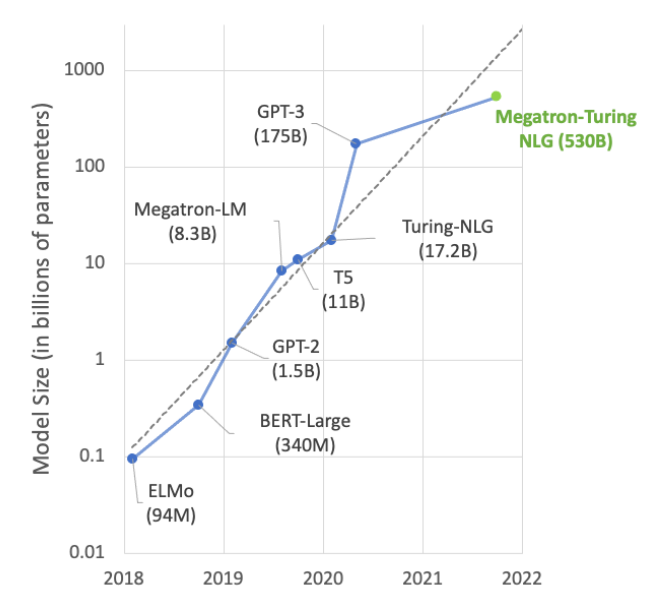
Mathematical topics in Machine Learning

(Lecture 5 – Ensemble learning)

Professor Gavin Brown

Our question

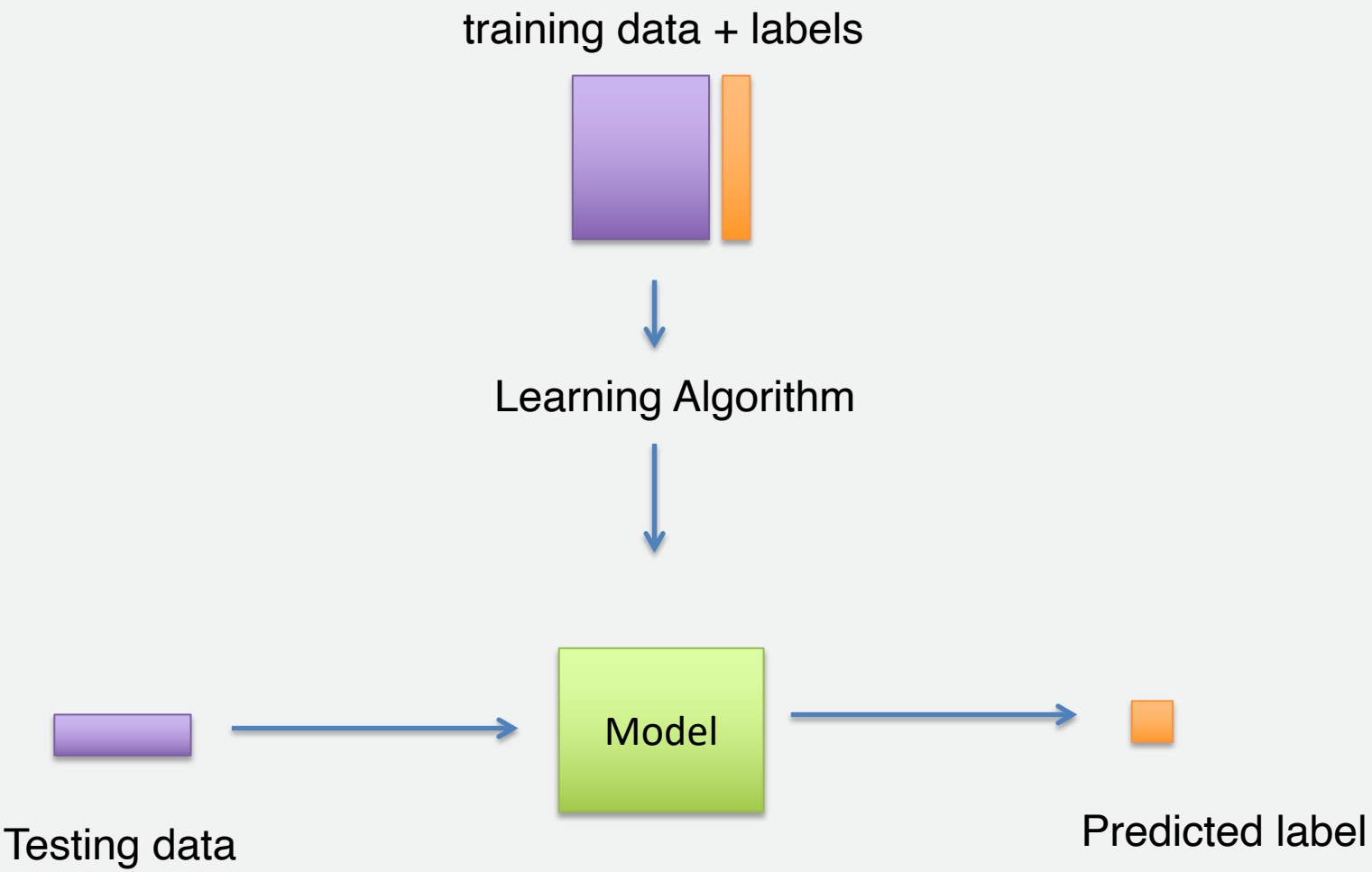
"Are bigger models always better models?"

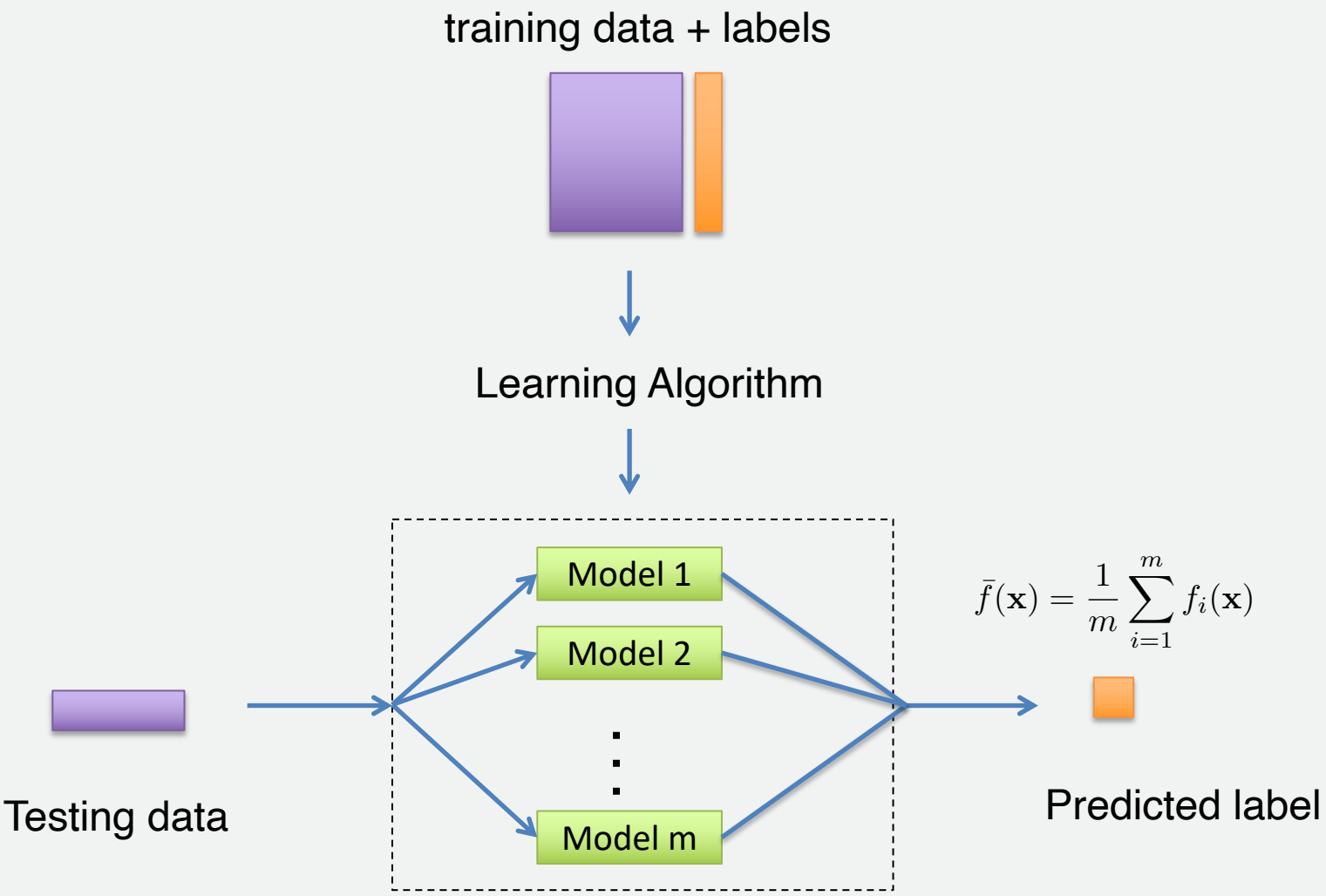


TODAY:

We make models larger by forming "**committees**".

We combine their "diverse" predictions, in the hope of cancelling out individual errors.


training data + labels
Learning Algorithm
Model
Testing data
Predicted label


training data + labels
Learning Algorithm
Model 1
Model 2
Model m
$\bar{f}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x})$
Testing data
Predicted label

The "**Ambiguity Decomposition**"

The ensemble will always have error less than or equal to the average.

Define $\bar{f}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x})$

We have a guarantee that… $\left(\bar{f}(\mathbf{x}) - y\right)^2 \leq \frac{1}{m}\sum_{i=1}^{m}\left(f_i(\mathbf{x}) - y\right)^2$

In fact…

$$\left(\bar{f}(\mathbf{x}) - y\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(f_i(\mathbf{x}) - y\right)^2 - \underbrace{\frac{1}{m}\sum_{i=1}^{m}\left(f_i(\mathbf{x}) - \bar{f}(\mathbf{x})\right)^2}_{\text{non-negative}}$$

Krogh & Vedelsby, "Neural Networks, Cross Validation and the Bias-Variance Dilemma" Neural Information Processing Systems 1995

The "**Ambiguity Decomposition**"

Squared loss

$$\left(\bar{f}(\mathbf{x}) - y\right)^2 = \frac{1}{m}\sum_{i=1}^{m}\left(f_i(\mathbf{x}) - y\right)^2 - \frac{1}{m}\sum_{i=1}^{m}\left(f_i(\mathbf{x}) - \bar{f}(\mathbf{x})\right)^2$$

Arithmetic mean

$$\bar{f}(\mathbf{x}) = \frac{1}{m}\sum_{i=1}^{m} f_i(\mathbf{x})$$

Cross-entropy loss

$$\ell(\mathbf{y}, \bar{f}(\mathbf{x})) = \frac{1}{m}\sum_{i=1}^{m}\ell(\mathbf{y}, f_i(\mathbf{x})) - \underbrace{\frac{1}{m}\sum_{i=1}^{m} K(\bar{f}(\mathbf{x}), f_i(\mathbf{x}))}$$

KL-divergence
(also non-negative)

Geometric mean

$$\bar{f}(\mathbf{x}) = Z^{-1}\prod_i f_i(\mathbf{x})^{\frac{1}{m}}$$

See notes for details.

Or… Majority voting (more complex!)

Marquis de Condorcet

In 1786 Nicolas de Condorcet (political theorist) asked how do parliaments behave when voting?

…assuming M independent voters…

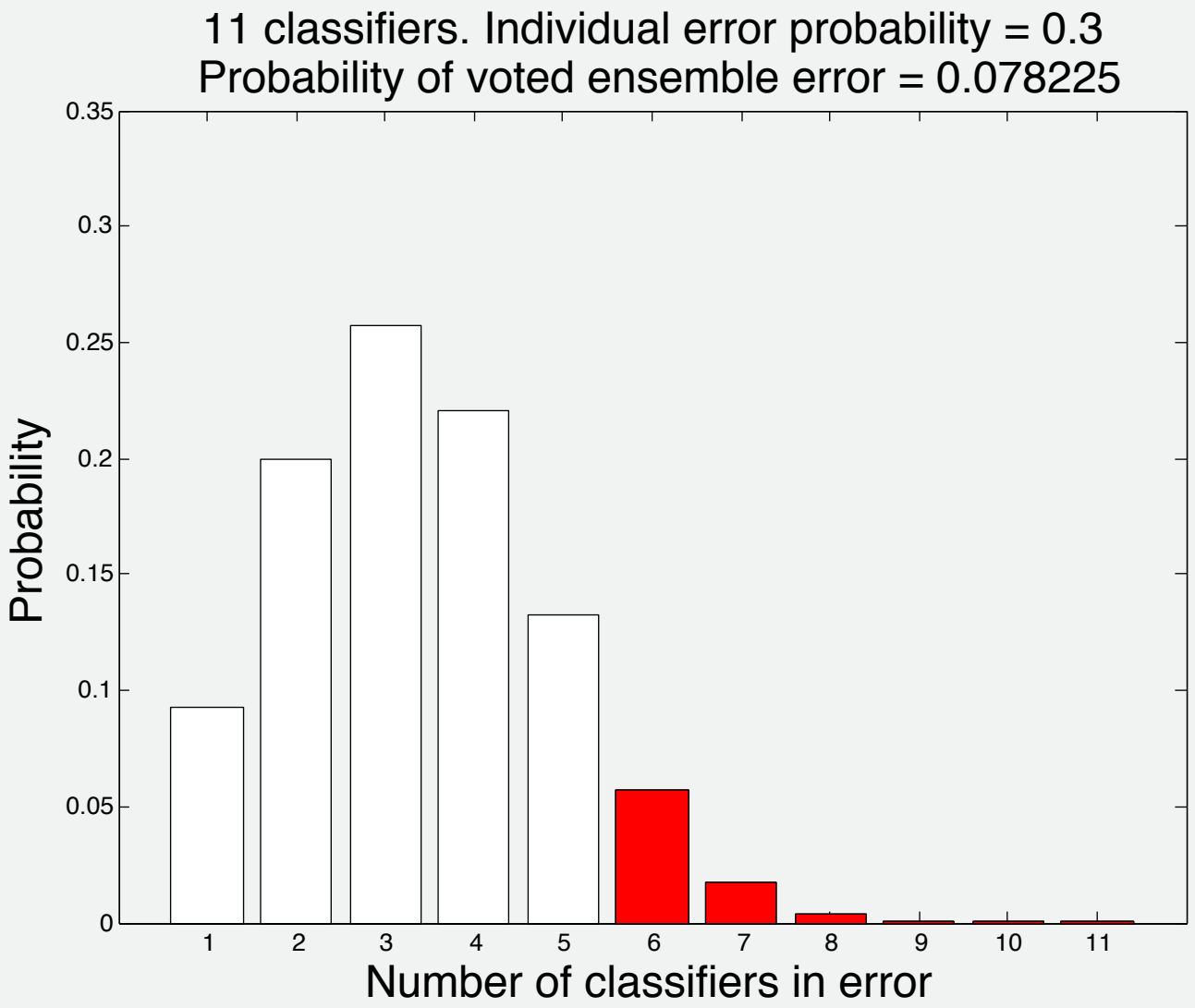
If a single voter has a probability ϵ of making an error, then

$$p(\text{exactly } k \text{ errors}) = \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)}$$

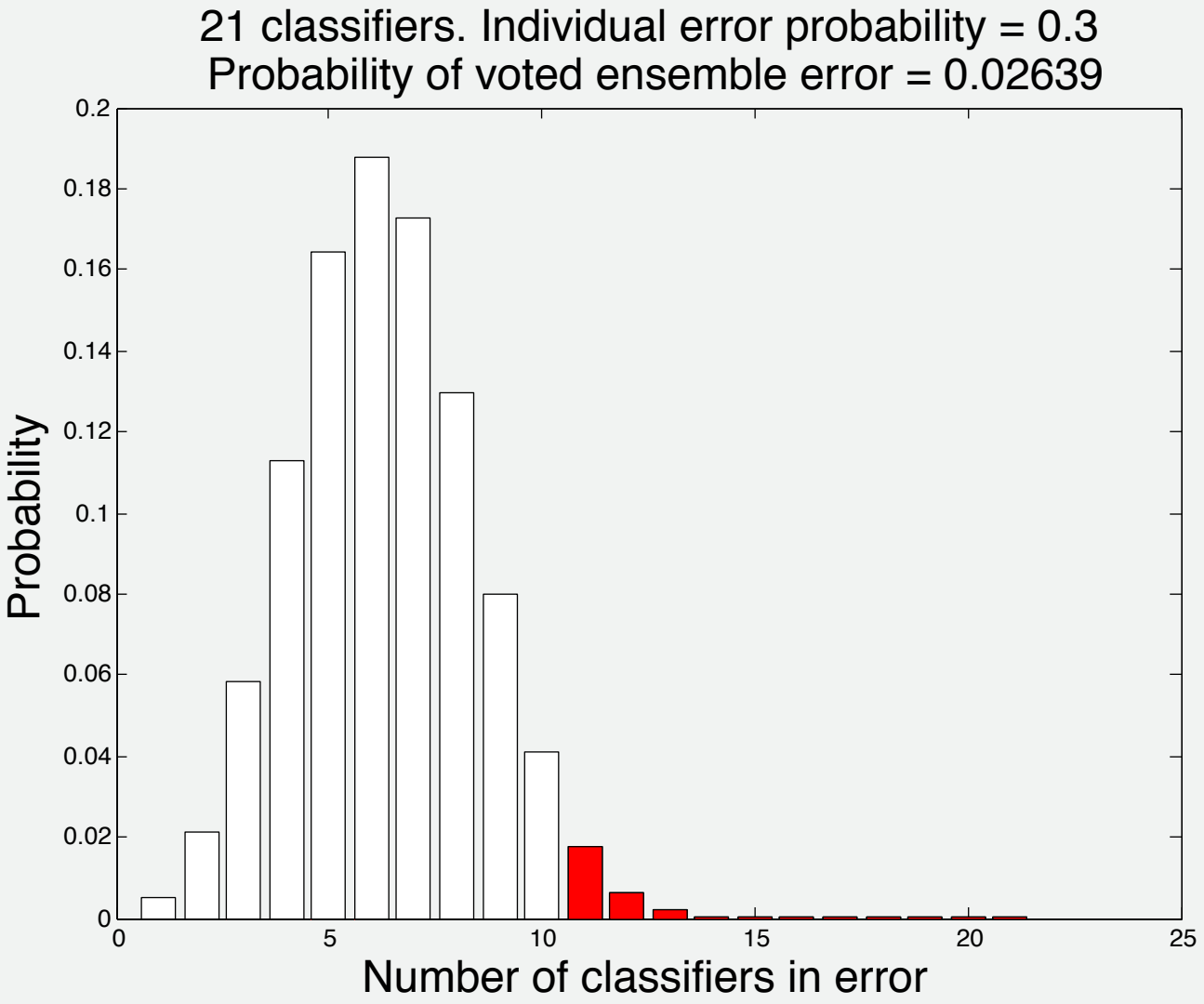
Voting error means more than half of the committee being in error…

$$p(majority\ vote\ error) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)}$$

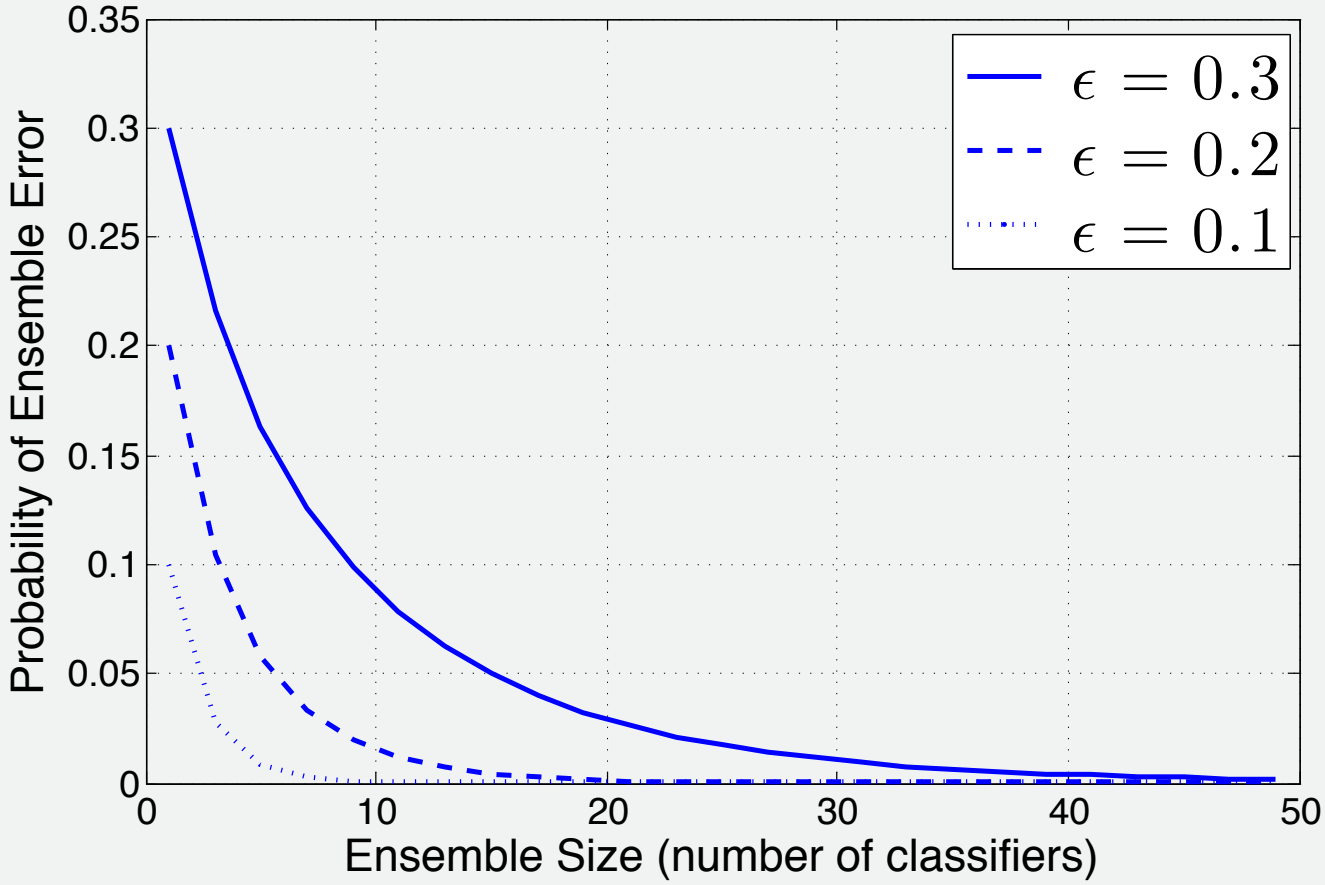
$$p(majority\ vote\ error) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)}$$



$$p(majority\ vote\ error) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)}$$

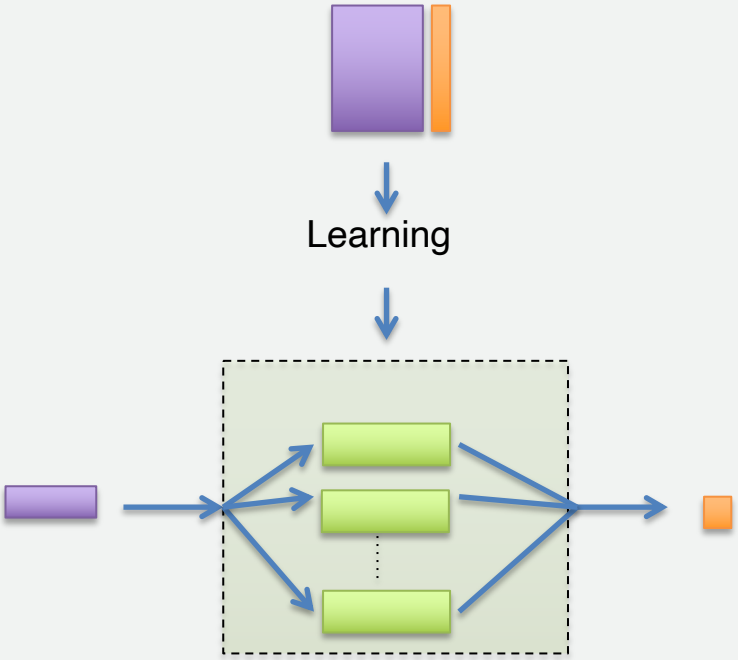


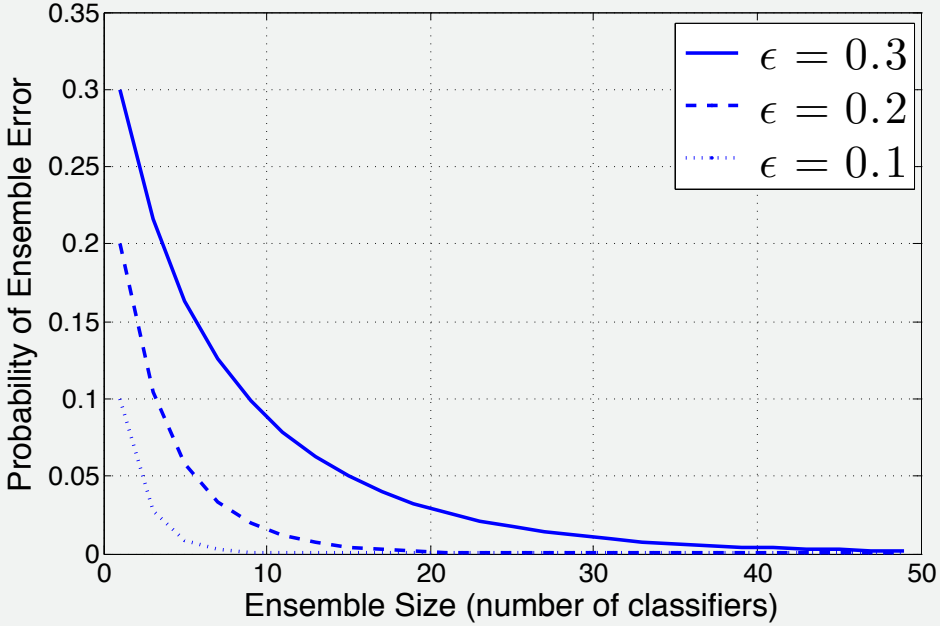
$$p(majority\ vote\ error) = \sum_{k \geq \lceil \frac{M+1}{2} \rceil} \binom{M}{k} \epsilon^k (1-\epsilon)^{(M-k)}$$



Virtually ZERO error by M = 50 !!

Independent models?



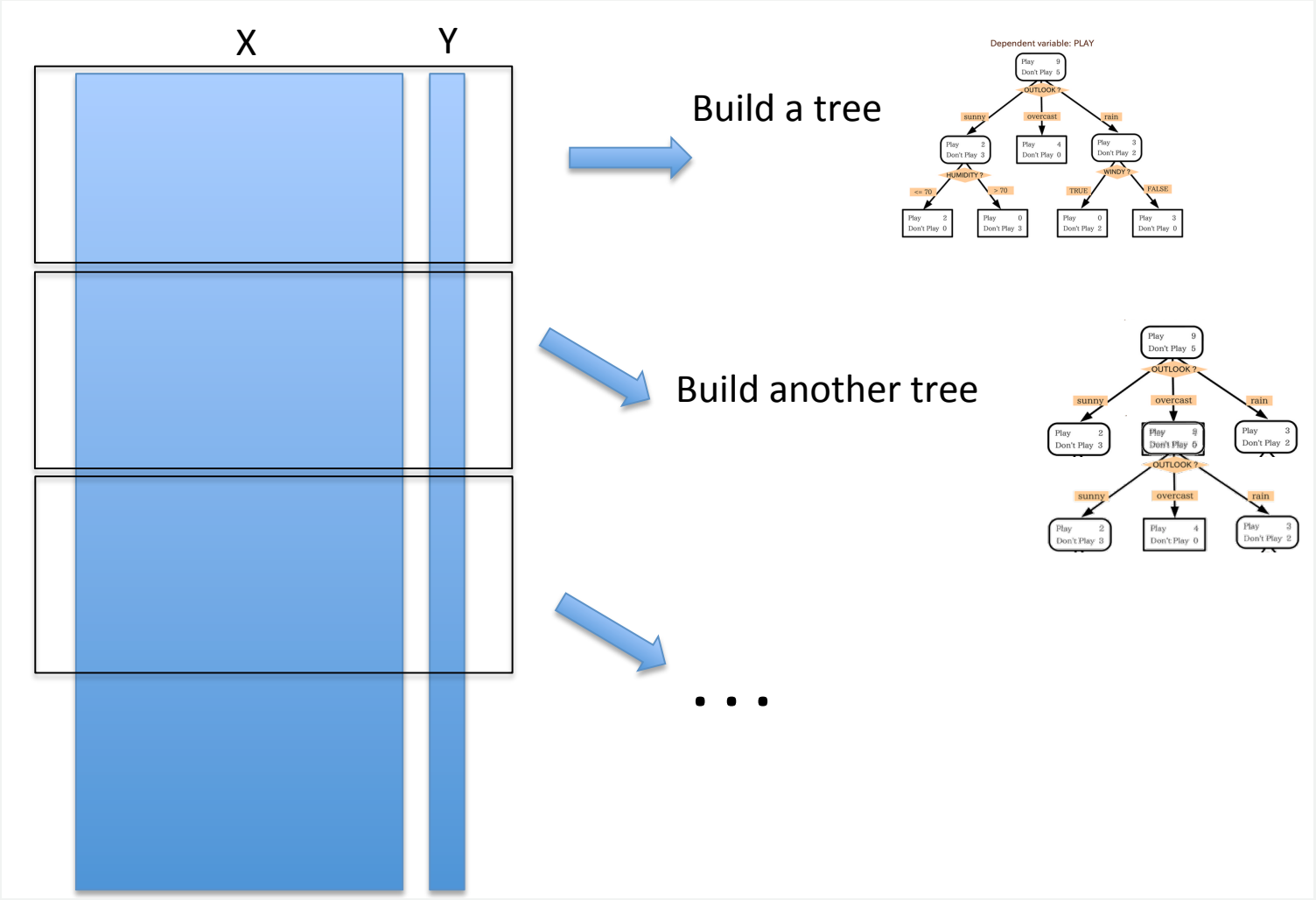


Models are produced by same process.

So are most likely **not** independent.

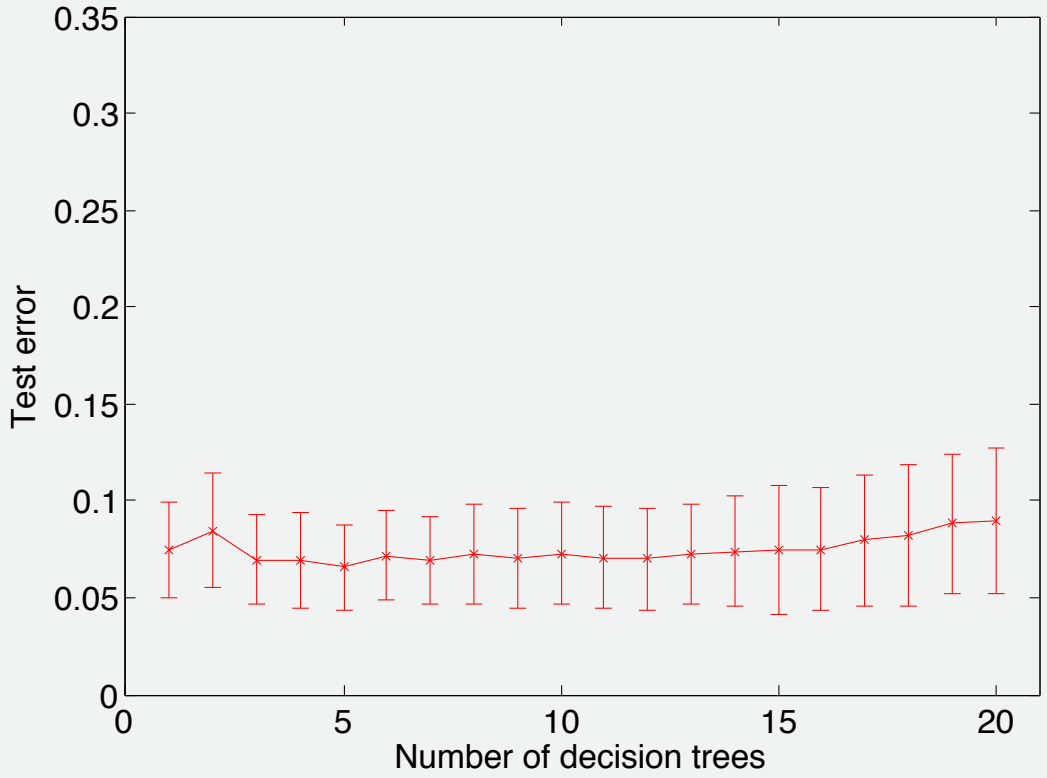
How could we **encourage** them to be independent?

Divide the training set between classifiers?



Divide the training set between classifiers?

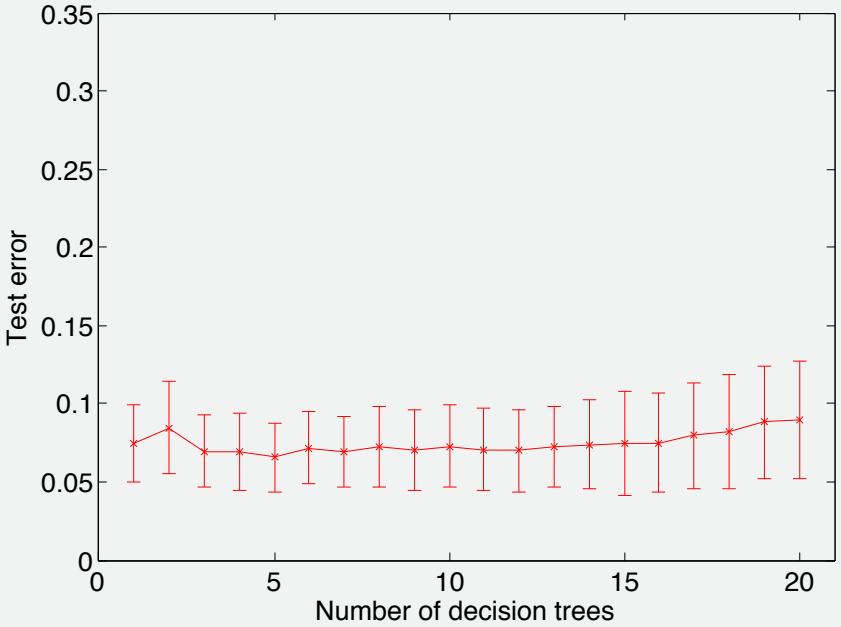
Splice data (categorical) – 3 classes, 60 features, 3175 examples – 20 x 2-fold CV



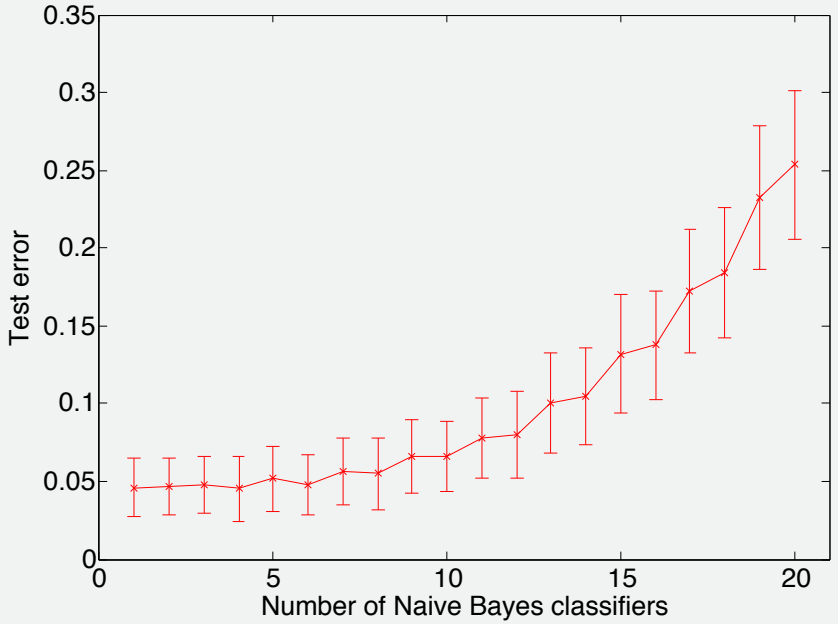
Not working. Why?

Divide the training set between classifiers?

Splice data (categorical) – 3 classes, 60 features, 3175 examples – 20 x 2-fold CV



Trees

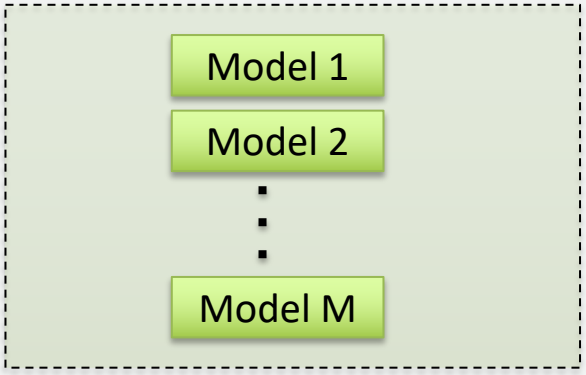


Naïve Bayes

Potential gain from independent errors outweighed by losses made by more error-prone classifiers!

It's a trade-off …

- Models need to be reasonably good by themselves
- Don't want all models to be identical.
- Don't want them too different – sacrificing individual performance

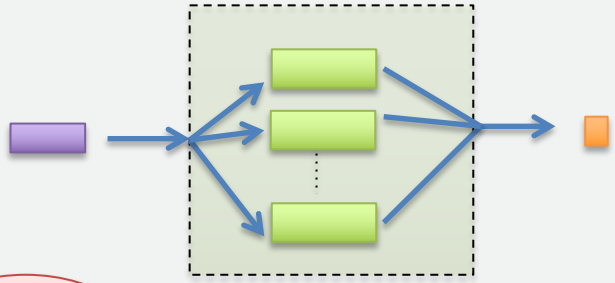


Equivalent to underfitting / overfitting – need the right balance.

The full theory of this is for next week…

For now, let's look at two practical methods : **parallel**, and **sequential**.

Error = Bias + Variance



Strength

(Bias)

SVM (linear)

Naïve Bayes

Bad place

Decision Tree

K-NN

SVM (RBF)

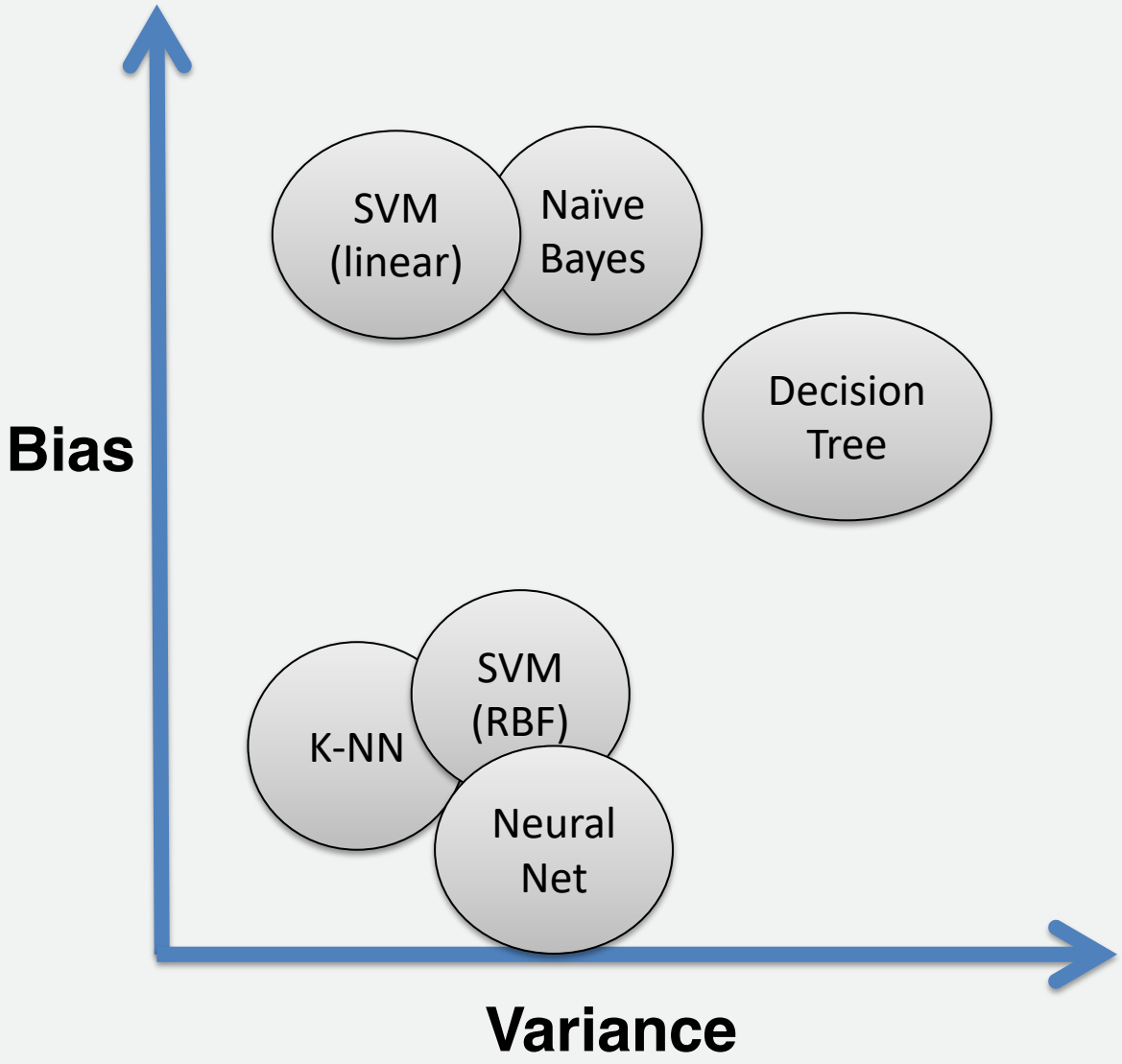
Neural Net

Good place

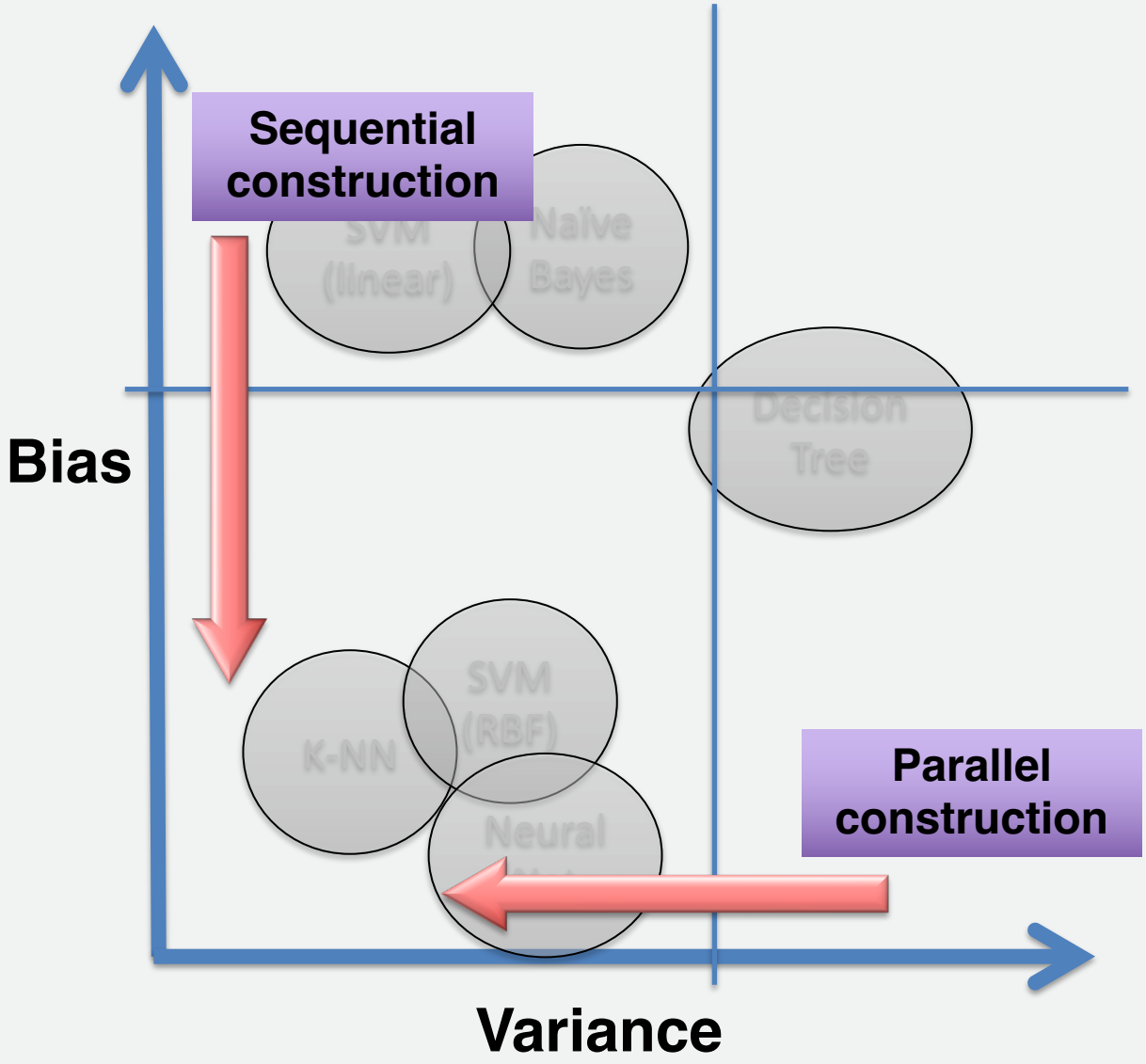
Sensitivity to training data

(Variance)

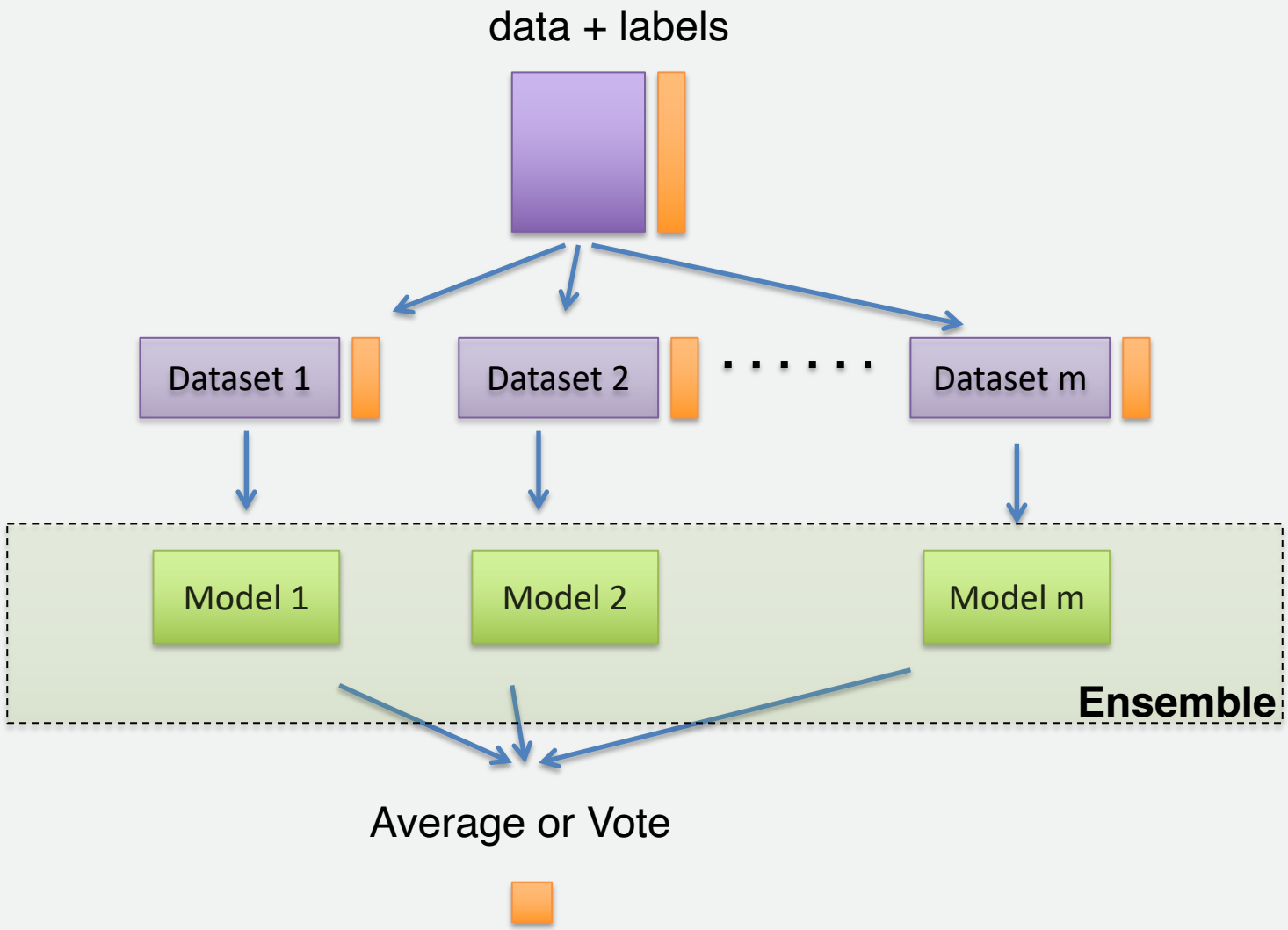
Two approaches to ensemble construction…



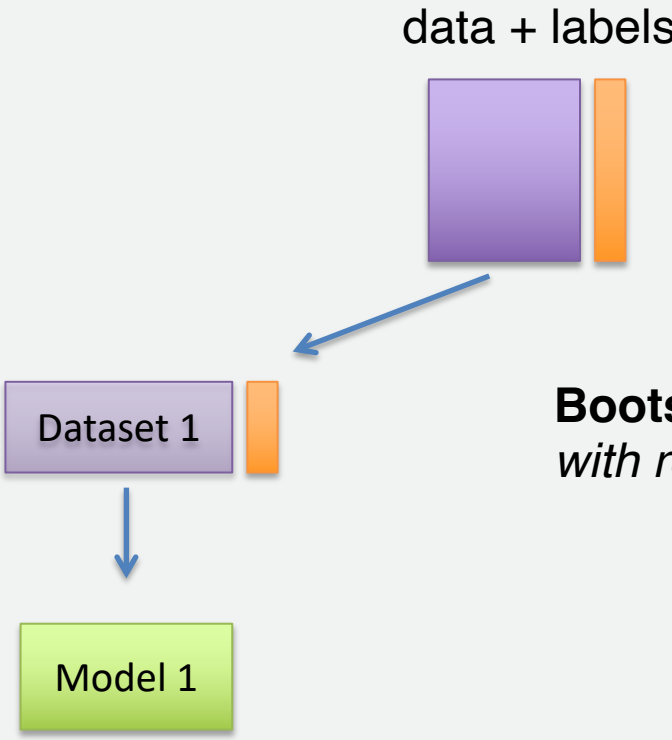
Two approaches to ensemble construction…



Parallel construction…

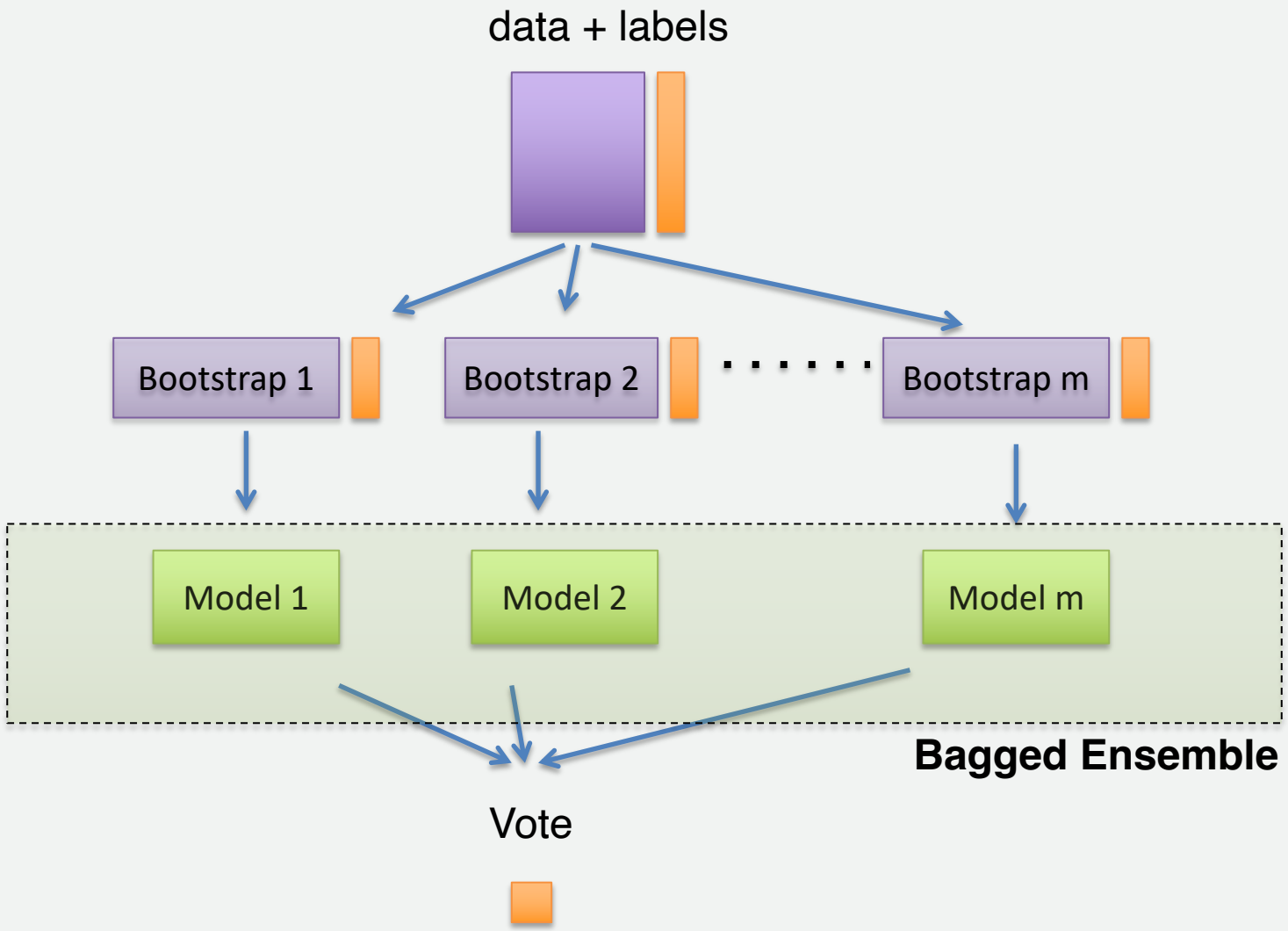


“Bagging” : **B**ootstrap **AGG**regat**ING**



Bootstrap: Sample N examples,
with replacement, from the original N.

“Bagging” : **B**ootstrap **AGG**regat**ING**



Generating a new dataset by “Bootstrapping”

- sample N items with replacement from the original N

Original data

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
2	0.11	0.82	0.07	0
3	0.87	0.3	0.21	1
4	0.34	0.49	0.18	1
5	0.95	0.64	0.63	0
6	0.03	0.59	0.15	1

Bootstrap 1

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
1	0.18	0.45	0.8	0
3	0.87	0.3	0.21	1
4	0.34	0.49	0.18	1
5	0.95	0.64	0.63	0
5	0.95	0.64	0.63	0

Bootstrap 2

id	x1	x2	x3	y
1	0.18	0.45	0.8	0
2	0.11	0.82	0.07	0
3	0.87	0.3	0.21	1
6	0.03	0.59	0.15	1
6	0.03	0.59	0.15	1
6	0.03	0.59	0.15	1

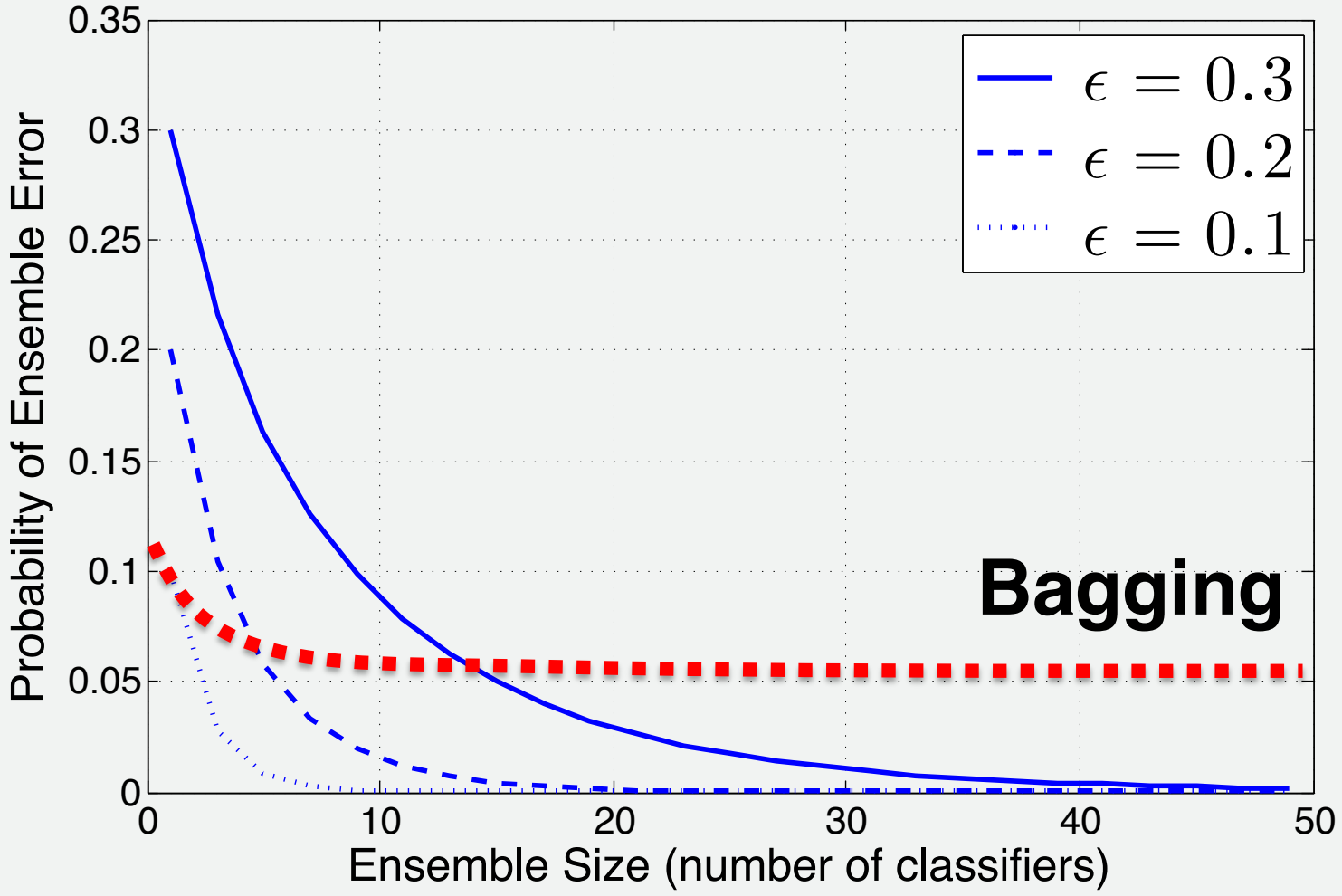
The “Bagging” algorithm (Breiman, 1996)

Bagging (requires training data+labels D, and choice of number of models M)

---

for $j = 1$ to M **do**
    Take a *bootstrap* sample D' from D
    Build a model using D'.
    Add the model to the set.
end for
return set of models
For a test point $\mathbf{x}$, get a response from each model, and combine predictions.

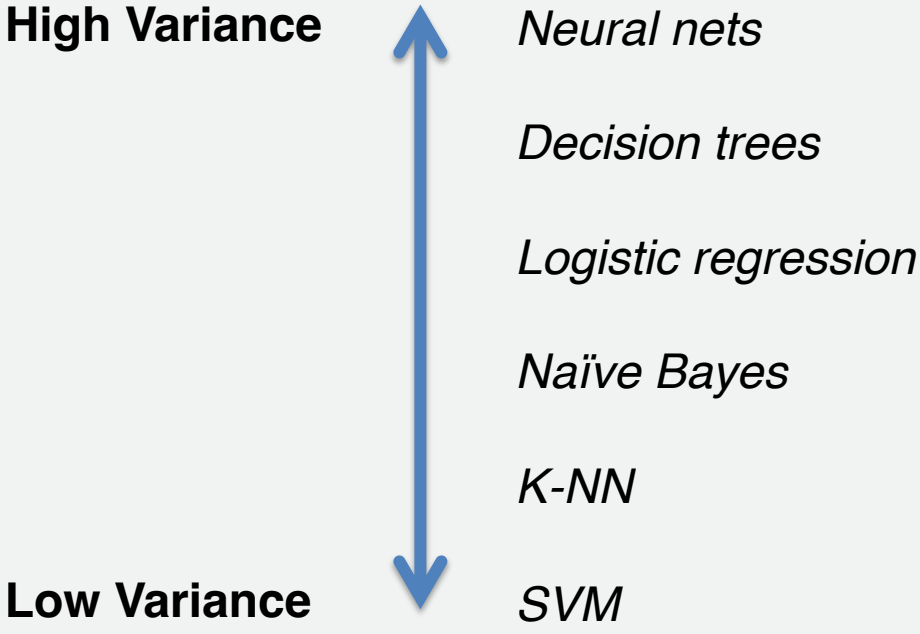
Theory vs reality…



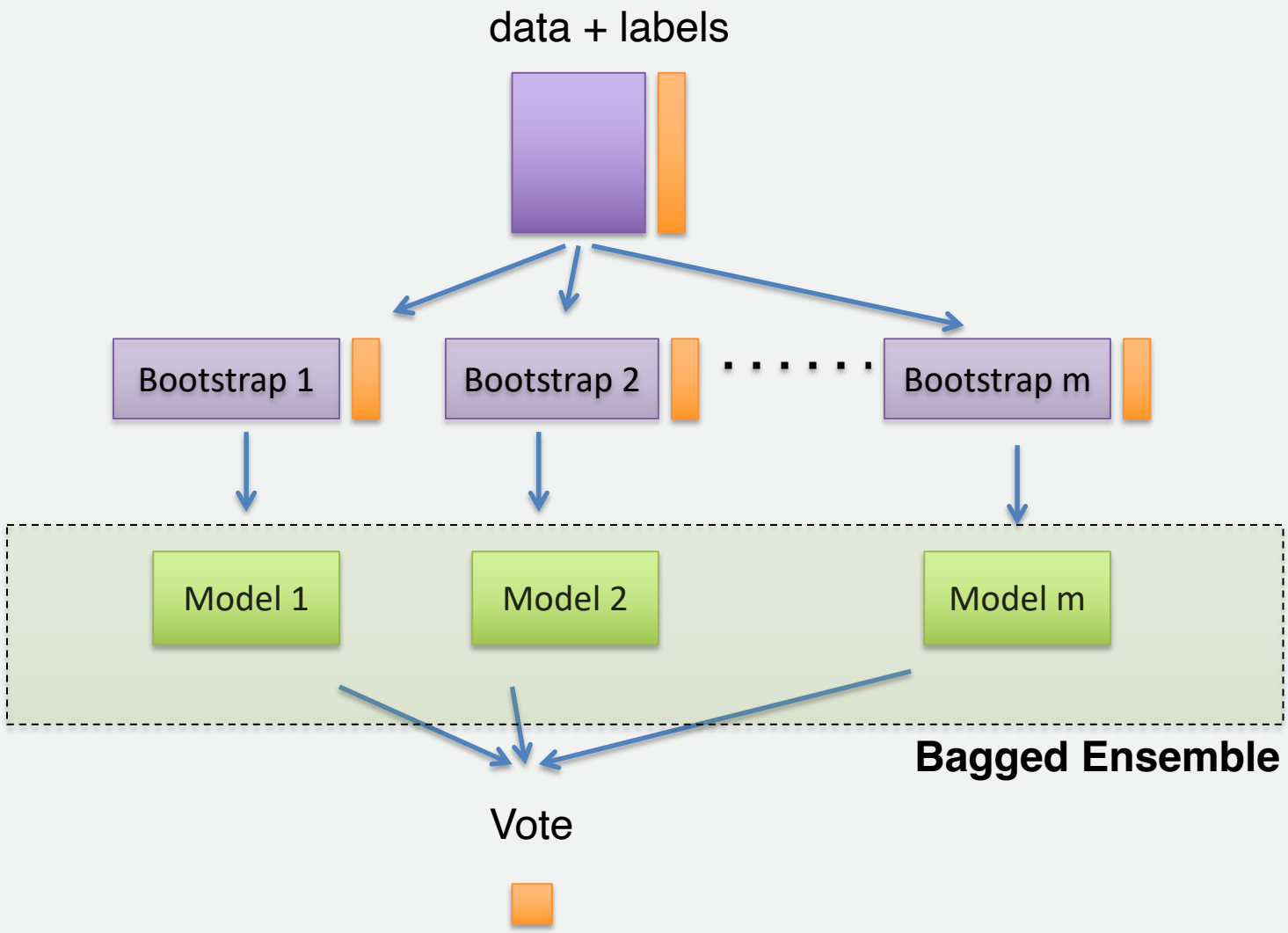
Some models are almost completely unaffected by bootstrapping.

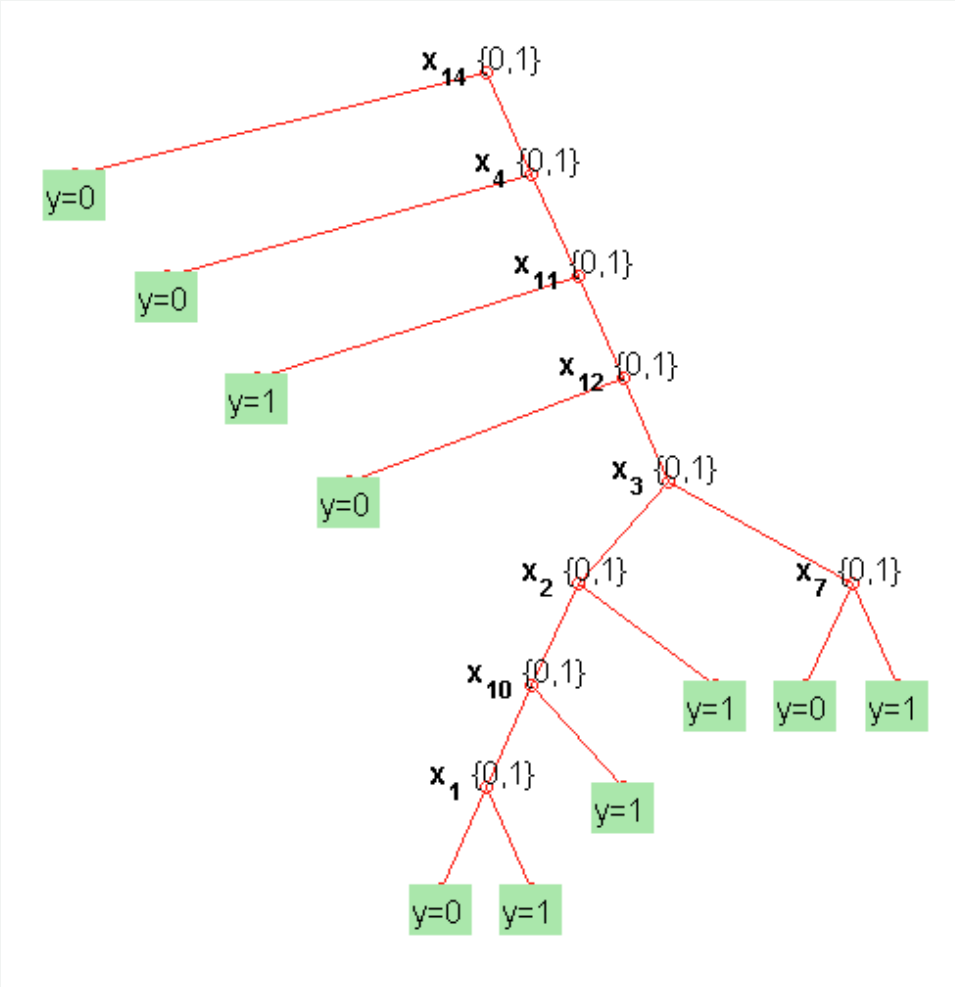
They become very similar to each other on test data.

These are **low variance** models. Not so suitable for parallel methods.



Maybe just bagging isn't enough?





Normal Tree building:

Pick the best feature to split on.

Instead, randomize this a bit.

At each split point…

Choose a random subset.

Pick the best from this subset.

Random Forests (Breiman, 2000)

$$K = \lceil \sqrt{d} \rceil$$

d = total # features

Random Forests (input training data+labels D, number of trees M)

---

for $j = 1$ to M **do**
    Take a bootstrap D' from D
    Build a tree using D', but, at every split point:
        - Choose a random fraction K of the remaining features.
        - Pick the best feature from that subset.
    Add the tree to the set.
end for
return set of trees
For a test point $\boldsymbol{x}$, get a response from each tree, and take a majority vote.

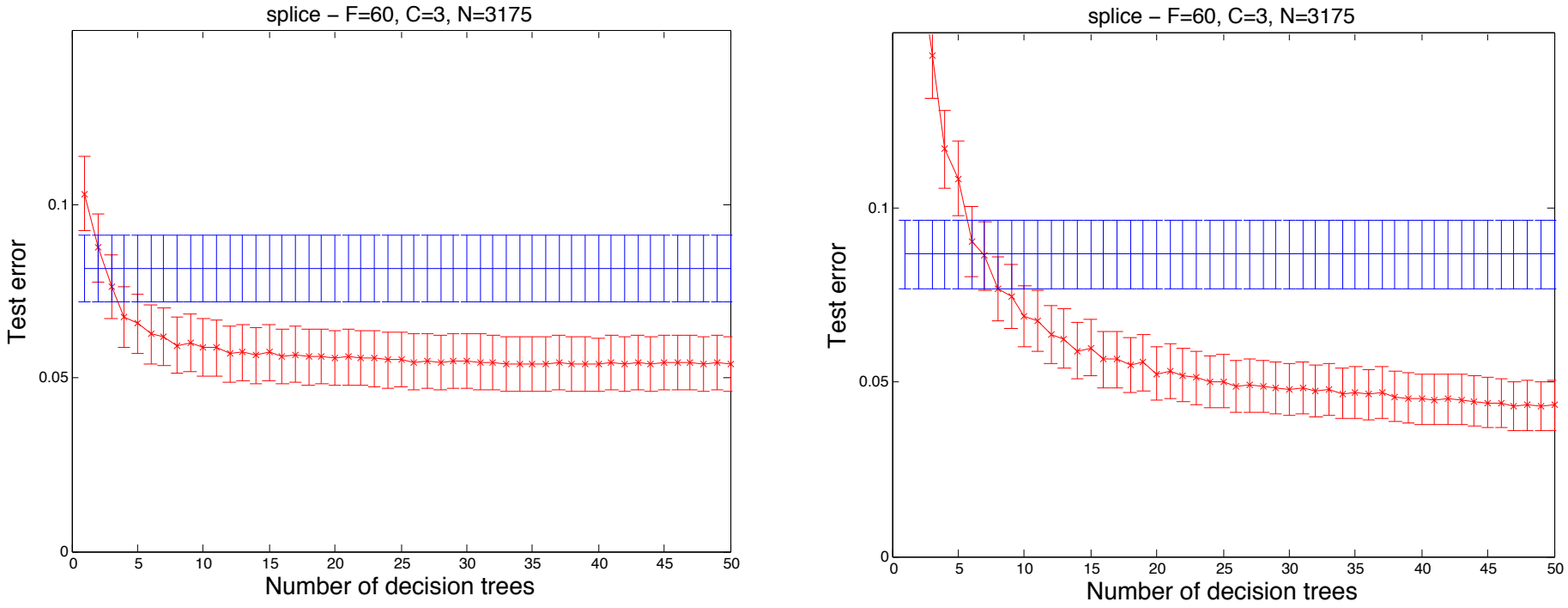
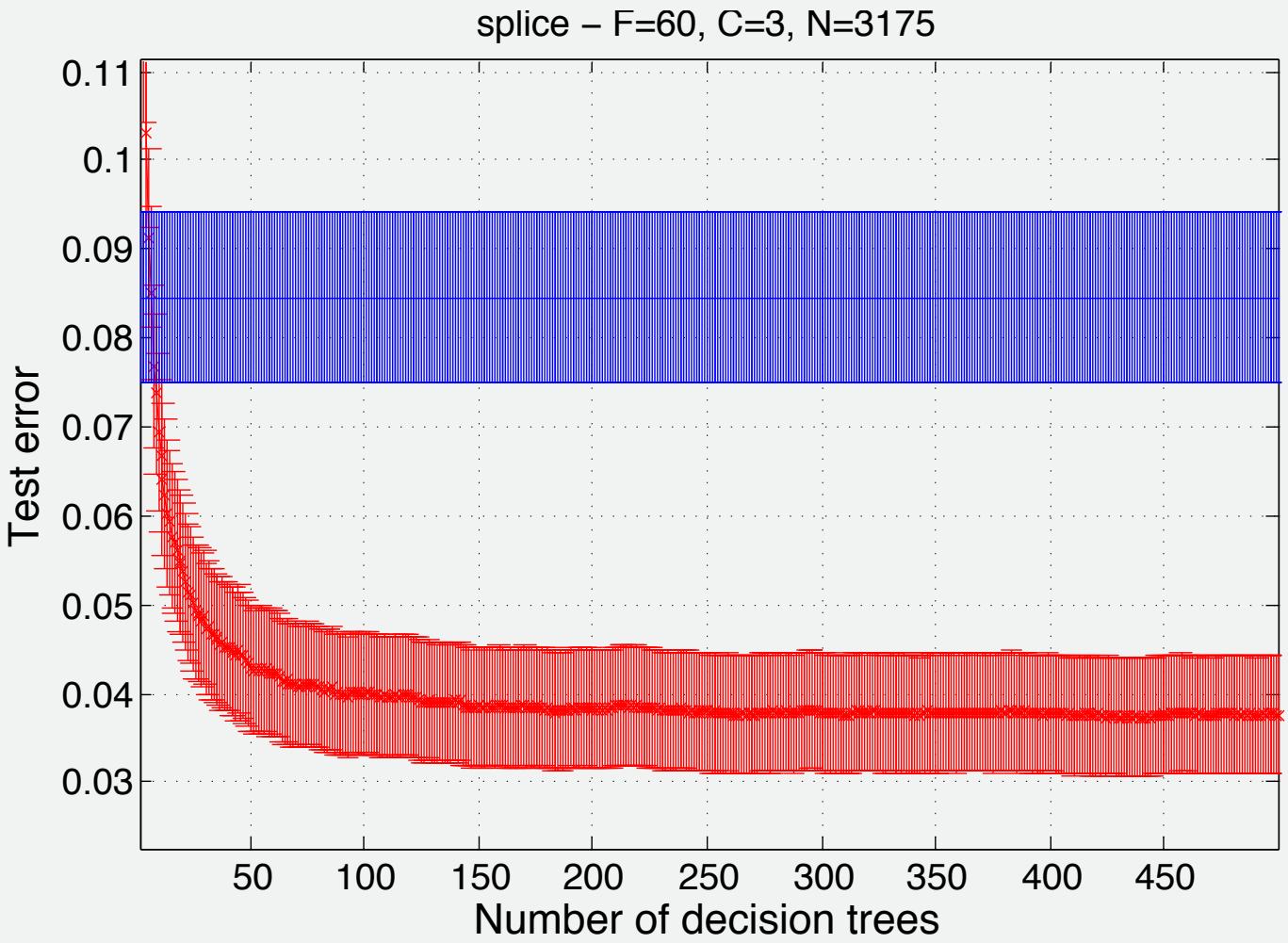
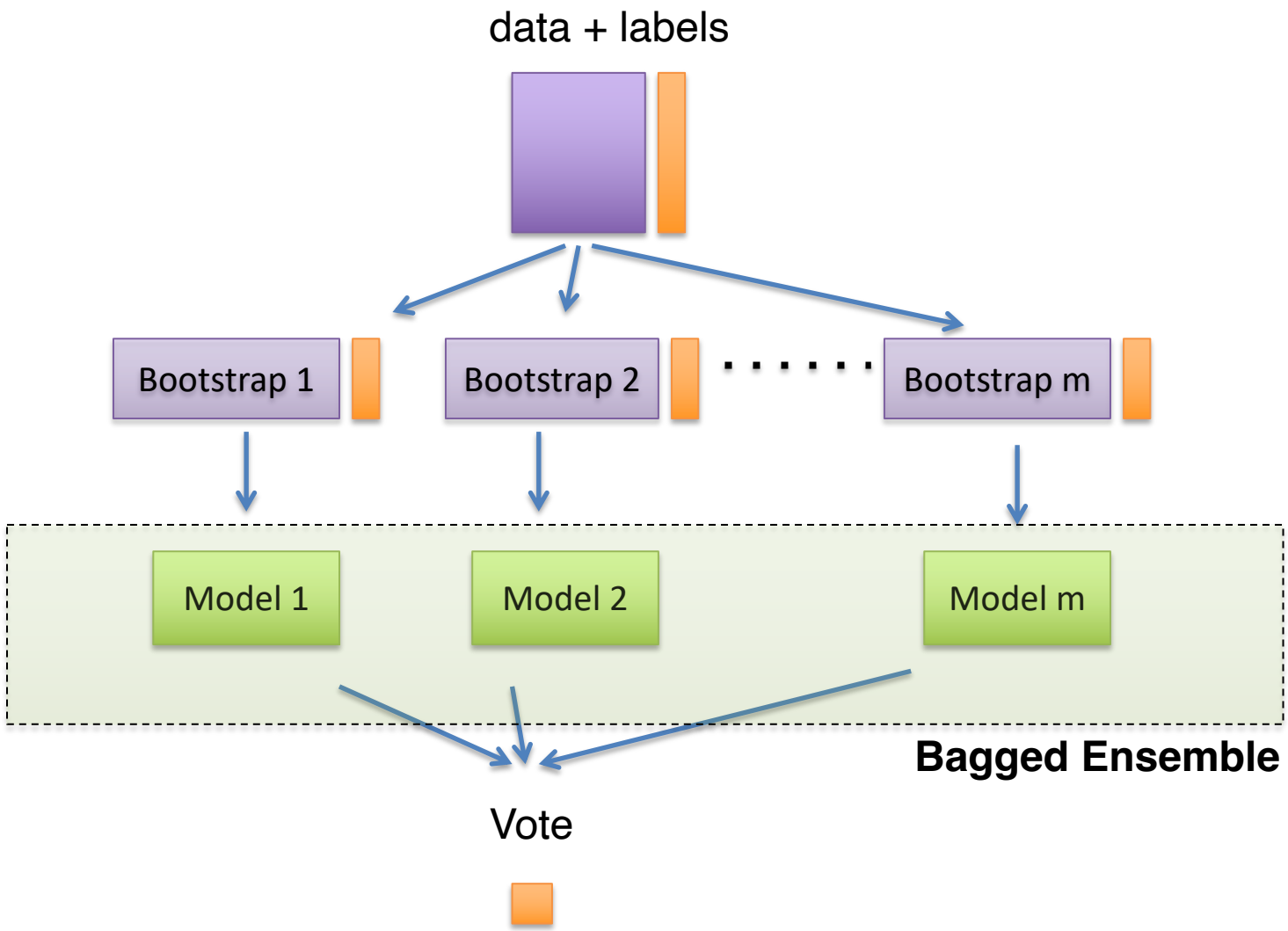


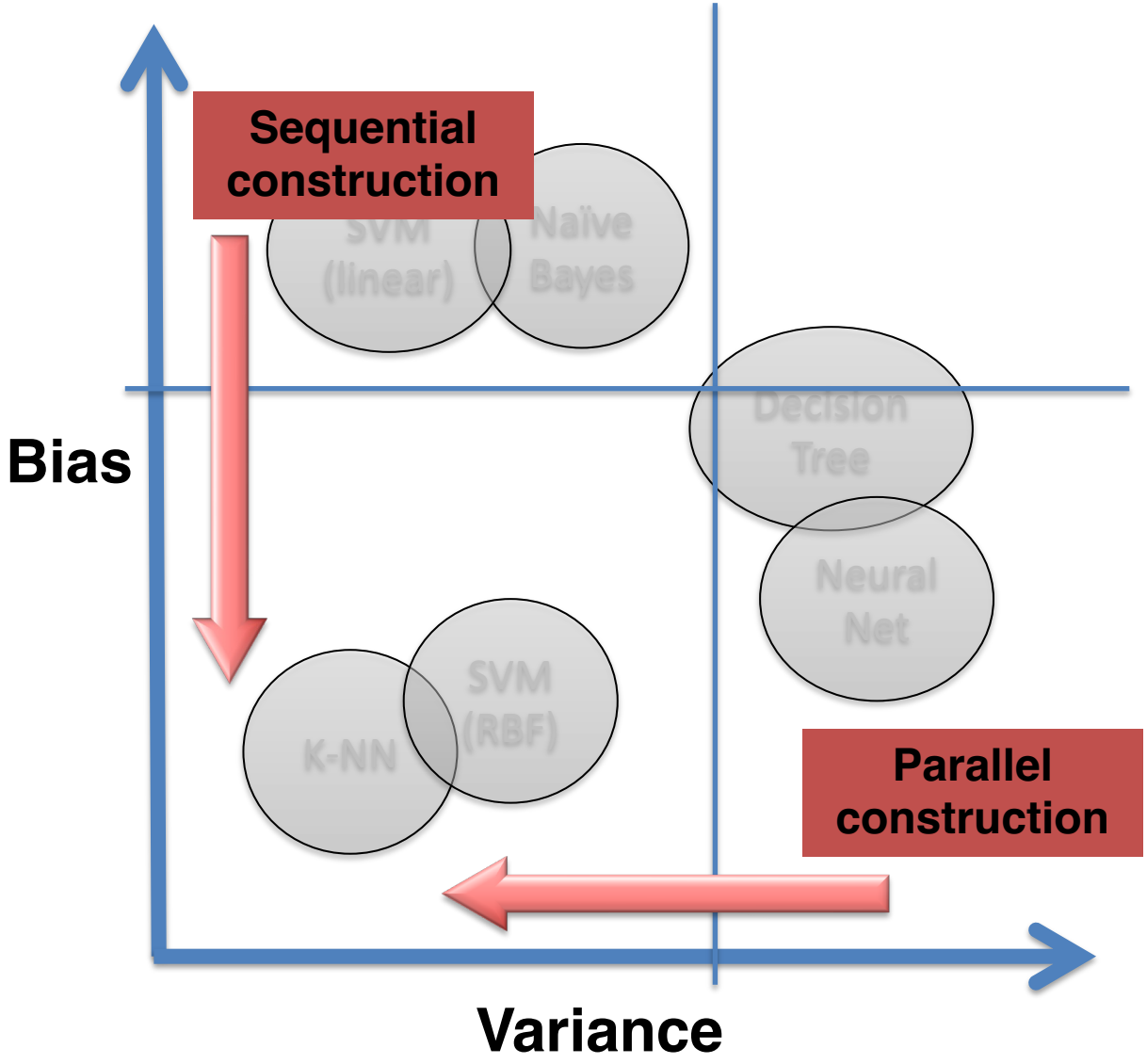
Figure 13: Bagging (LEFT) vs Random Forests (RIGHT) on the Splice dataset.

Random Forest up to 500 trees…

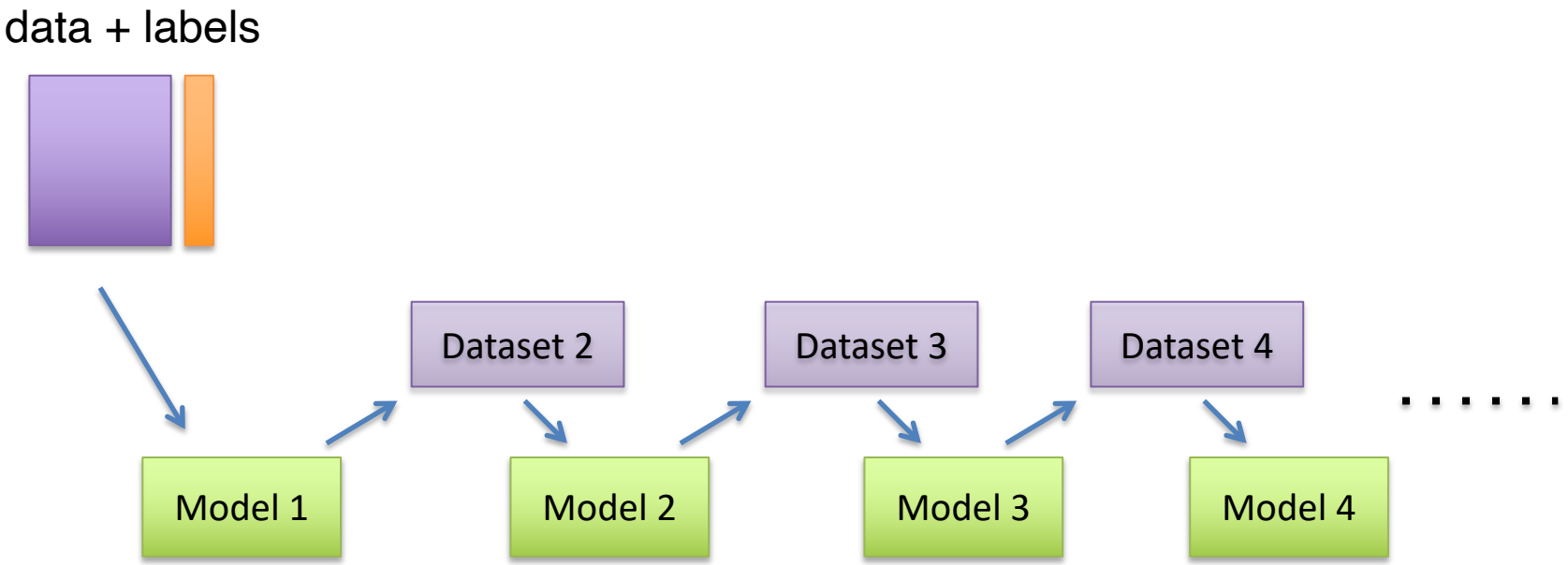


Maybe just Bagging / Random Forests isn't enough?



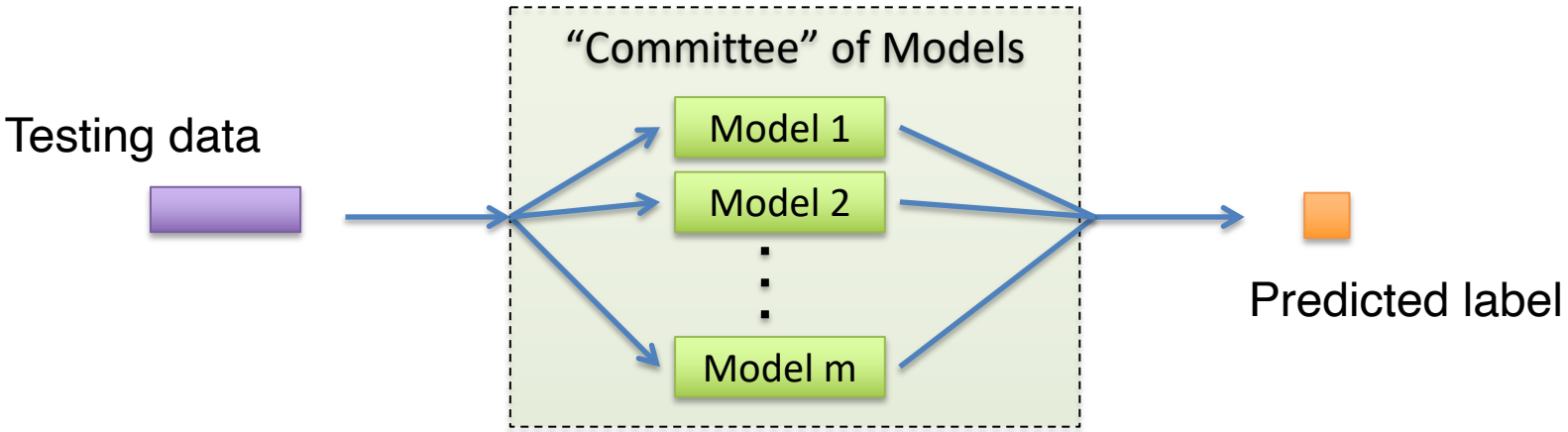

Sequential construction
SVM (linear)
Naïve Bayes
Decision Tree
Neural Net
K-NN
SVM (RBF)
Parallel construction
Bias
Variance

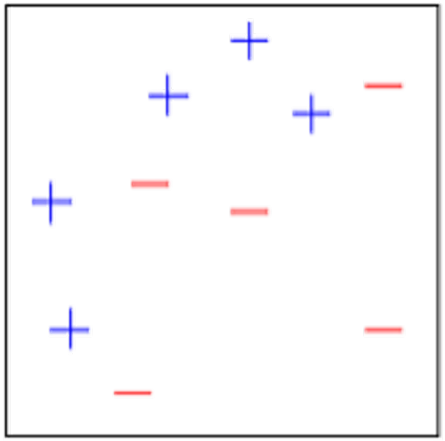
Sequential construction….

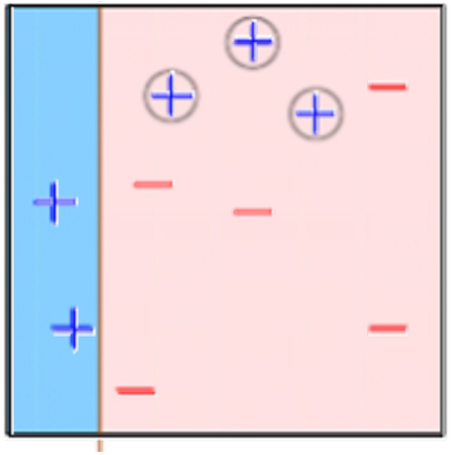


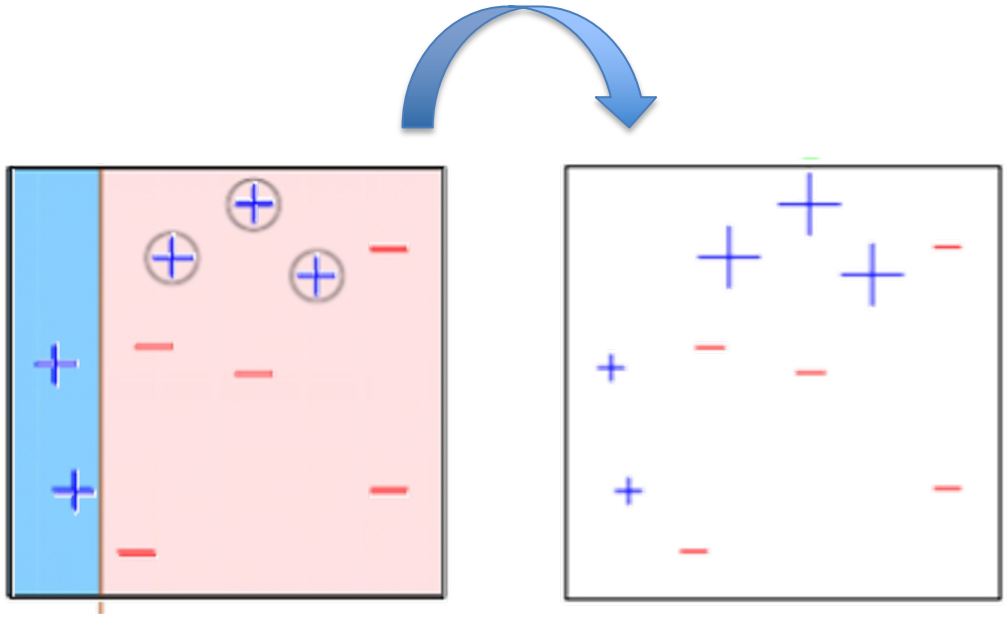
Each model corrects the mistakes of its predecessors.

At testing phase, parallel/sequential are the same

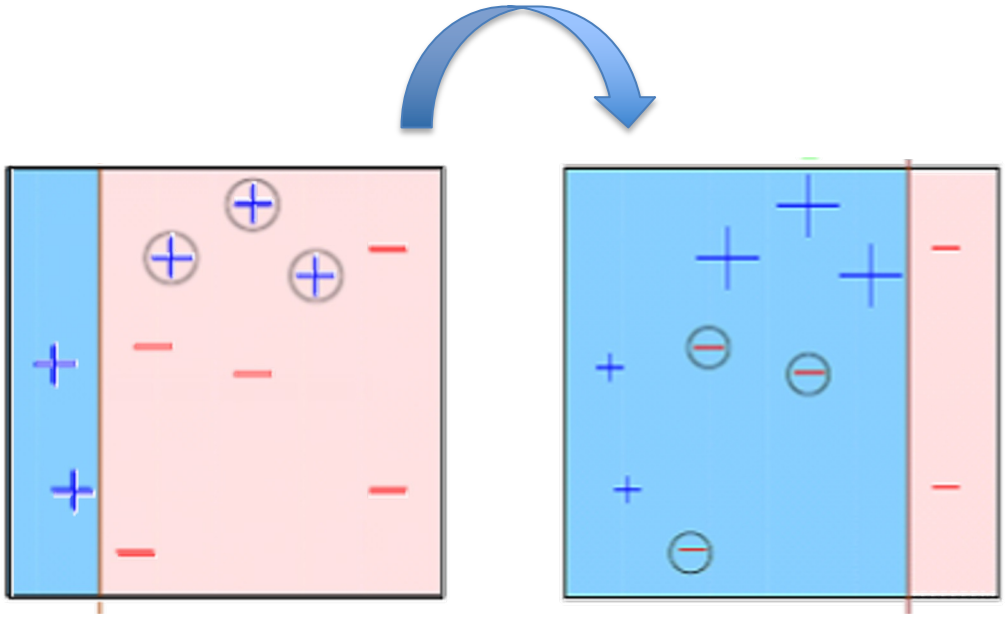




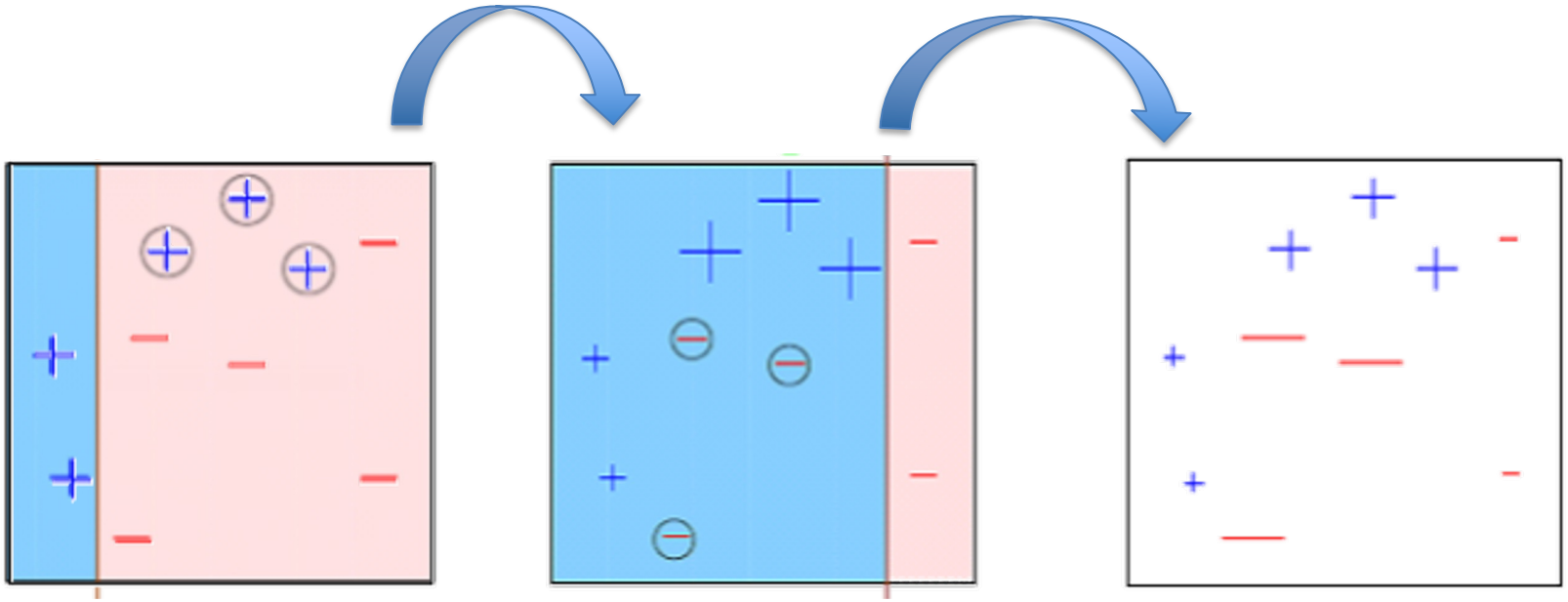




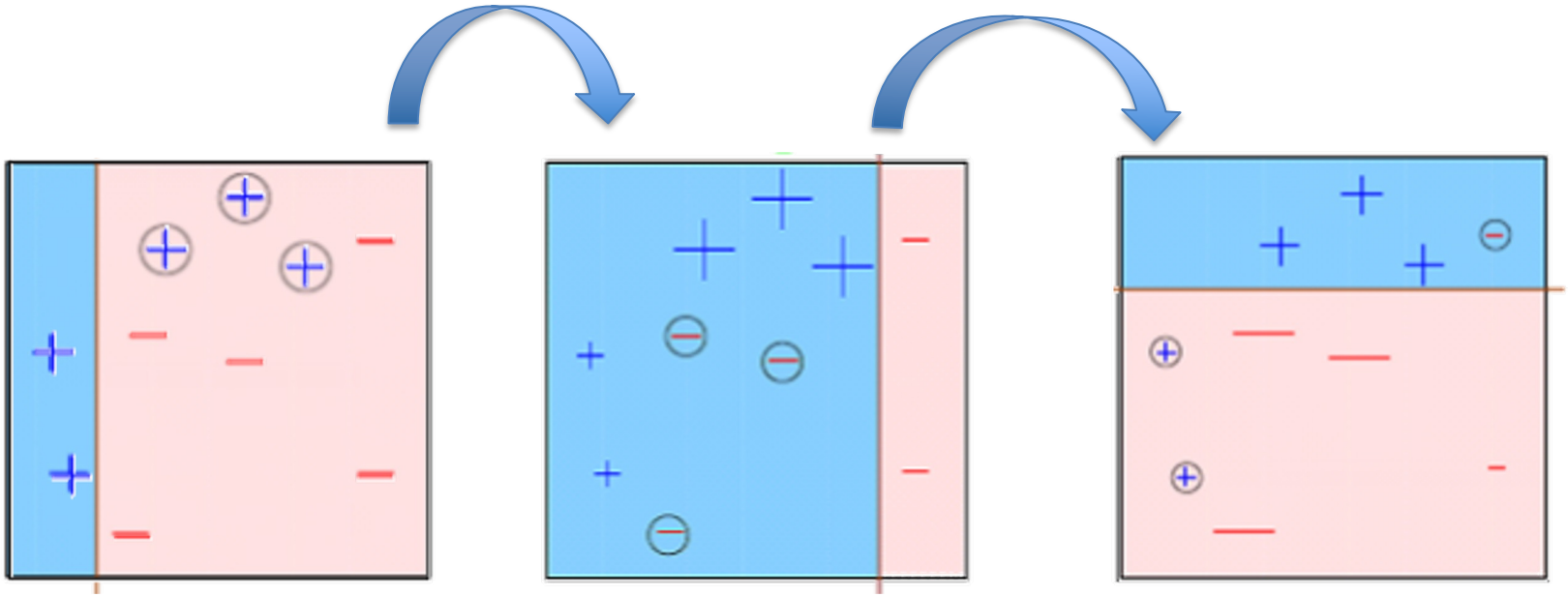
Half the distribution (~effort) goes on the incorrect examples.



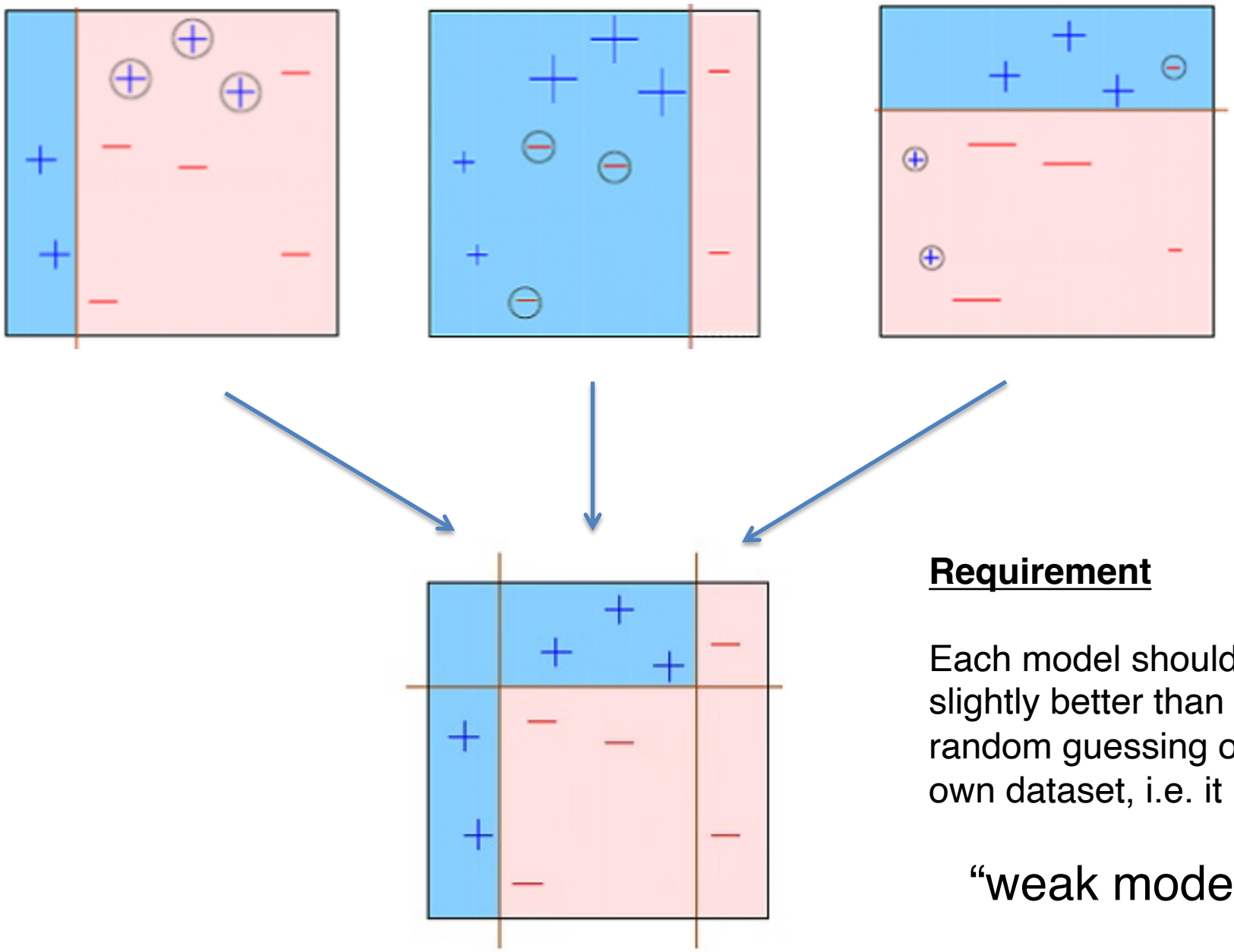
Half the distribution (~effort) goes on the incorrect examples.



Half the distribution (~effort) goes on the incorrect examples.



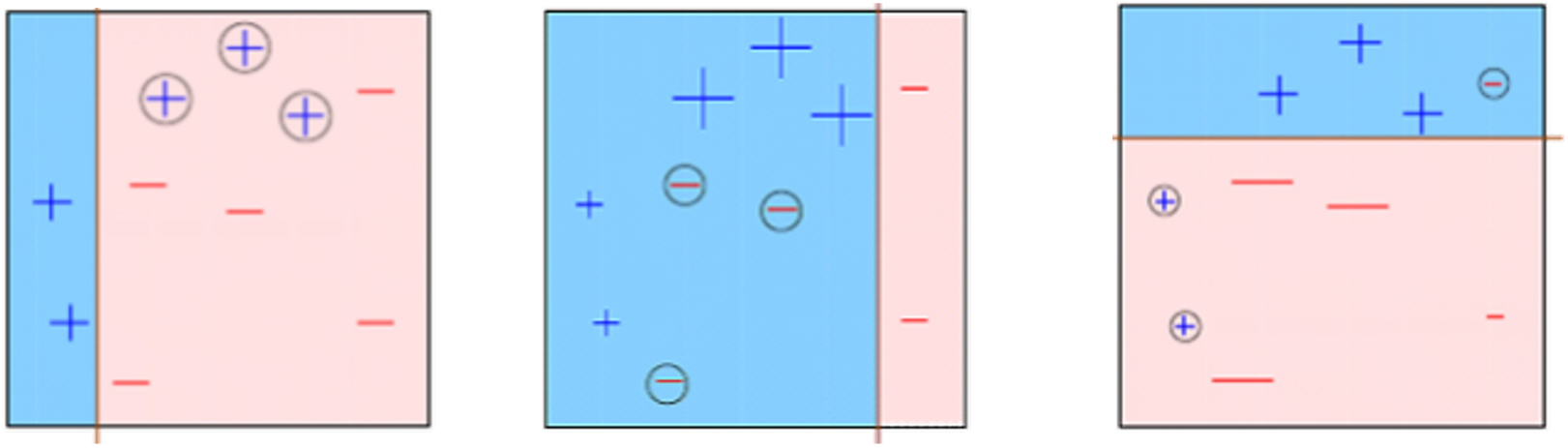
Half the distribution (~effort) goes on the incorrect examples.



Requirement

Each model should be slightly better than random guessing on its own dataset, i.e. it is a...

"weak model"



Can your classifier naturally take advantage of a weighted distribution?

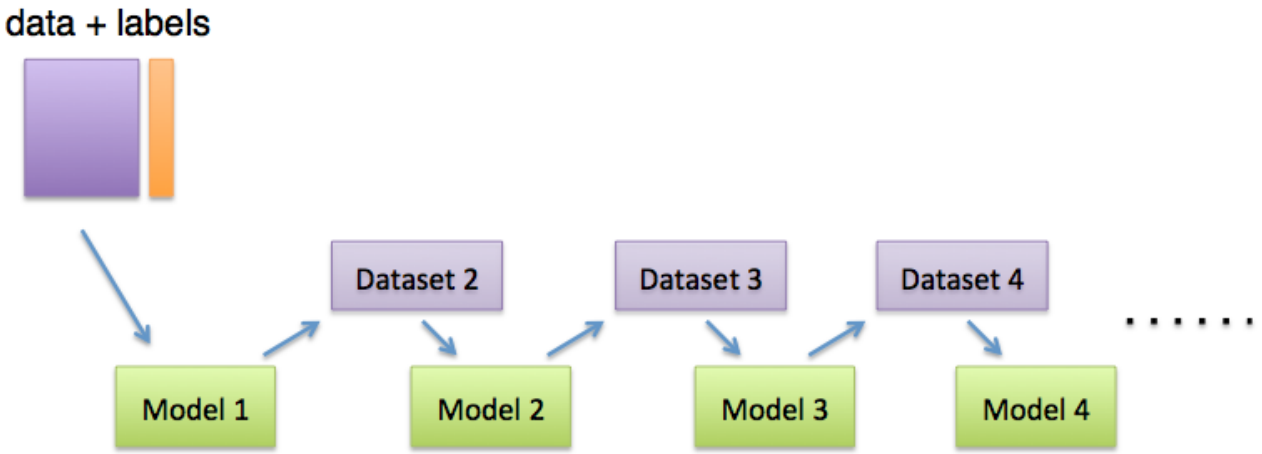
yes → Boosting by “Reweighting” (e.g. using Naive Bayes)

no → Boosting by “Resampling” (e.g. using linear discriminants)

"Boosting"

Boosting: input training data+labels $\{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_n, y_n)\}$, and required number of models M

Define a distribution over the training set, $P_1(i) = \frac{1}{n}$, $\forall i$.
for $t = 1$ to M **do**
    Build a model h_t from the training set, using distribution P_t.
    Update distribution to P_{t+1} :
        Increase the weight on examples that h_t incorrectly classifies.
        Decrease the weight on examples that h_t correctly classifies.
end for
For a new testing point $\mathbf{x}$, we take a weighted majority vote from $\{h_1, \ldots, h_M\}$.



The Boosting Family (Freund & Schapire papers 1990-97)

History in computational learning theory … Gödel Prize 2003

- two complexity classes (strongly/weakly PAC learnable) proved equivalent

- existence of algorithm predicted before invention!

- MANY extensions, e.g. multiclass, regression, ranking etc
- Original, and most well known, is “Adaboost”, **ADA**ptive **BOOST**ing

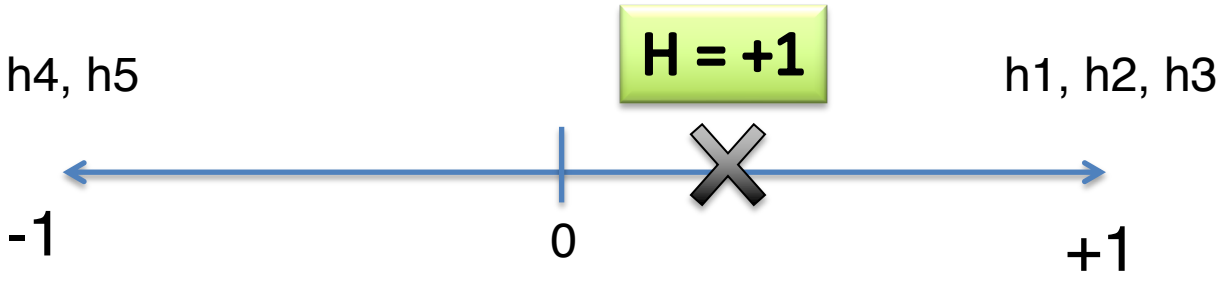
Adaboost

Each classifier ‘h(x)’ produces output in {-1,+1}
Target ‘y’ also in {-1,+1}

Weighted majority vote implemented as:

$$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x') \right)$$

Weights dictate how ‘important’ each model is.



Adaboost

Adaboost: input training data+labels $\{(\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)\}$, and required number of models M

Define a distribution over the training set, $P_1(i) = \frac{1}{n}$, $\forall i$.
for $t = 1$ to M **do**
Build a classifier h_t, using distribution P_t.
Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Update distribution to P_{t+1} :
Set $P_{t+1}(i) = P_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$
Renormalize: $P_{t+1} \leftarrow P_{t+1}/Z_t$.
end for

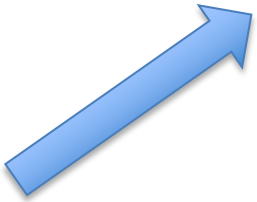
For a new testing point $(\mathbf{x}', y')$, we take a weighted majority vote, $H(\boldsymbol{x}') = sign\left(\sum_{t=1}^{M} \alpha_t h_t(\boldsymbol{x}')\right)$

Weighted majority vote

Game Theory
(Freund & Schapire 1996)

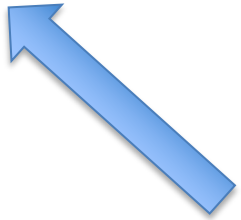
Probabilistic Models
(Lebanon & Lafferty 2001)
(Edakunni, Kovacs & Brown 2011)

Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
Update distribution to P_{t+1} :
Set $P_{t+1}(i) = P_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$
Renormalize: $P_{t+1} \leftarrow P_{t+1}/Z_t$.



Functional Gradient Descent
(Mason et al 2001)

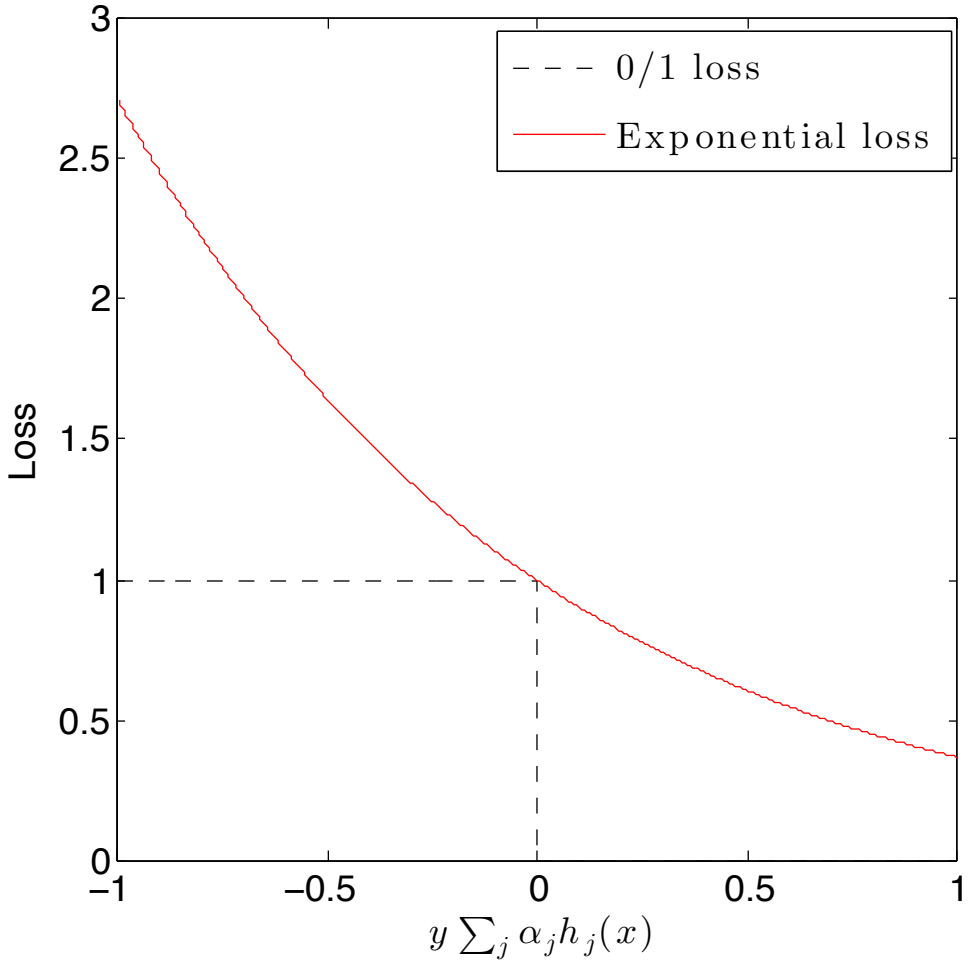
Optimizing Bregman divergences
(Collins 2000)



Dynamical Systems
(Rudin et al 2004)

Let's derive it ourselves ☺

$$\frac{1}{N}\sum_{i=1}^{N}\delta(H(x_i)\neq y_i) \leq \frac{1}{N}\sum_{i=1}^{N} e^{-y_i \sum_t \alpha_t h_t(x_i)}$$



$$E_1 = \frac{1}{N}\sum_{i=1}^{N} e^{-y_i h_1(x_i)}$$

…When fitting the first model.

$$E_2 = \sum_{i=1}^{N} \underbrace{\frac{1}{N} e^{-y_i \alpha_1 h_1(x_i)}}_{constant} e^{-y_i \alpha_2 h_2(x_i)}$$

… When fitting the second model.

$$E_2 = \sum_{i=1}^{N} w_2(i) e^{-y_i \alpha_2 h_2(x_i)}$$

Identical structure!

Greedy fitting of exponential.

$$E_3 = \sum_{i=1}^{N} \underbrace{\underbrace{\underbrace{\frac{1}{N}}_{w_1(i)} e^{-y_i \alpha_1 h_1(x_i)}}_{w_2(i)} e^{-y_i \alpha_2 h_2(x_i)}}_{w_3(i)} e^{-y_i \alpha_3 h_3(x_i)}$$

Recursive weight updates:

$$w_{t+1}(i) \leftarrow w_t(i) e^{-y_i \alpha_t h_t(x_i)}$$

Re-normalize to have distribution at each step:

$$P_{t+1}(i) \leftarrow \frac{P_t(i) e^{-y_i \alpha_t h_t(\boldsymbol{x}_i)}}{Z_t} \qquad where\ Z_t = \sum_{i=1}^{n} P_t(i) e^{-y_i \alpha_t h_t(\boldsymbol{x}_i)}$$

At **each** step *t*, we are minimizing:

$$E_t = \sum_{i=1}^{n} P_t(i) e^{-\alpha_t y h_t(\boldsymbol{x}_i)}$$

$$\alpha_t^* = \arg\min_{\alpha_t} \left\{ E_t \right\} = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

Epsilon is the weighted error rate of the classifier.

Adaboost: input training data+labels $\{(\mathbf{x}_1, y_1)...(\mathbf{x}_n, y_n)\}$, and required number of models M

Define a distribution over the training set, $P_1(i) = \frac{1}{n}$, $\forall i$.
for $t = 1$ to M **do**
    Build a classifier h_t, using distribution P_t.
    Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$
    Update distribution to P_{t+1} :
        Set $P_{t+1}(i) = P_t(i)\exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))$
        Renormalize: $P_{t+1} \leftarrow P_{t+1}/Z_t$.
end for

For a new testing point $(\mathbf{x}', y')$, we take a weighted majority vote, $H(\boldsymbol{x}') = sign\left(\sum_{t=1}^{M} \alpha_t h_t(\boldsymbol{x}')\right)$

"AdaBoost with trees is the best off-the-shelf classifier in the world."
(Leo Breiman 1998)

Ensemble Learning: Summary

- Treats models as a committee: build in parallel / sequential
- Standard tool for data science and ML
- Often tend to work extremely well “off-the-shelf”
  – very little need for parameter tuning.