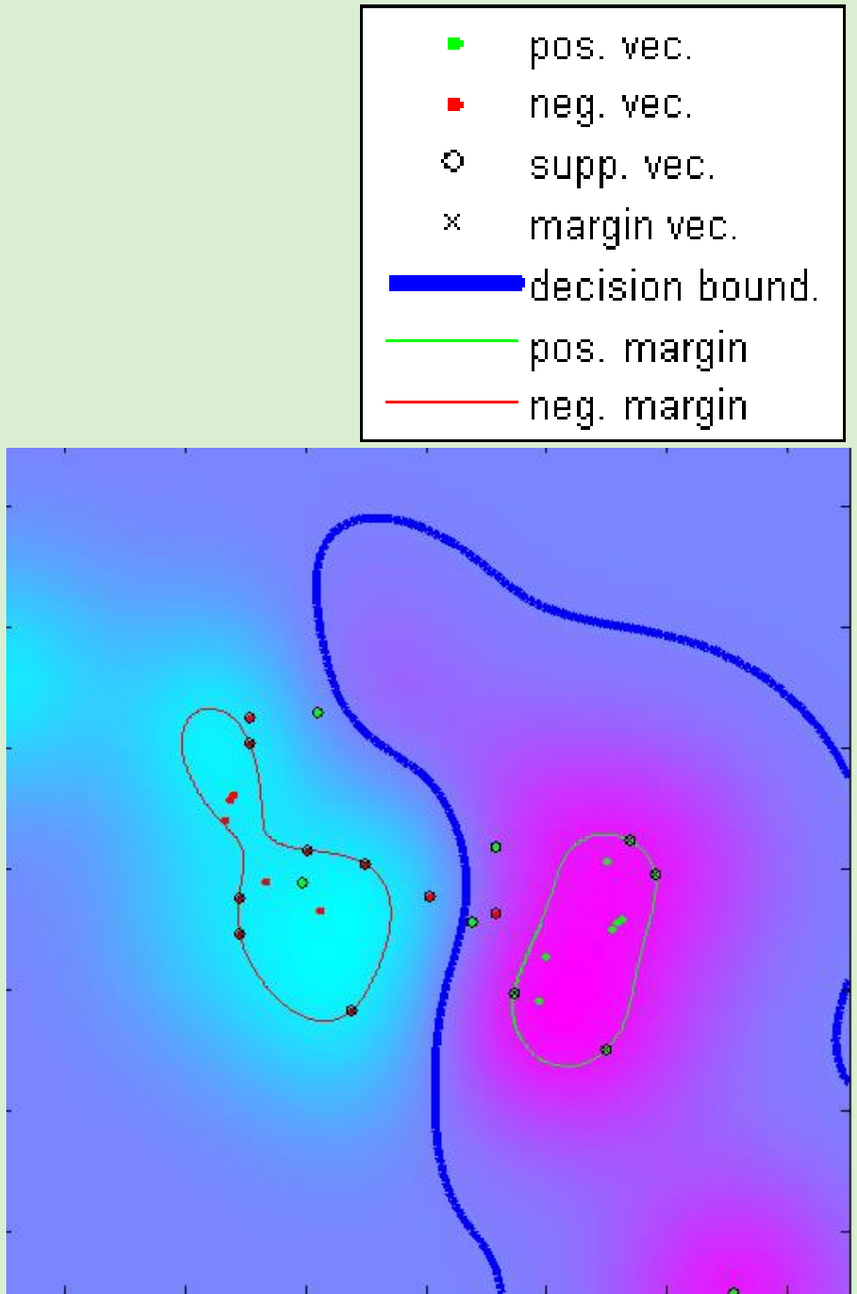
Introduction to Machine Learning

Dor Bank

Lecture #9
Tree based algorithms

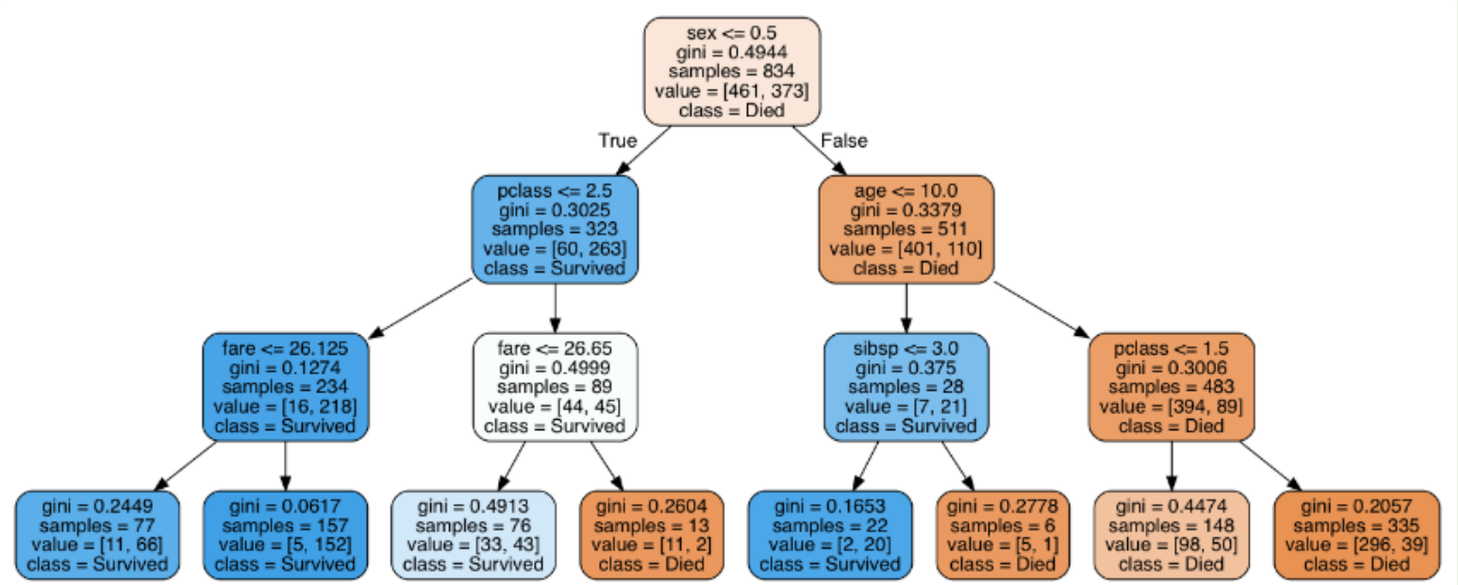
On last time - SVM

- Constrained optimization
- SVM
- Kernel methods



On last time - DT

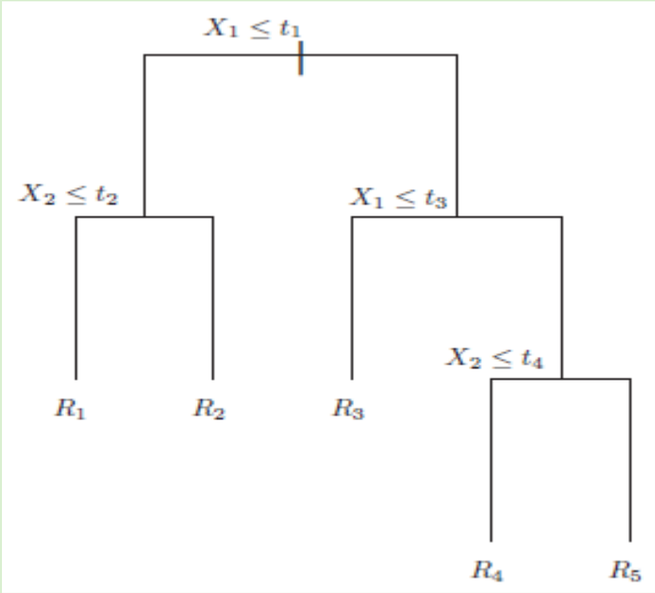
- Architecture
- Learning regression decision trees
- Learning classification decision trees
- Pruning

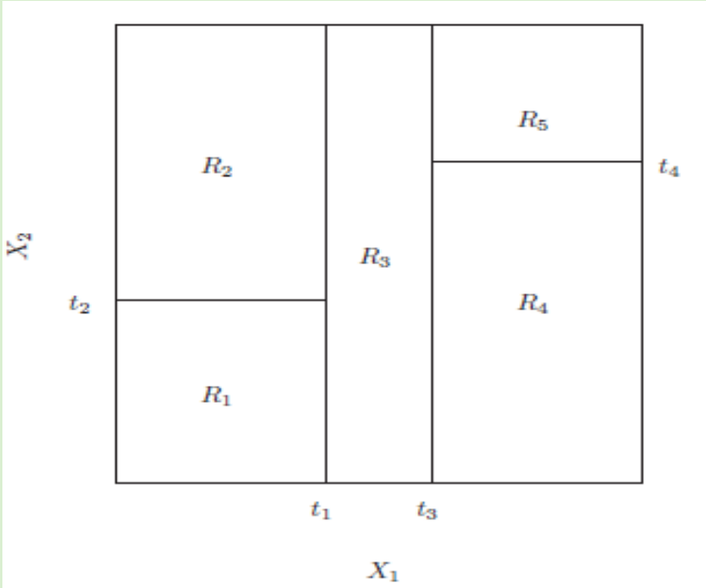


Titanic Decision Tree learned by Scikit

Decision Trees - recap

- Training the model includes
  - Splitting criterion
    - Regression: MSE
    - Classification: Entropy/Gini index
  - Prediction
    - Regression: mean
    - Classification: majority
  - Partition stop
    - Max depth, min samples, …

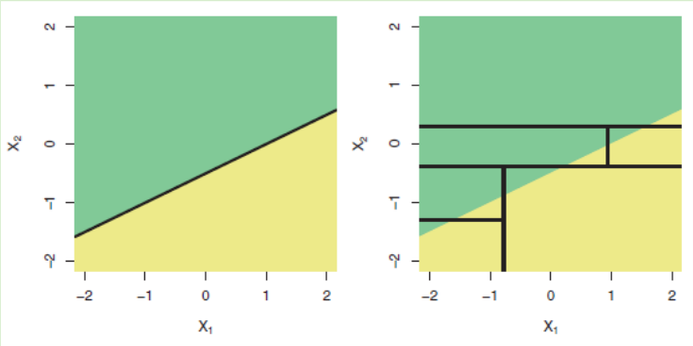




Introduction to Statistical Learning – G. James, D. Witten, T. Hastie, R. Tibshirani

Decision Trees - Summary

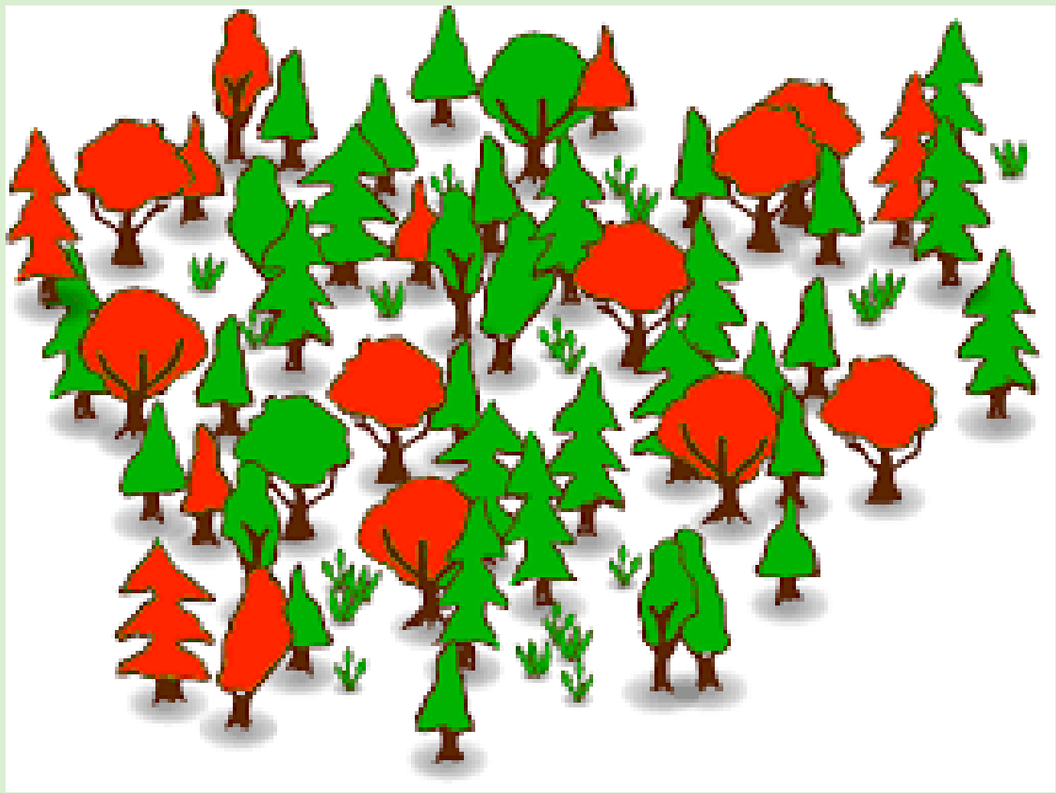
- Pros
  - Simple
  - Intuitive
  - “natural” Nan dealing
  - No need for dummy variables
- Cons
  - Bad model
  - High variance – small change in data might change the entire tree
  - Can’t deal with simple model such as linear



Introduction to Statistical Learning – G. James, D. Witten, T. Hastie, R. Tibshirani

Today – DT Ensemble methods

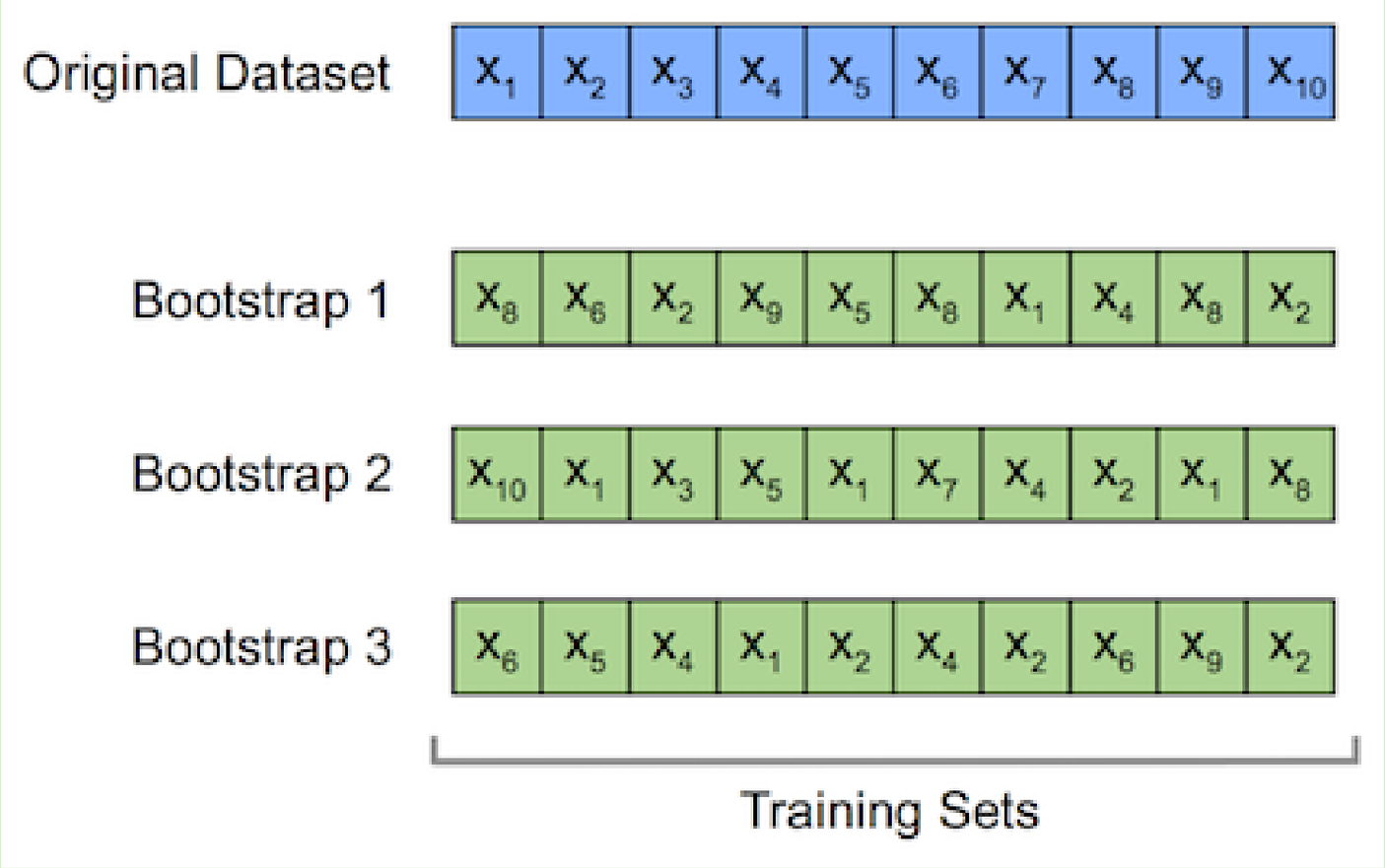
- Bagging
- Random Forest
- Boosting
  - Adaboost
  - Gradient Boosting



Bagging *(Breiman 94)*

- Training DTs is noisy and sub-optimal (suffer from high variance)

- Make K datasets of size n. For each, sample each sample with probability $\frac{1}{n}$ (with replacement)

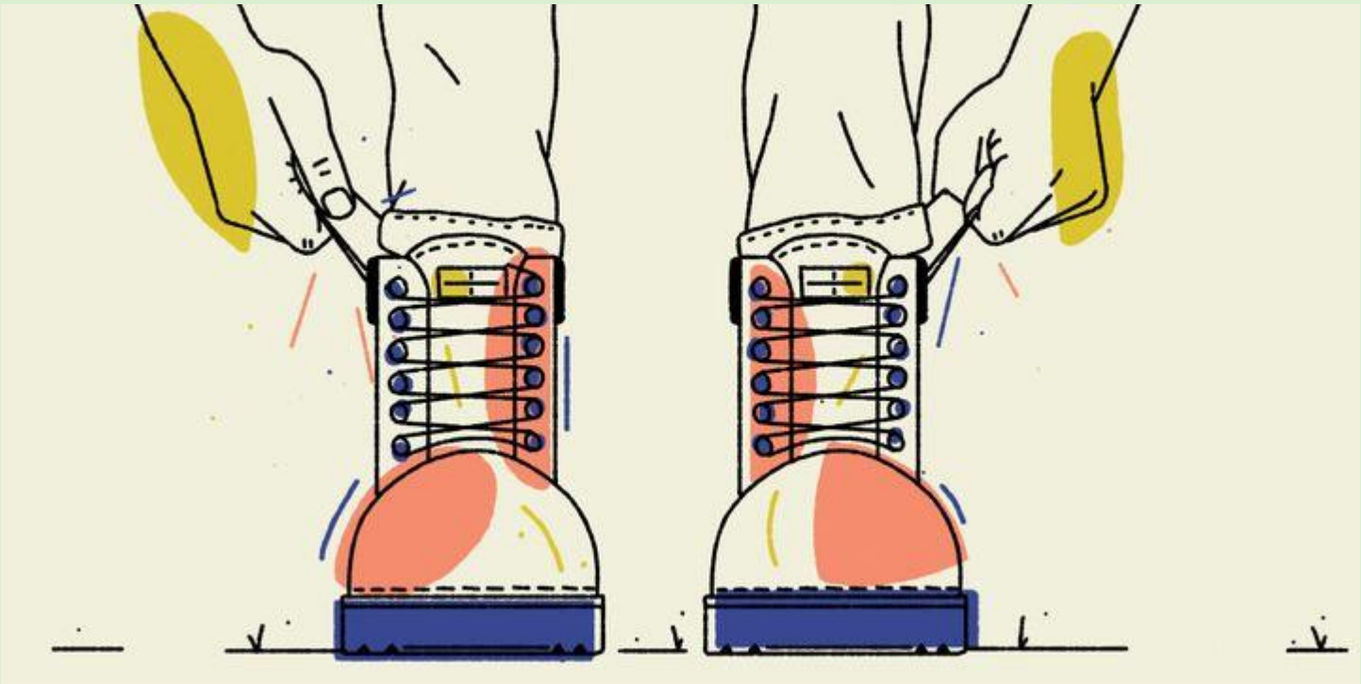


http://rasbt.github.io/mlxtend/user_guide/evaluate/bootstrap_point632_score/

Bagging *(Breiman 94)*

- Bootstrap: generating a distribution from samples

Baron Munchausen’s remarkable leap

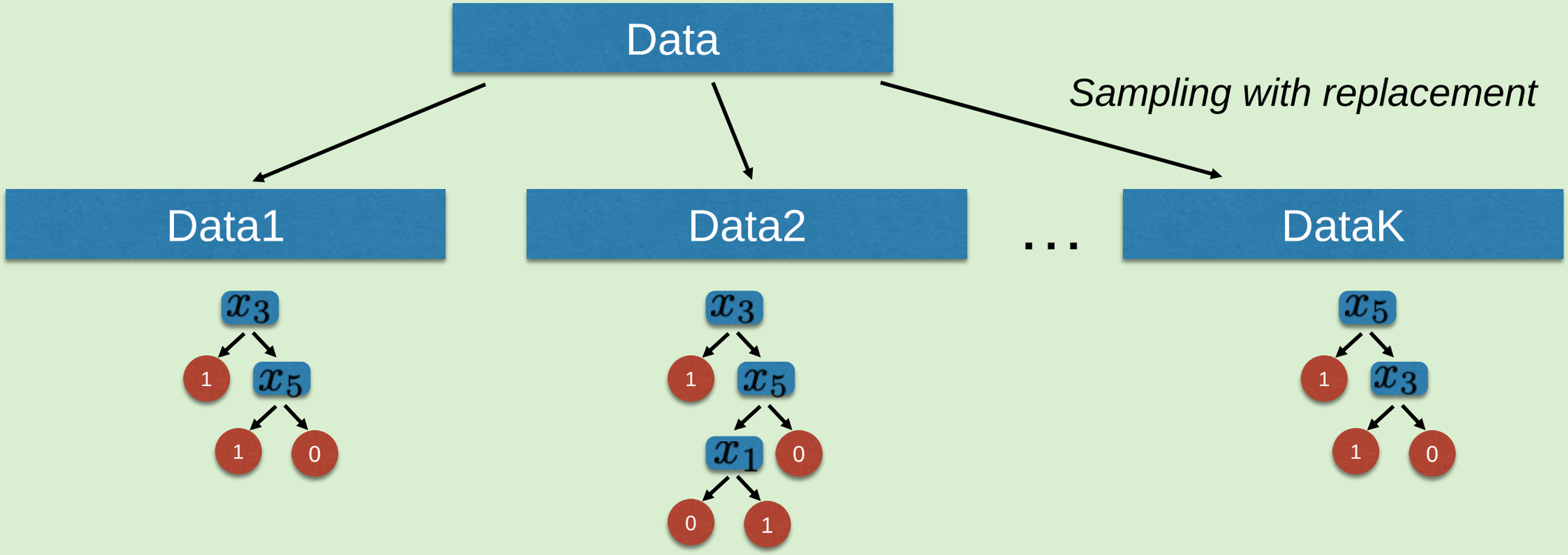


https://www.huffpost.com/entry/pull-yourself-up-by-your-bootstraps-nonsense_n_5b1ed024e4b0bbb7a0e037d4

Bagging *(Breiman 94)*

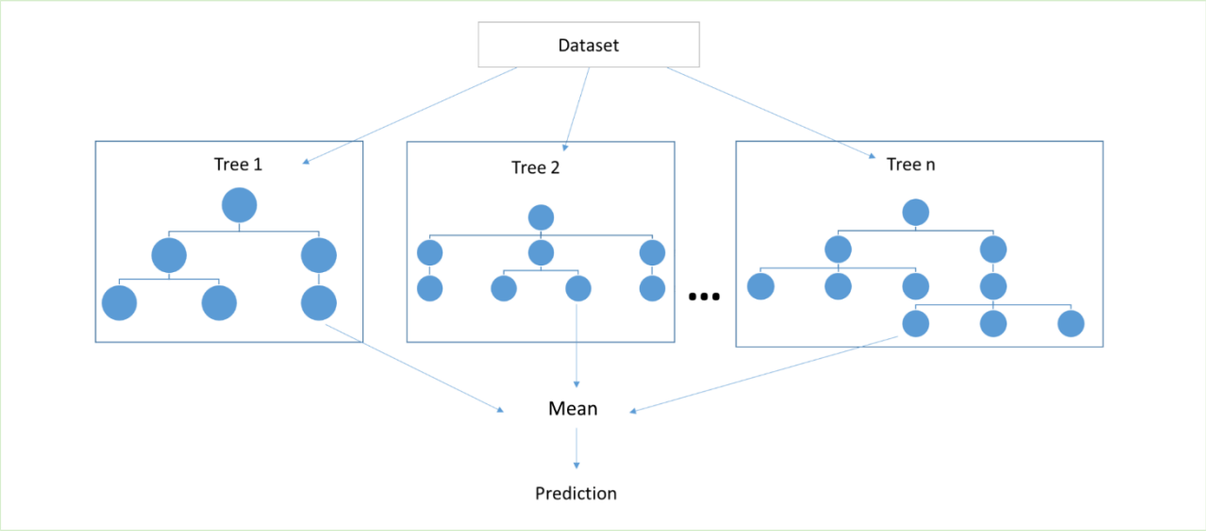
- Training DTs is noisy and sub-optimal (suffer from high variance)

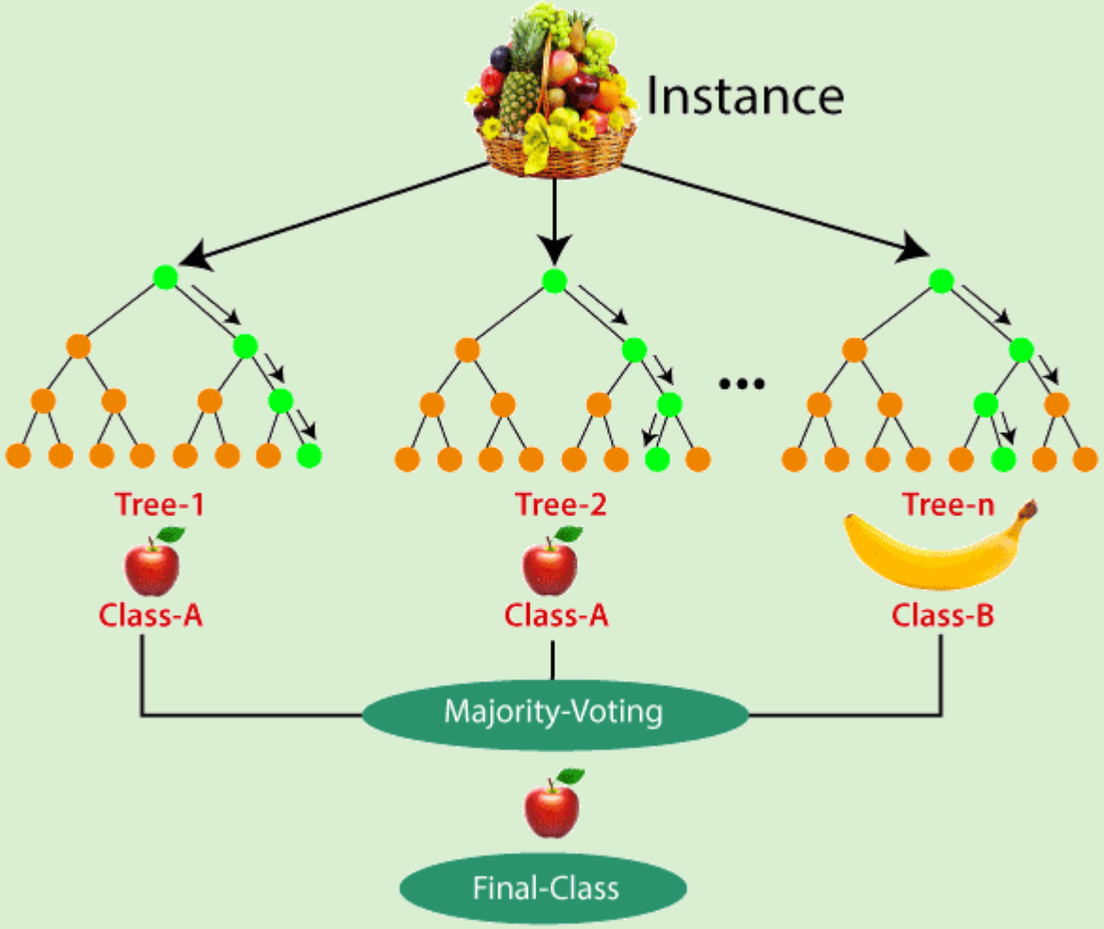
- Make K datasets of size n. For each, sample each sample with probability $\frac{1}{n}$ (with replacement)



Bagging

- We have multiple trees. For a given input x, each tree will have its own prediction
- A natural way to combine these is take majority\mean

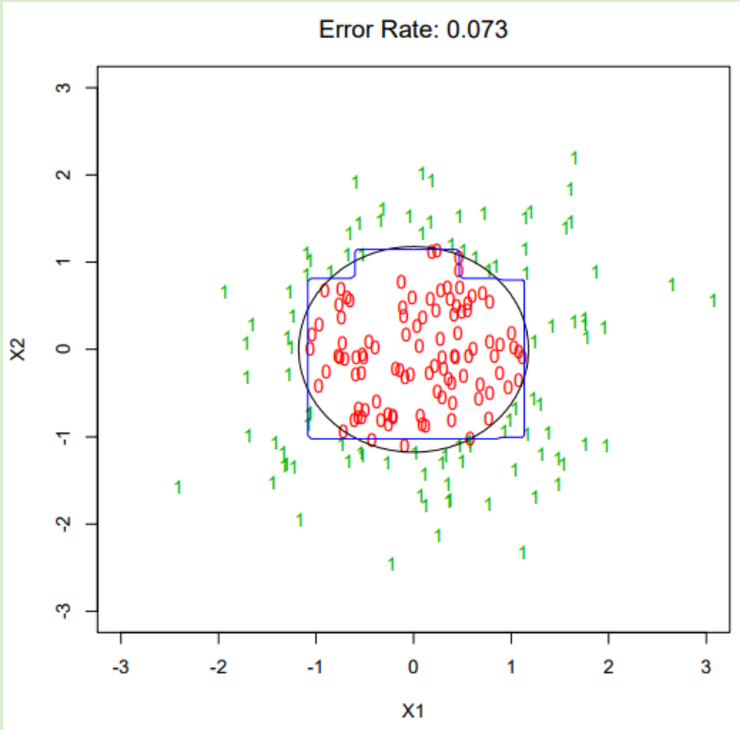




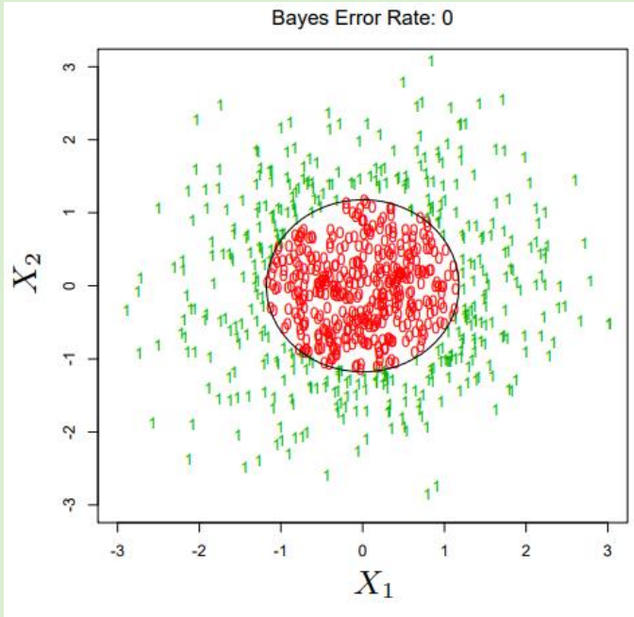
Bagging – Bias-Variance

- The variance drops significantly (approximately $O(\#trees)$)
- (very) Little bias is added due to smaller trees

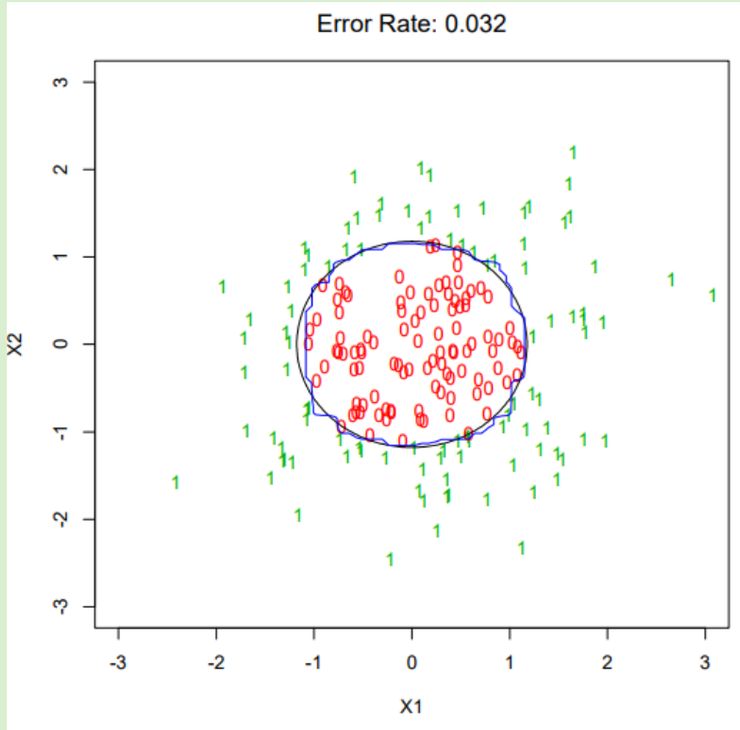
Test data –
8 depth DT



Train data and
true model
(nested sphere)



Test data –
Bagging



https://www.slideshare.net/0xdata/gbm-27891077?from_action=save

Bagging – Estimating E_{out}

- Each DT has Out-Of-Bag (OOB) samples which were not taken into account
- Probability of a sample not being chosen one time: $\frac{n-1}{n} = 1 - \frac{1}{n}$
- Probability of a sample not being chosen in a dataset: $\left(1 - \frac{1}{n}\right)^n \approx \frac{1}{e} = 0.37$
- Around 37% of the samples is OOB for each DT!
- For each sample $\mathbf{x}$, we can take all the DT that $\mathbf{x}$ is in their OOB and measure its Error!

http://rasbt.github.io/mlxtend/user_guide/evaluate/bootstrap_point632_score/

Bagging – Interpretability

- Bagging improves the DT E_{out} at the expense of interpretability
- Variable Importance
    - One way to gain intuition is to get for each feature, the average MSE\impurity dropped by it in training

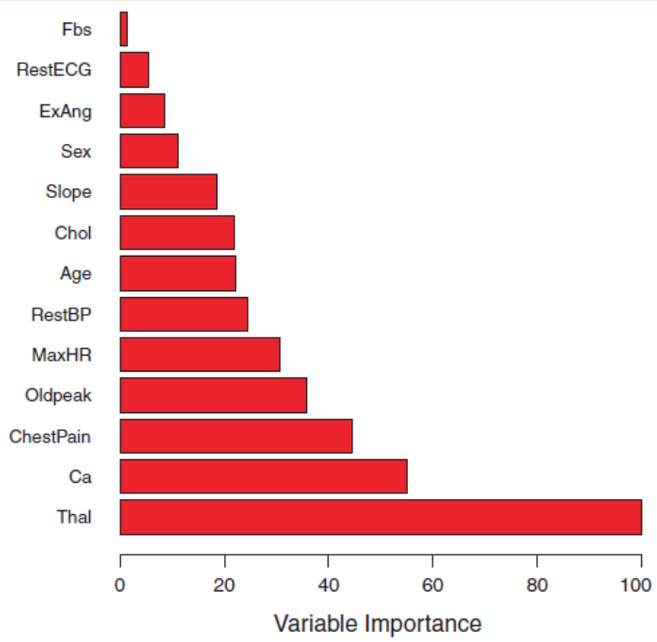
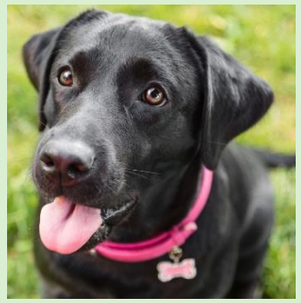


FIGURE 8.9. *A variable importance plot for the* `Heart` *data. Variable importance is computed using the mean decrease in Gini index, and expressed relative to the maximum.*

Correlated features

- One issue with Bagging is that certain “strong” features may be picked by all trees, and correlative (bit weaker) features will never be chosen!

- Example: differentiating between dogs and cats.
- We have 2 wonderful features: dog/cat eyes & dog/cat ears (the first is a bit stronger)

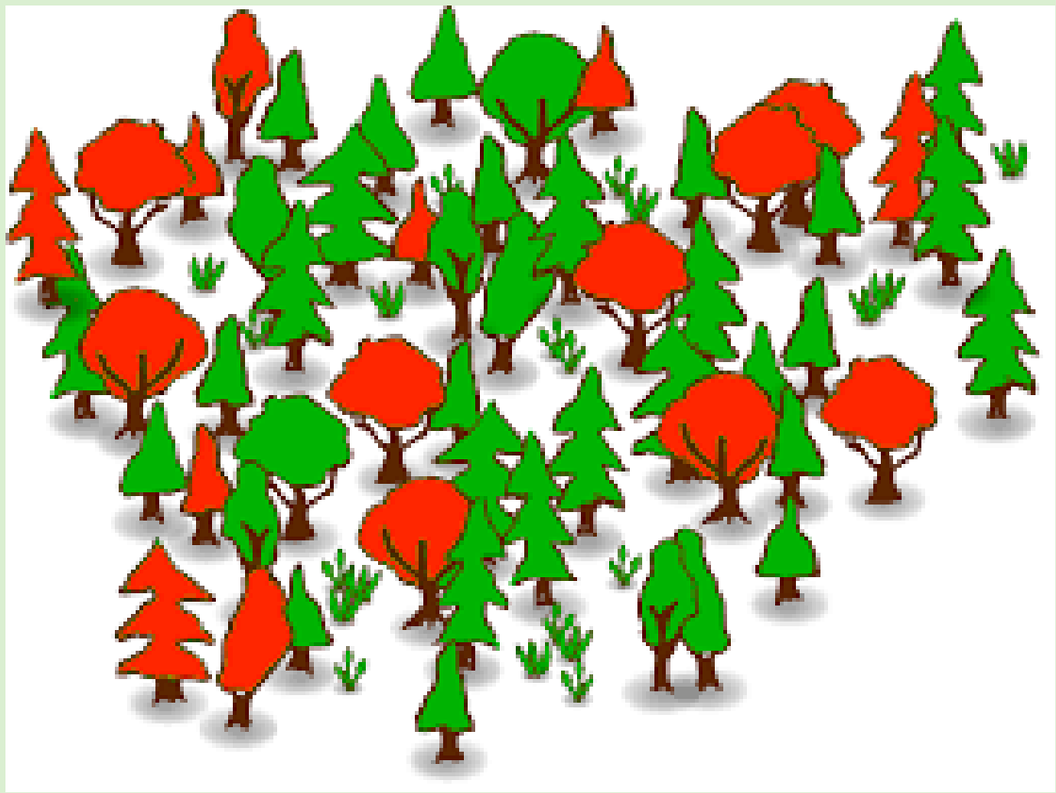
#sample	Eyes	Ears
1	Dog	Dog
2	Cat	Cat
3	Cat	Cat
…	…	…

- We learn to classify perfectly solely by the first feature
- Suddenly:

?

Today – DT Ensemble methods

- Bagging
- Random Forest
- Boosting
  - Adaboost
  - Gradient Boosting



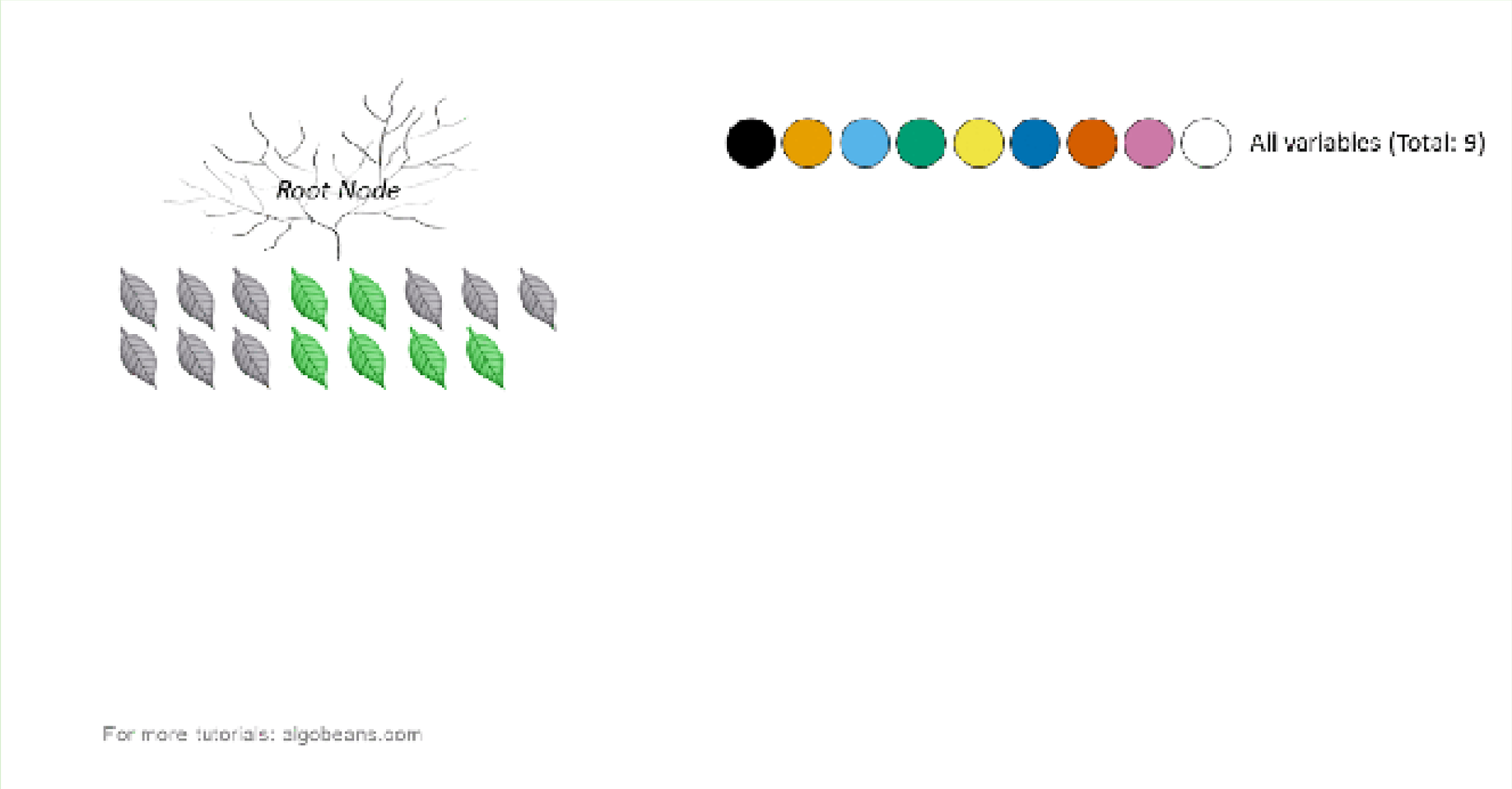
Random Forests (Breiman 2001)

- Random Forests overcome the problem of correlative features by considering random subsets of splits at every splitting decision (usually, $\sqrt{|\#features|}$ are considered)
- A very competitive method!
- Some loss of interpretability

	$feature_1$	$feature_2$	$feature_3$
$sample_1$	0.1	0.12	0.1
$sample_2$	0.4	0.21	0.3
$sample_3$	-0.2	0.11	0.4
$sample_4$	1.1	0.02	0.13
$sample_5$	0.35	0.3	-0.1
$sample_6$	0.12	0.2	0.08

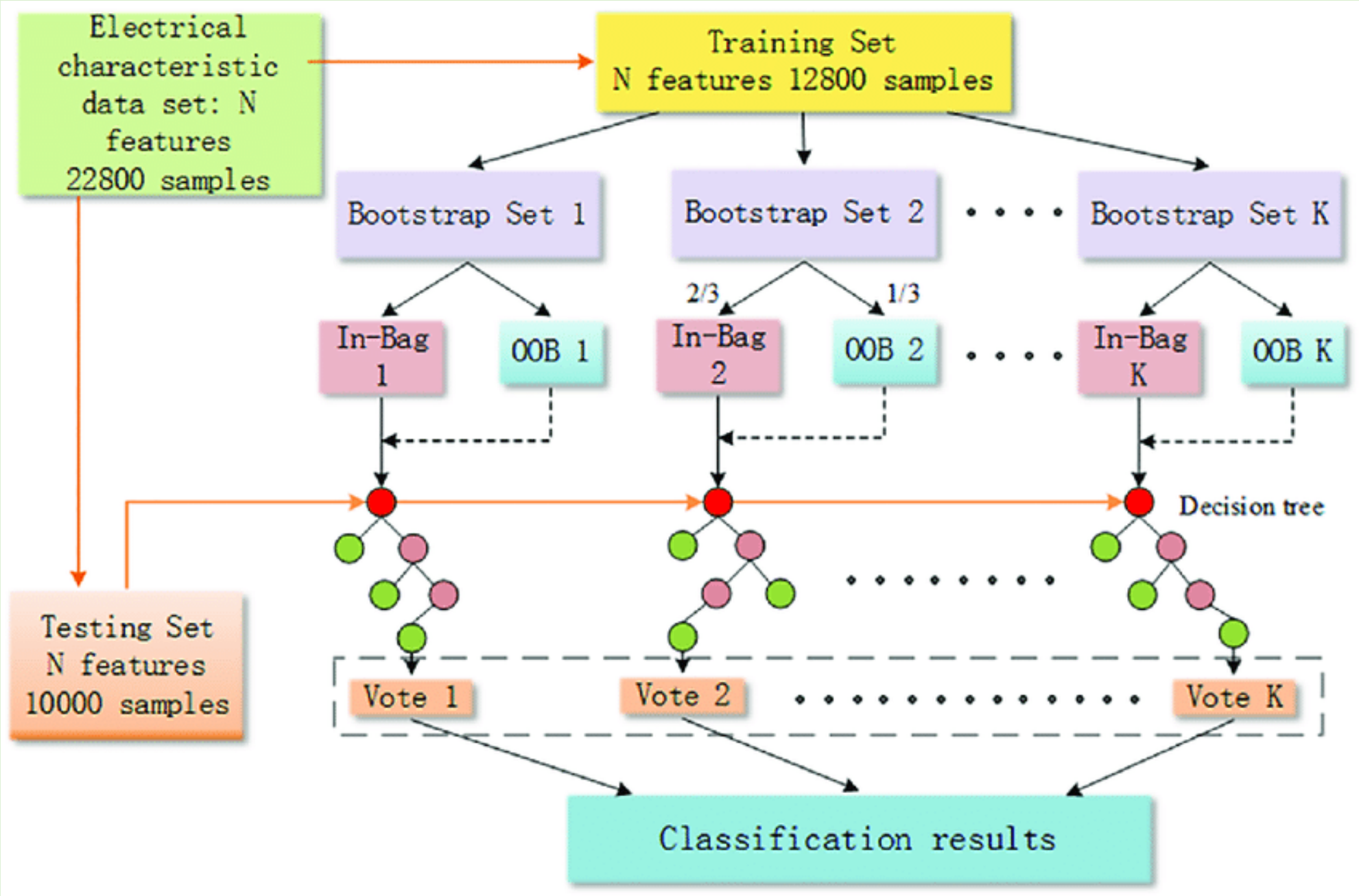
Random Forests (Breiman 2001)

- Different data sets are generated via bootstrap
- Each DT is created for a different dataset, where each split considers a subset of the features



https://www.kdnuggets.com/lmth.lairotut-lanimirc-tserof-modnaer/2016/09

Random Forests (Breiman 2001)

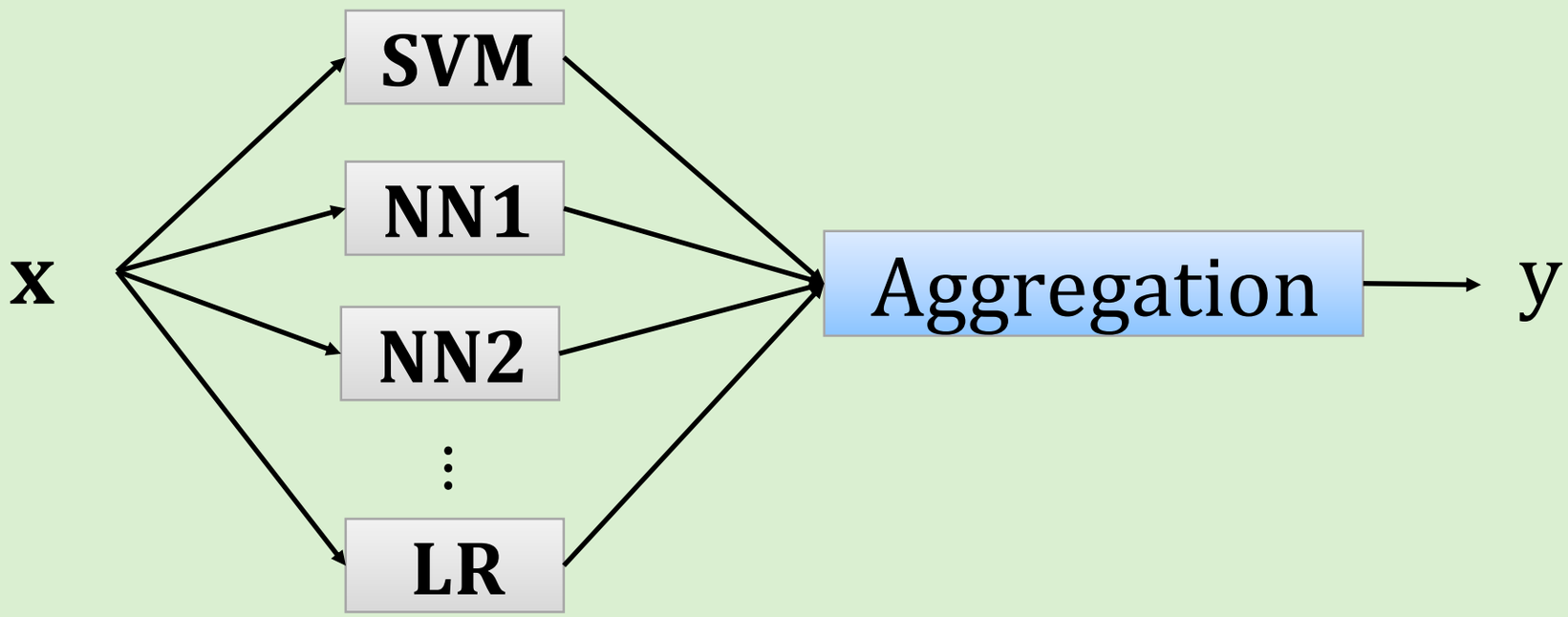


Random Forests (Breiman 2001)

- Question: how many trees should we use?
- Answer: Statistically, the more the merrier. More trees would reduce the variance. The limitation comes from computational costs.
- Question: how big should the trees be?
- Answer: **BIG**. The bigger the better, as long as you use many trees, the variance will reduce. The limitation comes again from computational costs.

Ensembles in general

- Both bagging and random forests combine multiple decision trees to obtain a more accurate one.
- This approach is known as an **ensemble method**
- In general, any group of models can be ensembled together
- VERY common at Kaggle competitions!

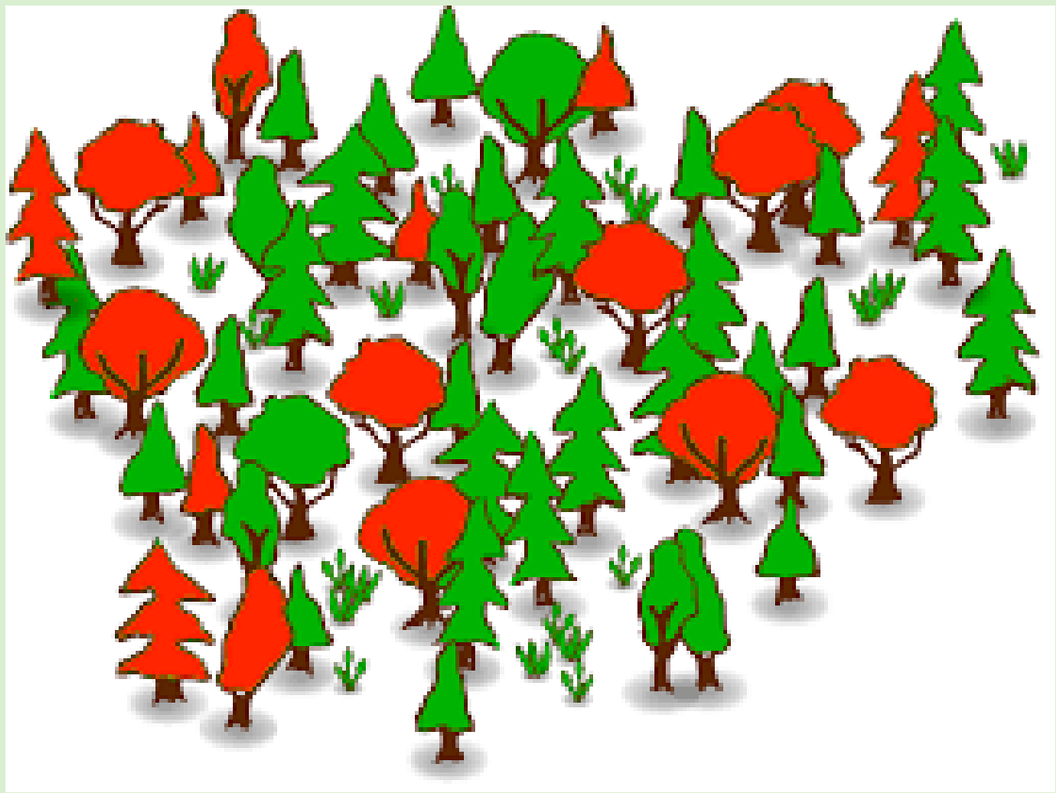


Ensembles in general

- The aggregation applied is usually mean/majority
- Thus means, that each classifier trains and classifies independently from the others
- Can we do better?

Today – DT Ensemble methods

- Bagging
- Random Forest
- Boosting
  - Adaboost
  - Gradient Boosting

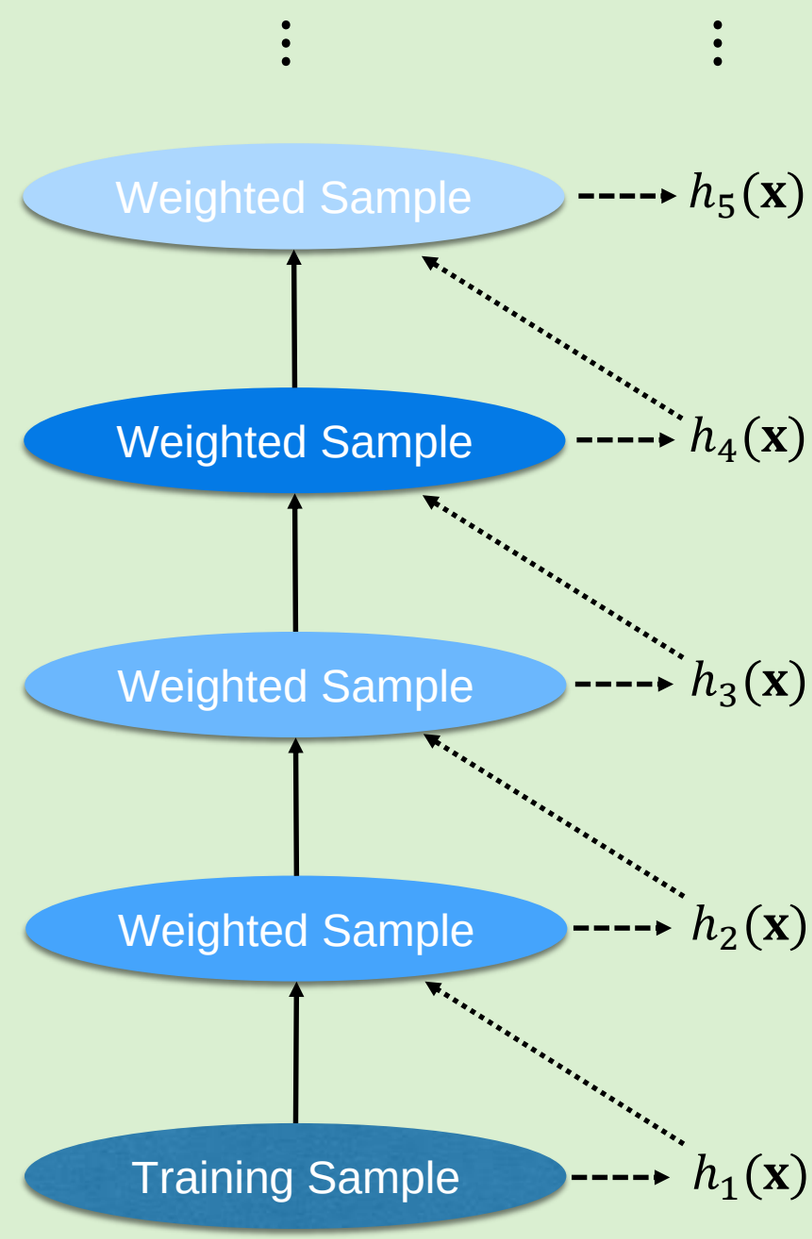


Boosting

- Breakthrough invention of Freund & Schapire (1997)
- Instead of building an ensemble and aggregate them, build them sequentially in a “smart” way
- Average many trees, each grown to a weighted versions of the training data
- Using decorrelated trees, by focusing on regions missed by past trees
- Final Classifier is based on a weighted average of the classifiers

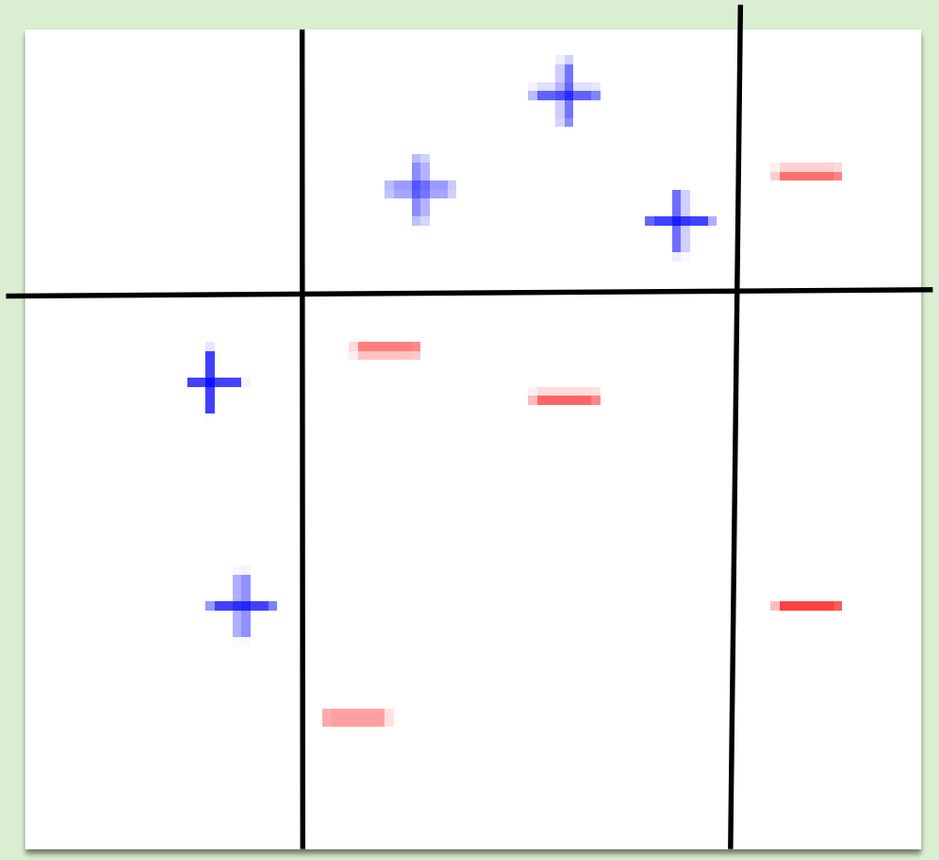
$$\hat{f}(\mathbf{x}) = \text{sign}\left[\sum_i \alpha_i h_i(\mathbf{x})\right]$$

- Note: $y \in \{-1, +1\}$. We will get to regression later on



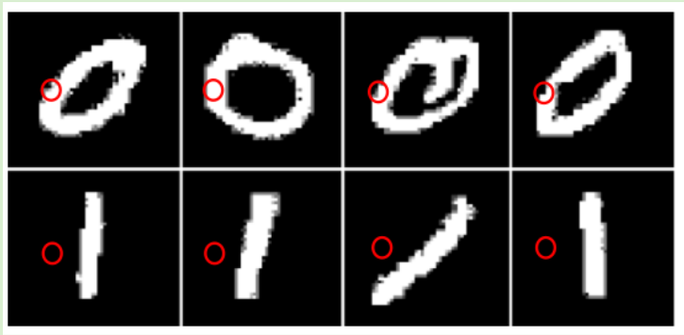
Boosting classification - Classify this!

- Can use RBF, but will possibly overfit
- Intuitively, can be solved by passing horizontal and vertical lines
- Decision trees do something similar, but construction is heuristic without guarantees



Combining Rules of Thumb

- Often, it is easy to come up with simple classifiers that can perform reasonably well.
- **Spam**: If “prince” appears then Spam
- **Digits**: If pixel (i, j) is greater than threshold, then “0”



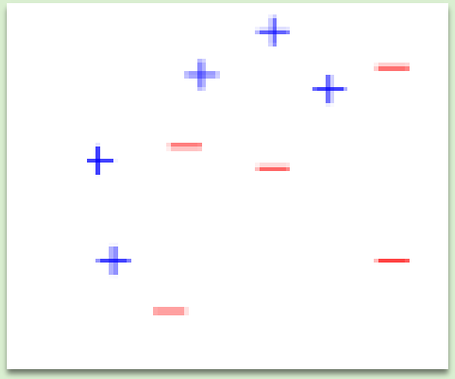
Source: Hazan and Singer

- **Face detection**: hair darker than face, eye lighter than nose

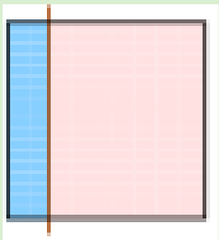
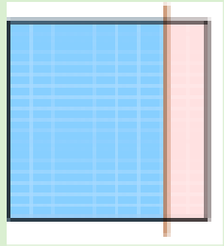
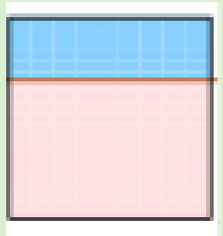
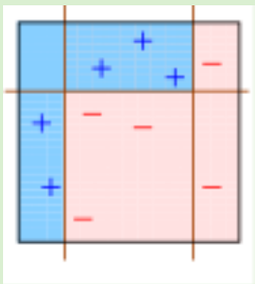
Combining Rules of Thumb

- How do we combine rules of thumb?
- Technically: if we have a set of classifiers how do we combine these
- Important: Each of these classifiers on its own might not be very accurate. We refer to it as **“weak”**
- A weak classifier is one whose error rate is only slightly better than random guessing
- **There may be infinitely many weak classifiers (e.g., all threshold functions)**

Boosting Accuracy



- How can we learn if we can only train 1-depth DTs (vertical and horizontal lines)?
- How do we combine these into a single hypothesis?
- Weighted Combination!

sign (4 *  +6 *  +9 * ) = 

- How do we find the lines?
- How do we learn the weights?

AdaBoost!

AdaBoost - High Level

- Input: training set D, Weak learner WL, rounds T
- Initialize dataset "probability" weights $w^{(1)}$
- **for** $t = 1 \ldots T$:
  - Train weak learner h_t on D with weights $w^{(t)}$
  - Set classifier weight α_t that is high if h_t is accurate
  - Set $w^{(t+1)}$ to give larger probability to each x_i where h_t errors
- Return $\hat{f}(\mathbf{x}) = \mathrm{sign}[\sum_t \alpha_t h_t(\mathbf{x})]$

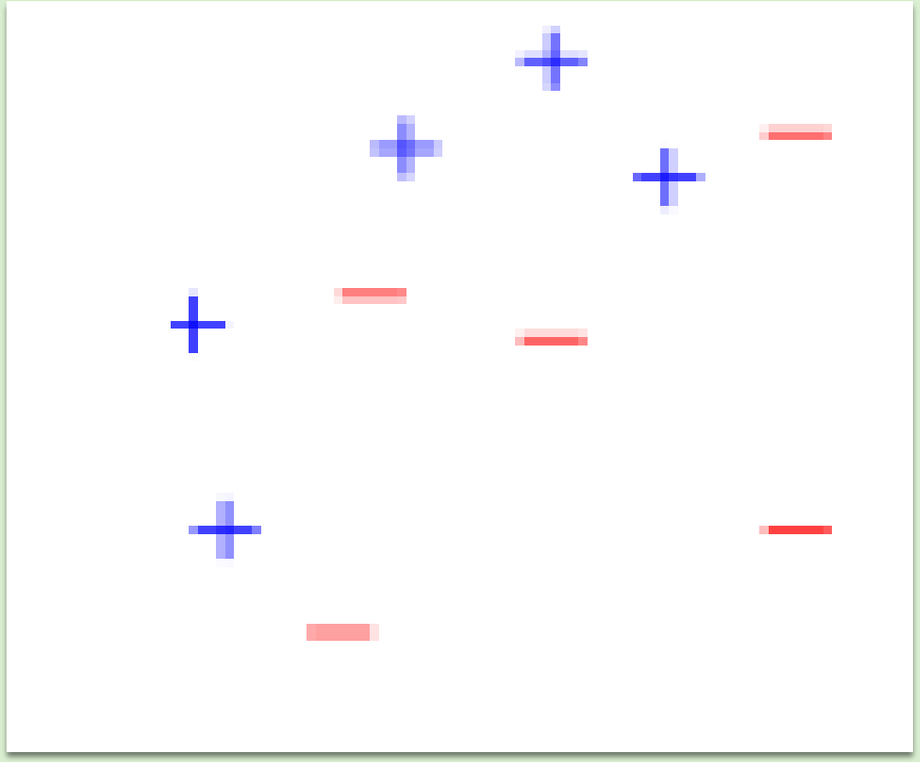
More accurate h_t are more influential

AdaBoost - Details

- Initialize the dataset weights: $\mathbf{w}_i^{(1)} = \frac{1}{n}, \quad \forall i \in \{1 \dots n\}$
- For $t = 1 \dots T$:
  - $h_t = WL(w^{(t)}, D)$ Return “weak” hypothesis
  - $\epsilon_t = Err_{w^{(t)},D}[h_t] = \sum_i w_i^{(t)} I[y_i \neq h_t(\mathbf{x}_i)]$ Empirical Error Probability
  - $\alpha_t = \frac{1}{2} \ln \frac{1-\epsilon_t}{\epsilon_t}$ $\alpha_t \geq 0$ for WL. High for “good” h_t!
  - $\forall i.\ \mathbf{w}_i^{(t+1)} = \frac{\mathbf{w}_i^{(t)} e^{-\alpha_t y_i h_t(\mathbf{x}_i)}}{\sum_j \mathbf{w}_j^{(t)} e^{-\alpha_t y_j h_t(\mathbf{x}_j)}}$ Update $\mathbf{w}$. Put large weight on errors. Normalize to $\sum_i \mathbf{w}_i = 1$.
- Return $\hat{f}(\mathbf{x}) = \text{sign}[\sum_t \alpha_t h_t(\mathbf{x})]$

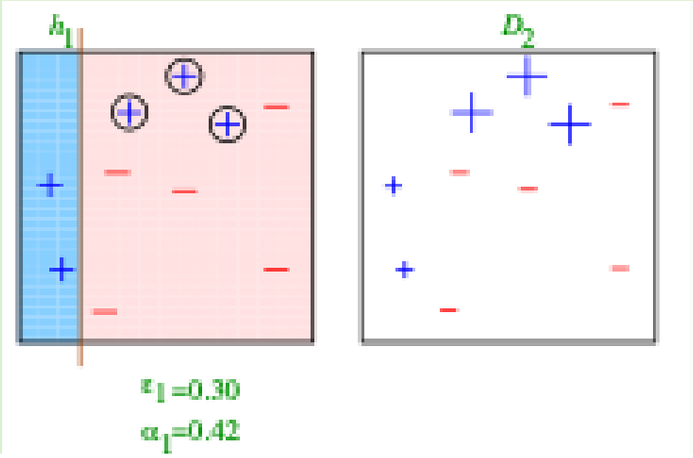
AdaBoost Example

Using DT of size 1 (stumps)!

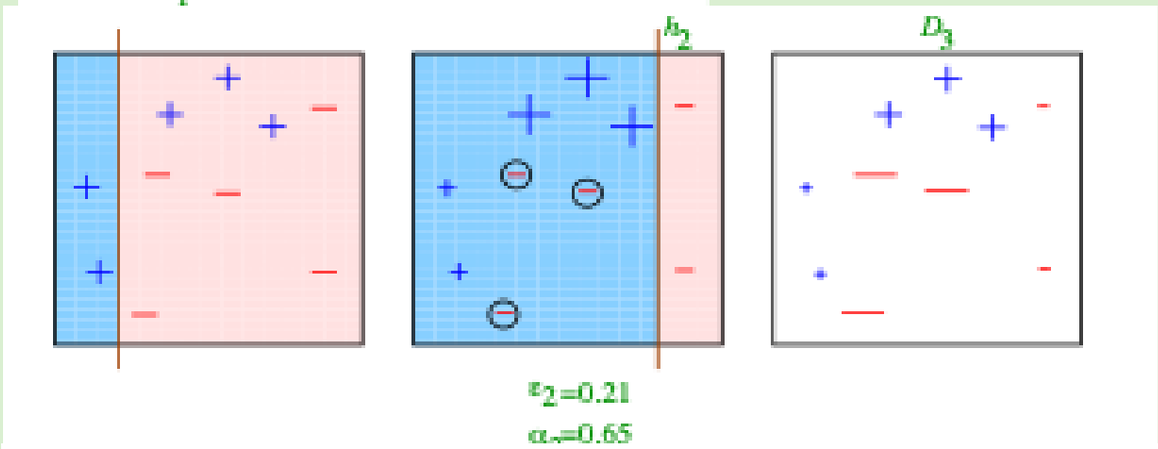


AdaBoost Example

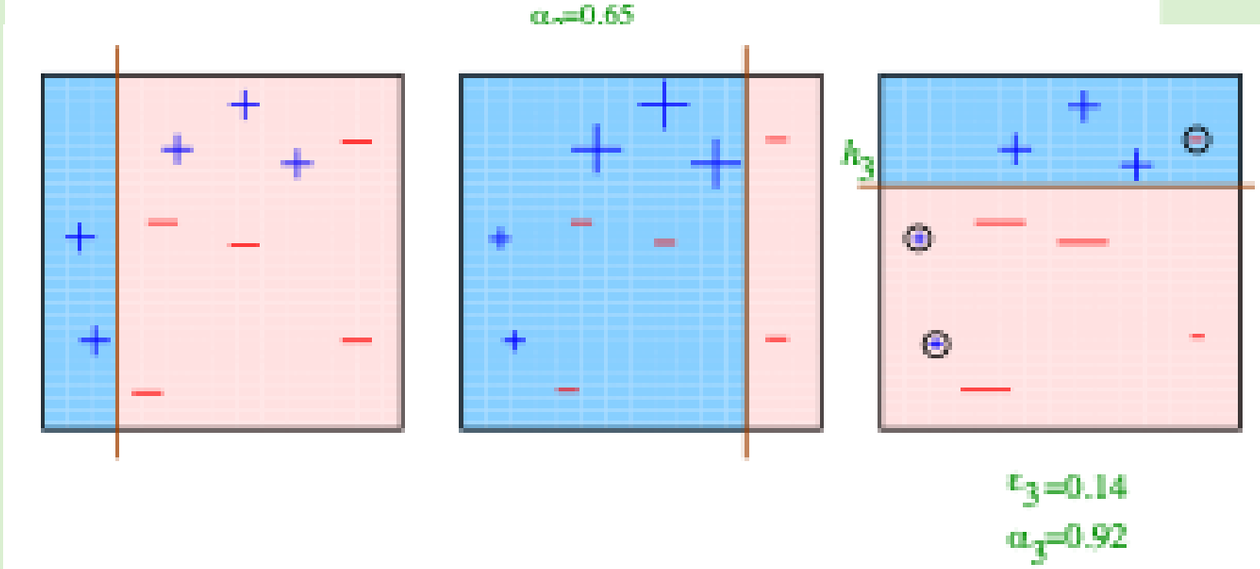
Round 1:



Round 2:



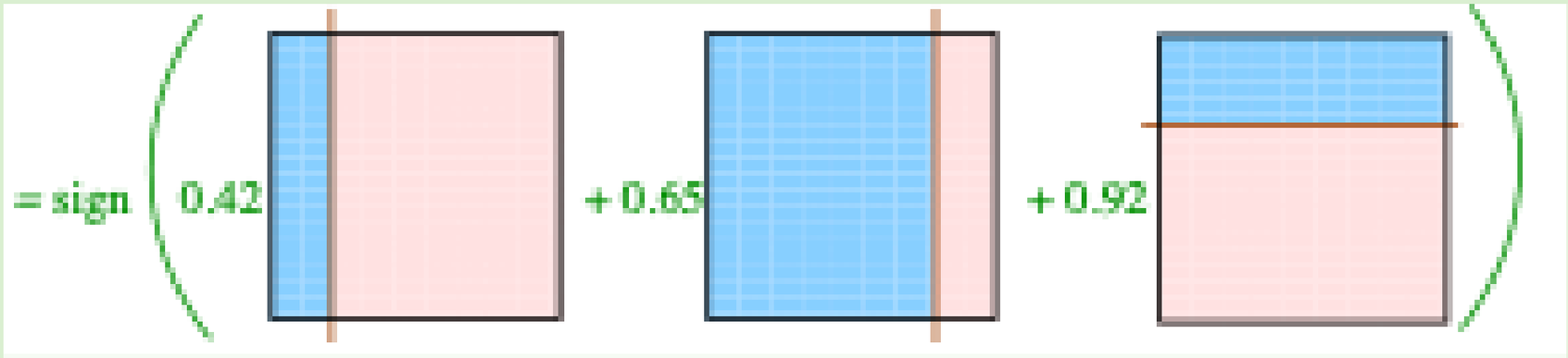
Round 3:

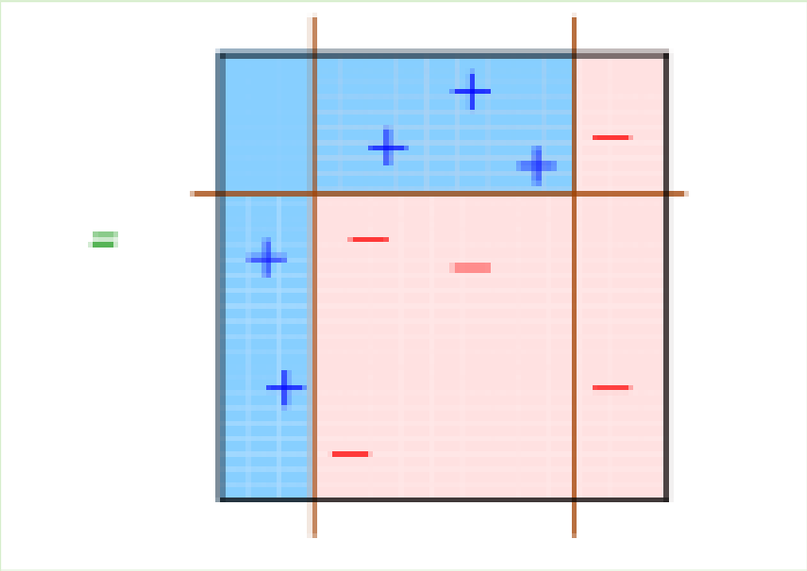


AdaBoost Example

- Final Classifier:

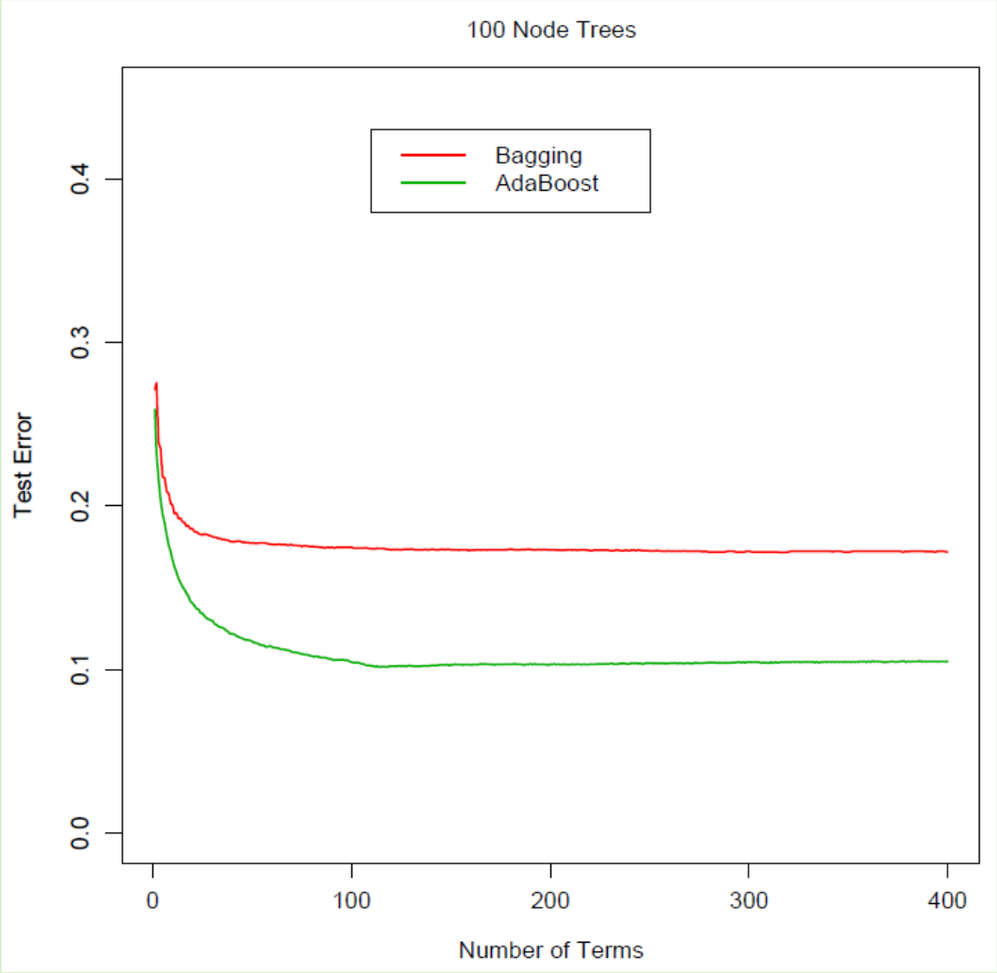
$\hat{f}(\mathbf{x})$





AdaBoost VS Bagging

- 2000 points from nested sphere in $\mathbb{R}^{10}$
- Bayes error rate (best we can do): 0% - no noise on the samples



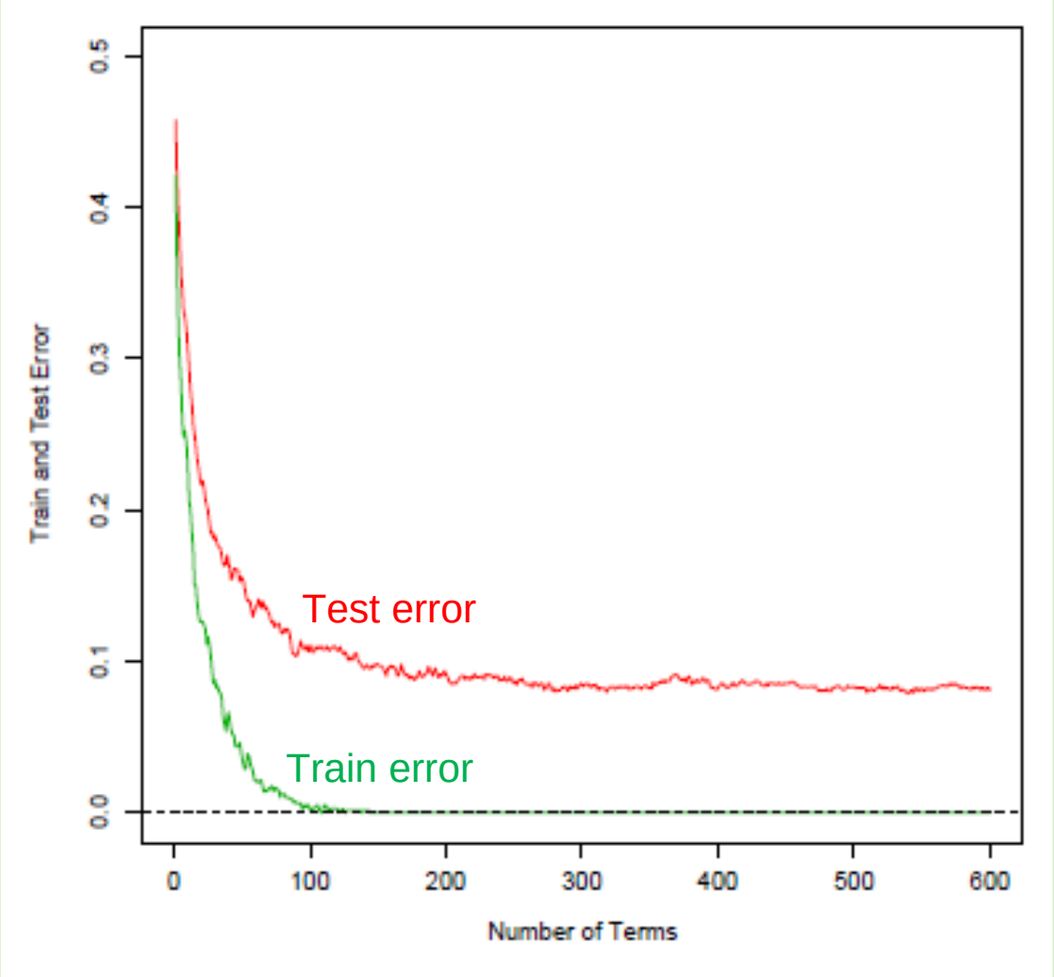
- In general:

$$Boosting \succ Random\ Forest \succ Bagging \succ single\ tree$$

https://www.slideshare.net/0xdata/gbm-27891077?from_action=save

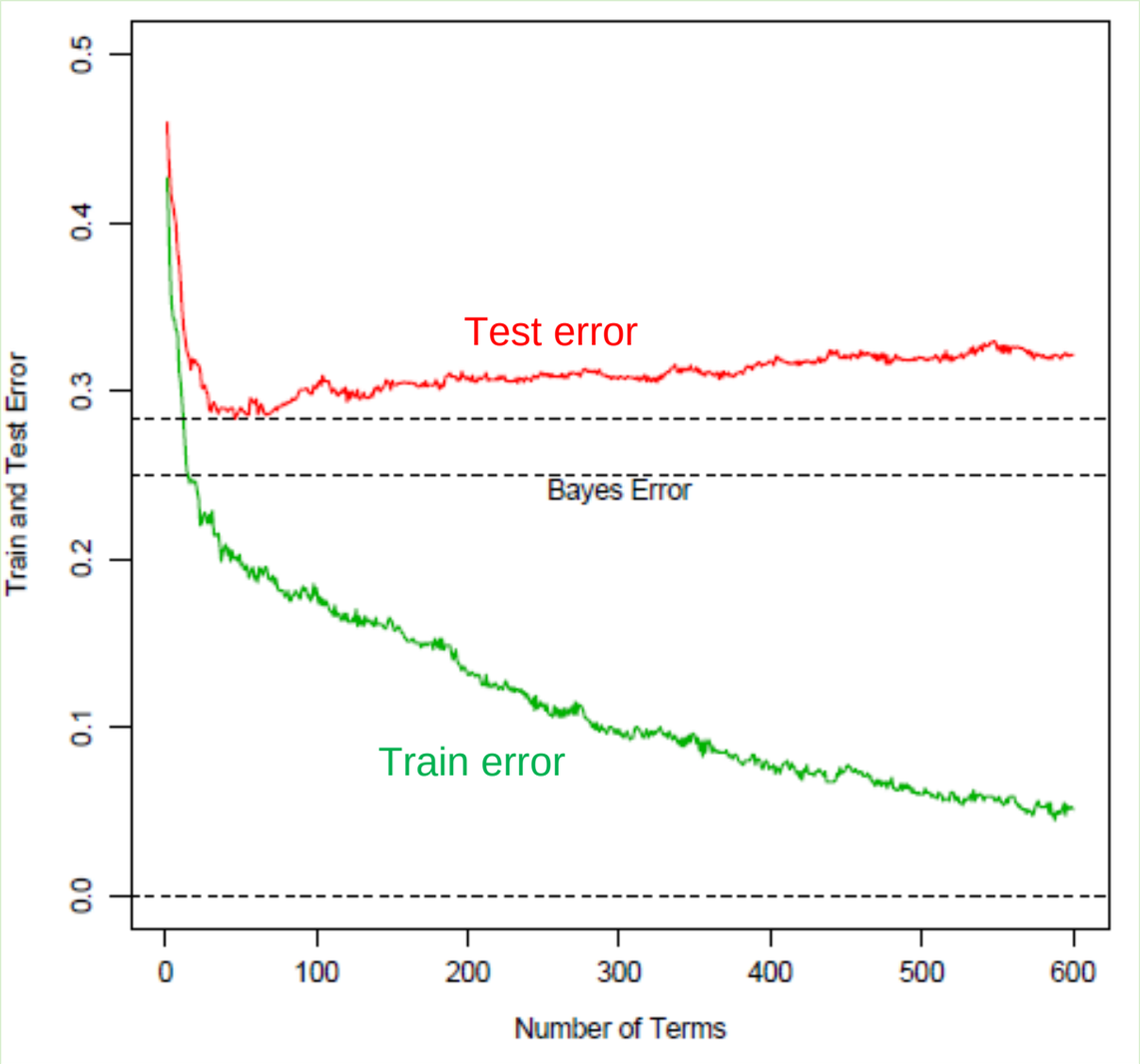
AdaBoost Magic!

- 2000 points from nested sphere in $\mathbb{R}^{10}$
- Bayes error rate (best we can do): 0%
  - no noise on the samples
- Our bounds predict that test error will increase with more rounds
- But in practice it usually doesn't
- There are margin based explanations for this, but we won't discuss them



AdaBoost on noisy samples

- Nested Gaussians in $\mathbb{R}^{10}$
- Bayes error rate (best we can do): 25% - high noise on the samples
- Error does increase in those situations, but quite slowly
- Can we adjust it?

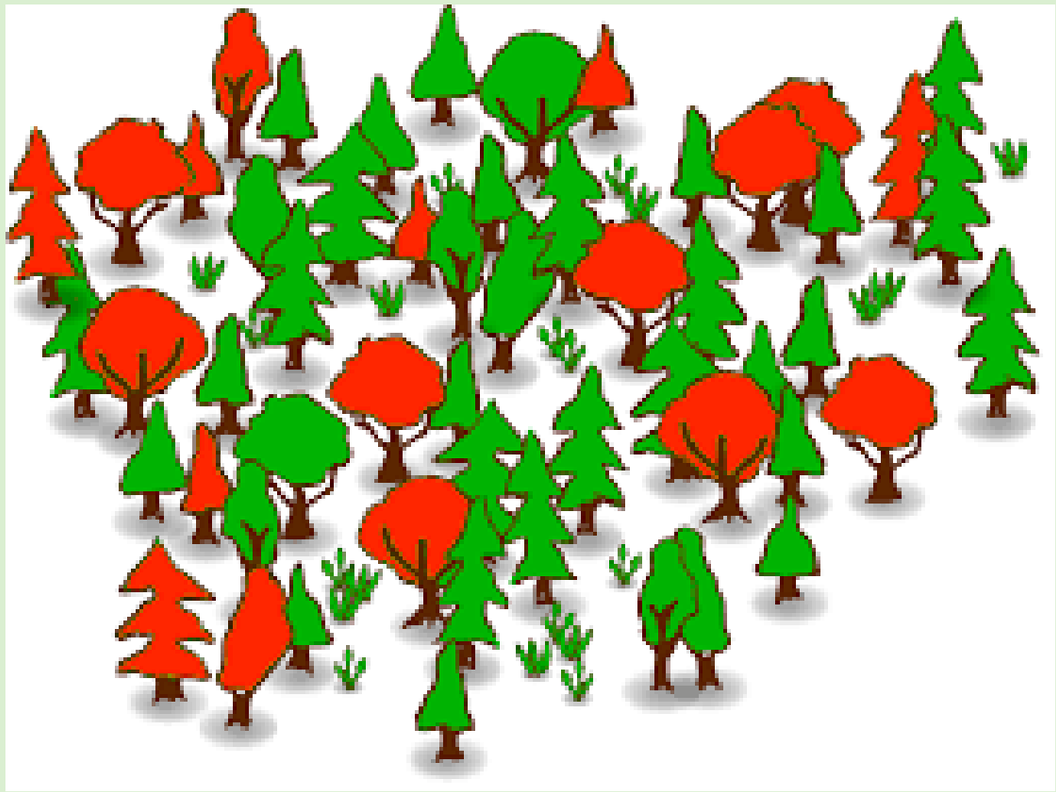


Adaboost Summary

- Boosting model
- Though generalizes for any weak learners, commonly used with DTs
- The example shown is also known as “Discrete adaboost” and is used for classification. For regression, Adaboost can be modified appropriately (see “Real Adaboost” by Friedman et al.)
- Suffers very little from overfitting when the Bayes error rate is close to 0

Today – DT Ensemble methods

- Bagging
- Random Forest
- Boosting
  - Adaboost
  - Gradient Boosting



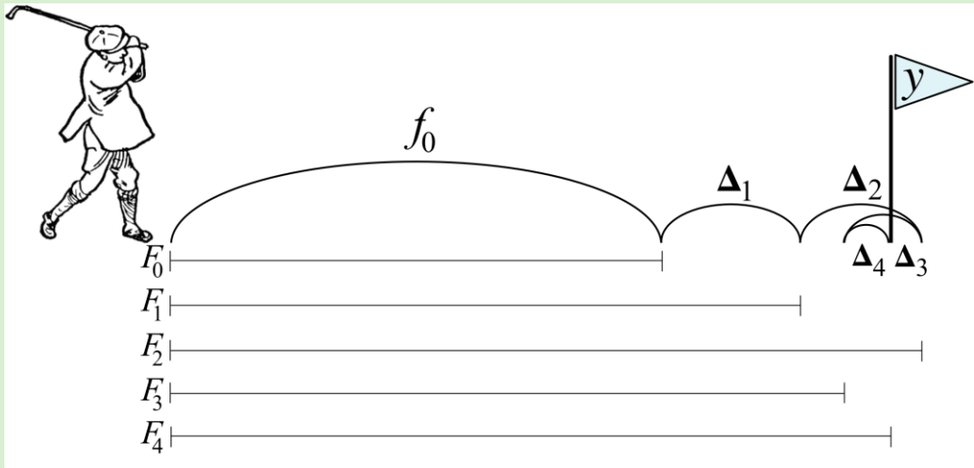
Forward Stage wise Additive modeling

- In general, when adding models (like stumps) sequentially, we would like to perform the following:

1. Initialize $\hat{f}^0(\mathbf{x}) = 0$
2. For $\mathrm{t} = 1$ to T:
    1. Compute $\alpha^t, h^t = \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1} \mathcal{L}(y_i, \hat{f}^{t-1}(\boldsymbol{x}_i) + \alpha \cdot h(\boldsymbol{x}_i))$
    2. Set $\hat{f}^t(\mathbf{x}) = \hat{f}^{t-1}(\mathbf{x}) + \alpha^t \cdot h^t(\mathbf{x})$

Where h^t is the recent added DT, $\hat{f}^{t-1}$ is the boosted model before h^t, and $\mathcal{L}$ is some loss function

- For some $\mathcal{L}$, the computation part can be simple!



Forward Stage wise Additive modeling - classification

- A possible loss function for classification is the exponential loss:

$$\mathcal{L}(y, \hat{y}) = e^{-y\hat{y}}$$

Where $y \in \{-1,1\}$

- Interestingly, it later turned out that:

Adaboost performs exactly that!

- Formal proof can be found at “Elements in Statistical Learning, Section 10.4”

- But, using the more general formulation, enables expanding easily to regression and conduct further adjustments

Forward Stage wise Additive modeling – squared loss

- Lets take $\mathcal{L}_{\mathrm{MSE}}(y, \hat{y}) = \frac{1}{2}(y - \hat{y})^2$

1. Initialize $\hat{f}^0(\mathbf{x}) = 0$
2. For $\mathrm{t} = 1$ to T:
   1. Compute $\alpha^t, h^t = \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1} \mathcal{L}(y_i, \hat{f}^{t-1}(\boldsymbol{x}_i) + \alpha \cdot h(\boldsymbol{x}_i))$

   $$= \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1} \frac{1}{2}\left(y_i - \left(\hat{f}^{t-1}(\boldsymbol{x}_i) + \alpha \cdot h(\boldsymbol{x}_i)\right)\right)^2 = \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1} \frac{1}{2}\left(r_i^{t-1} - \alpha \cdot h(\boldsymbol{x}_i)\right)^2$$

   2. Set $\hat{f}^t(\mathbf{x}) = \hat{f}^{t-1}(\mathbf{x}) + \alpha^t \cdot h^t(\mathbf{x})$

- The next tree only tries to fit the residuals (parts not covered so far)

Forward Stage wise Additive modeling And Gradient boosting

- In gradient descent, we take a step towards the negative gradient
- For squared loss, the negative gradient is:

$$-\frac{\partial \mathcal{L}\left(y_i, \hat{f}^{t-1}(\boldsymbol{x}_i)\right)}{\partial \hat{f}^{t-1}(\boldsymbol{x}_i)} = -\frac{\partial \frac{1}{2}\left(y_i - \hat{f}^{t-1}(\boldsymbol{x}_i)\right)^2}{\partial \hat{f}^{t-1}(\boldsymbol{x}_i)} = y_i - \hat{f}^{t-1}(\boldsymbol{x}_i) = r_i^{t-1}$$

- So we get:

$$\begin{aligned} residual \quad &\Leftrightarrow \quad negative\ gradient \\ fit\ h\ to\ residual \quad &\Leftrightarrow \quad fit\ h\ to\ negative\ gradient \\ update\ \hat{f}\ based\ on\ residual \quad &\Leftrightarrow \quad update\ \hat{f}\ based\ on\ negative\ gradient \end{aligned}$$

- Building a tree that minimizes the loss = taking a step towards the steepest gradient!

Shrinkage parameter

- So h fits to the negative gradient. What about α?

$$\hat{f}^t(\mathbf{x}) = \hat{f}^{t-1}(\mathbf{x}) + \alpha^t \cdot h^t(\mathbf{x})$$

- This is simply the step size!
  - Can be computed at each step by $\alpha^t = \underset{\alpha,h}{\operatorname{argmin}} \sum_{i=1} \mathcal{L}(y_i, \hat{f}^{t-1}(\boldsymbol{x}_i) + \alpha \cdot h^t(\boldsymbol{x}_i))$
  - To simplify computation, and prevent overfitting, Can be simply set as a constant between 0 and 1, or reduce it as function of iteration

We will use this approach from now on ☺

Gradient Boosting

- Initialize: $\hat{f}^0(x) = 0$, $\mathbf{r}^0 = \mathbf{y}$, and $D^0 = (\mathrm{X}, \mathbf{r})$
- for $t = 1 \ldots T$:
  - $\forall i.\ r_i^t \leftarrow -\left[\frac{\partial \mathcal{L}\left(y_i, \hat{f}^{t-1}(\mathbf{x}_i)\right)}{\partial \hat{f}^{t-1}(\mathbf{x}_i)}\right]$ Find the negative gradient
  - Set $D^t = (\mathrm{X}, \mathbf{r}^t)$ Set the next "labels" as the negative gradients
  - fit a tree h^t with J regions to D^t Fit a new DT
  - Update $\hat{f}^t(\mathbf{x})$ with a "learning rate" α: Add a step of size α
    $$\hat{f}^t(\mathbf{x}) \leftarrow \hat{f}^{t-1}(\mathbf{x}) + \alpha h^t(\mathbf{x})$$
- Return: $\hat{f}(x) = \hat{f}^T(\mathbf{x})$ Return the final hypothesis

Example – squared loss

- Initialize: $\hat{f}^0(x) = 0$, $\mathbf{r}^0 = \mathbf{y}$, and $D^0 = (\mathrm{X}, \mathbf{r}^0)$
- for $t = 1 \ldots T$:
  - $\forall i.\ r_i^t \leftarrow y_i - \hat{f}^{t-1}(\mathbf{x}_i)$ — Find the negative gradient
  - Set $D^t = (\mathrm{X}, \mathbf{r}^t)$ — Set the next "labels" as the negative gradients
  - fit a tree h^t with J regions to D^t — Fit a new DT
  - Update $\hat{f}^t(\mathbf{x})$ with a "learning rate" α: — Add a step of size α

$$\hat{f}^t(\mathbf{x}) \leftarrow \hat{f}^{t-1}(\mathbf{x}) + \alpha h^t(\mathbf{x})$$

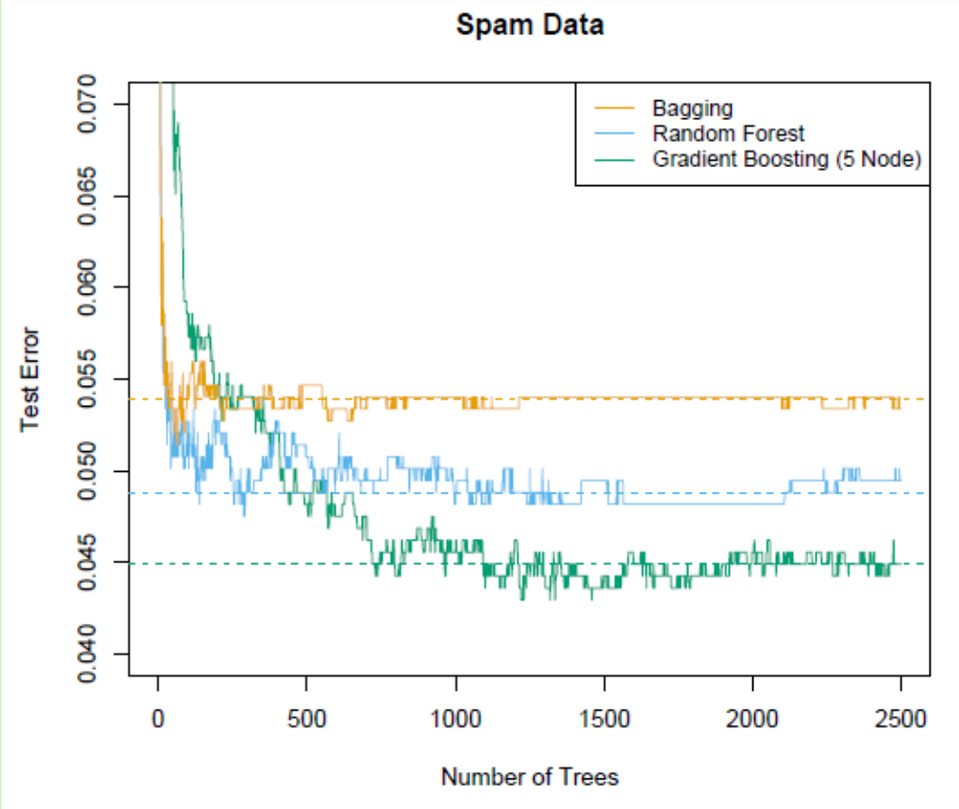
- Return: $\hat{f}(x) = \hat{f}^T(\mathbf{x})$ — Return the final hypothesis

Gradient boosting hyper parameters

- Number of trees T:
  - Unlike bagging and RF, boosting can overfit if T is too large, although this overfitting tends to occur slowly if at all.
  - Use cross validation to determine it
  - Usually 100-10000
- The shrinkage parameter α
  - A small positive number which controls the rate at which the boosting learns
  - Typical values are $0.0001 - 1$.
  - Small α requires big T
- The tree depth d in each tree:
  - Often $d = 1$ works well (stumps)
  - Can be enlarged to 2 or 3 or even up to 10

DT methods comparison

- 3000 training samples and 1500 test samples
- Target is spam/not spam



- In general:

$$Boosting \succ Random\ Forest \succ Bagging \succ single\ tree$$

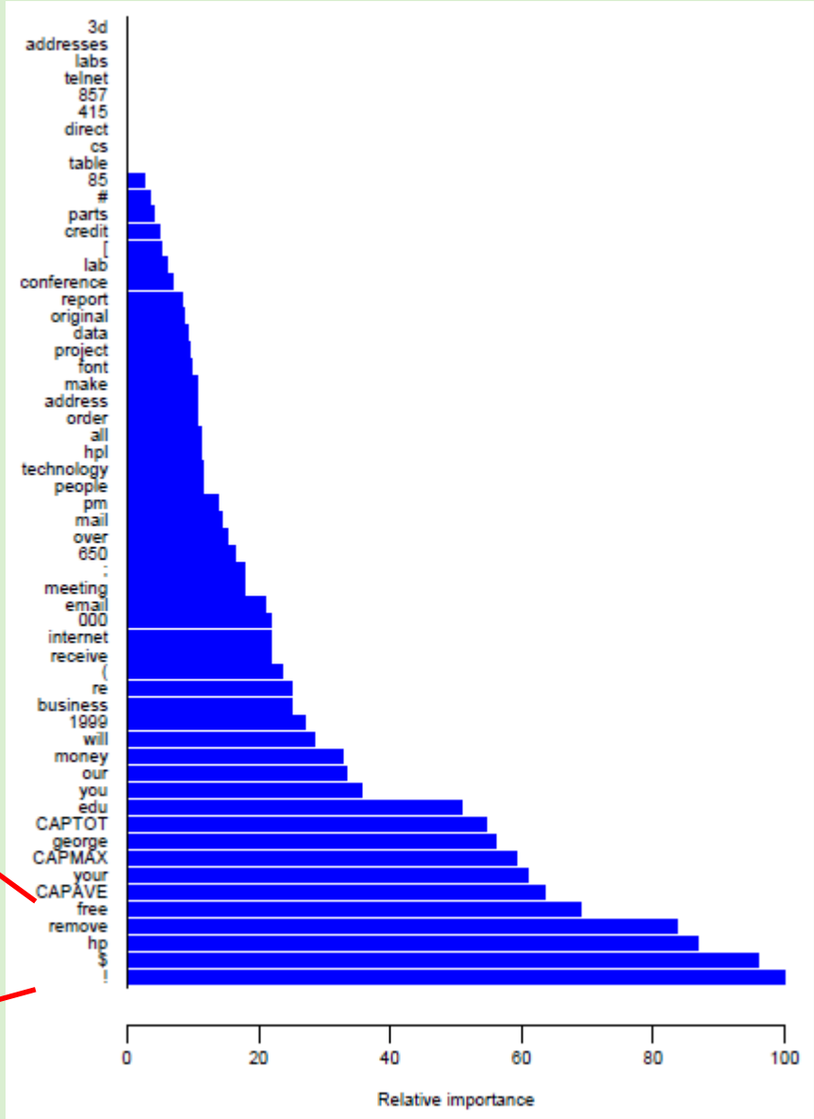
- Random forests are simpler to train and tune than boosting

https://www.slideshare.net/0xdata/gbm-27891077?from_action=save

Gradient Boosting methods – feature importance on spam detectors

- Feature importance

- Free
- Remove
- Hp
- $
- !



https://www.slideshare.net/0xdata/gbm-27891077?from_action=save

Boosting - Summary

- Instead of evenly aggregating random classifiers, do it sequentially, using weak classifiers
- Many different variants of weak classifiers
- One of the top methods!
- Many awesome expansions for it (see XGBoost & CatBoost)

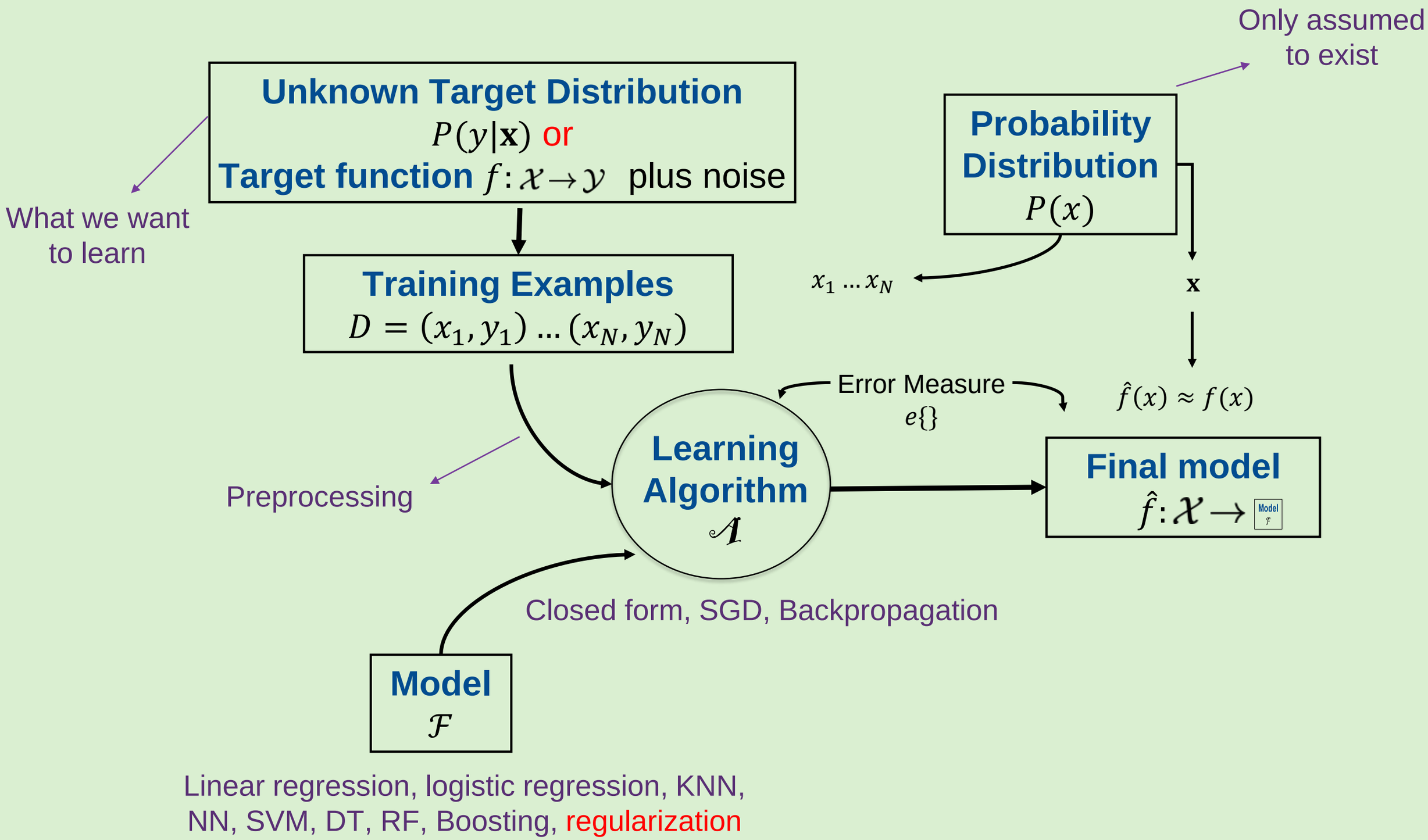
Trees Notebook

Questions?

Summary – so far

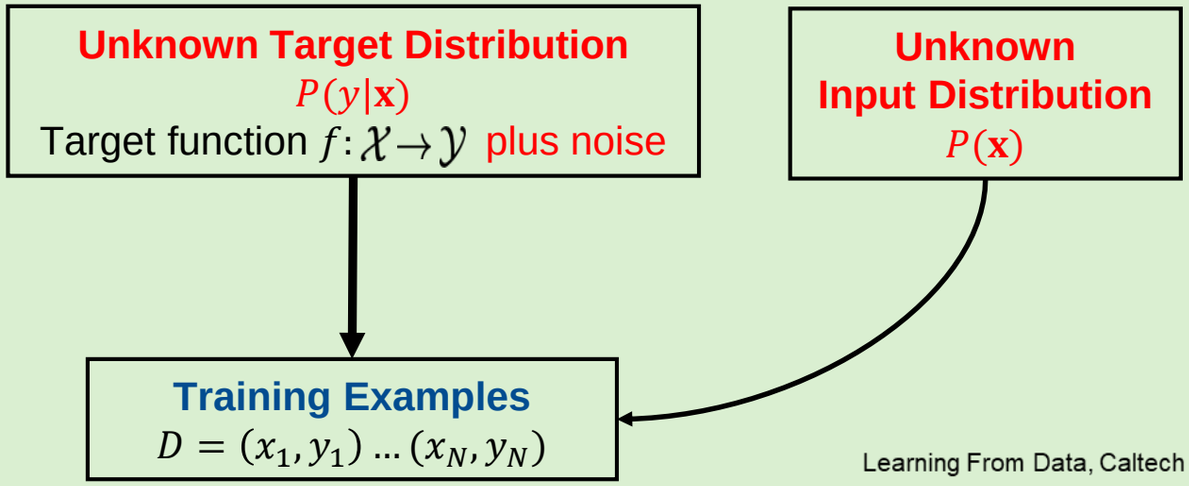
- We started by defining what it means to learn
  - Bias variance tradeoff: $E_{\text{test}} = variance + bias^2 + irredicible$
- Discussed the learning diagram and its components
- Practical part: Dimensionality reduction
- Saw different models with there learning algorithms

Summary – so far



Using probability approximations

- So far, tried to approximate the unknown function f by a model $\hat{f}$
- Even when using $\hat{P}(y|\mathrm{x})$, we modeled it through $\hat{f}(x)$
- What if we had known $P(\mathbf{x})$? How would we use it?
- Would the prior $P(\mathbf{y})$ also help us? Would $P(\mathbf{x}|y)$?
- How can we approximate those probabilities?



Next steps

- We will first introduce Bayes (optimal) classifier — Lecture 10
- Discuss different approximations for it
  - Each limiting assumption would lead to a different result
- Detach from the supervised:
  - Learn ways to assess $P(\mathbf{x})$ — Lecture 11
  - Clustering – the most notorious unsupervised problem (together with dimensionality reduction) — Lecture 12