Introduction to Machine Learning (CSCI-UA 473): Fall 2021

Lecture 11: Bagging, Decision Trees, and Boosting (Ensemble Learning)

Sumit Chopra
Courant Institute of Mathematical Sciences
Department of Radiology - Grossman School of Medicine
NYU

Slides derived from materials from David Sontag, Tuo Zhao, and Alexander Chouldechova

Lecture Outline

- Ensemble Learning
  - Bagging
  - Random Forests
  - Boosting
    - Adaboost
    - Gradient Boosted Decision Trees (if time permits)

Decision Tree Takeaways

Decision trees are quite popular in the machine learning community, especially in practice

Easy to understand, implement and use

Computationally they are cheap

While they are human readable they may not necessarily use human logic

What variable to select on is chosen based on information gain

They can also be used for regression purposes: typically take the average of examples in the bin

Decision trees will overfit!

Need to use tricks/heuristics to avoid overfitting (limit depth, threshold number of points in a leaf bin etc)

Decision trees have a very high variance

A small change in the data will result in a completely different decision tree

Recap of Bias-Variance Tradeoff

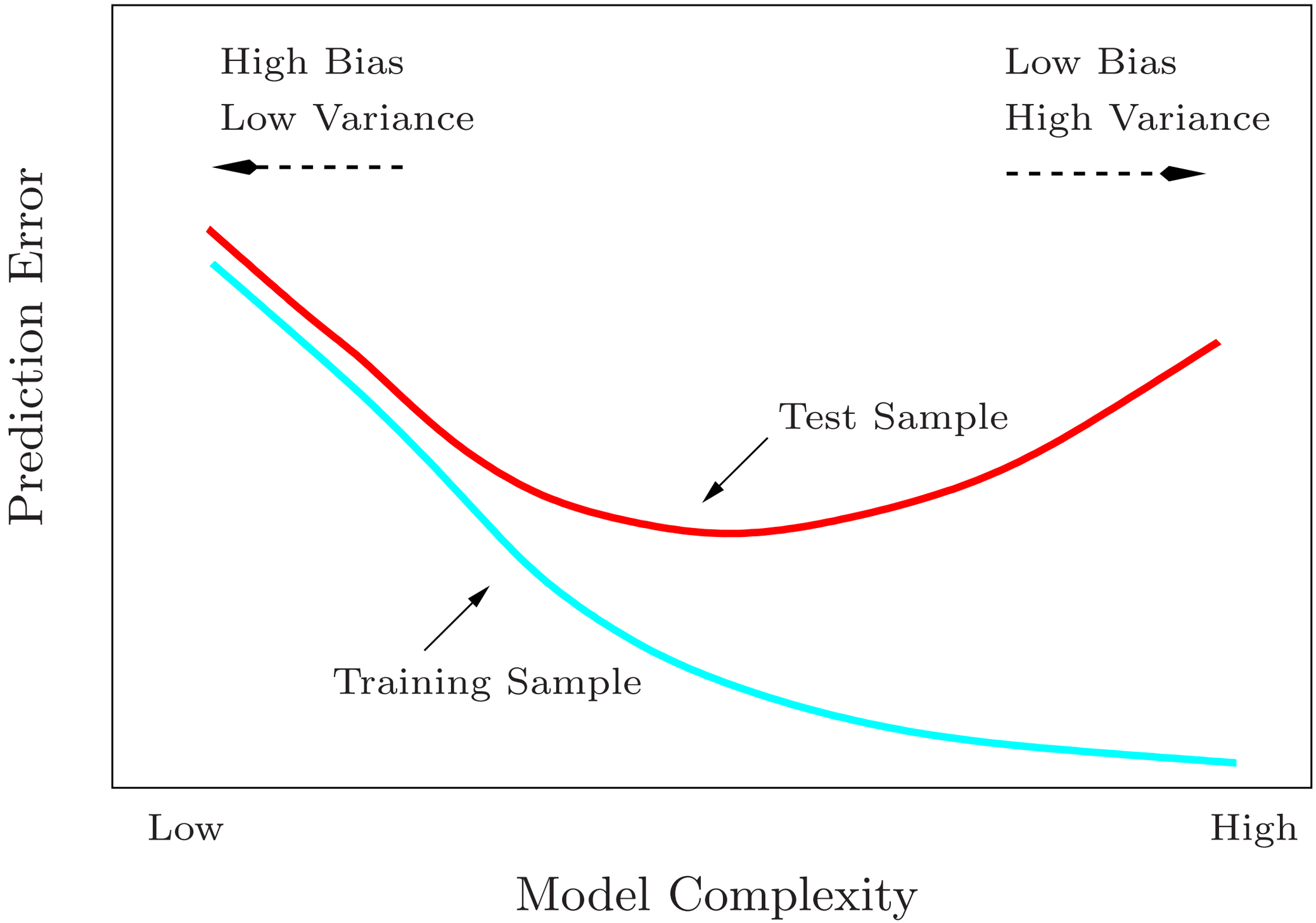


Figure taken from: Elements of Statistical Learning

Is there a way to reduce Variance without increasing Bias?

The answer is Yes!

Averaging reduces the variance, provided the model predictions are independent

If we have N models then $Var(\bar{X}) = \frac{Var(X)}{N}$

Consider the comparison between the two processes: Validation and Cross-validation

While a single validation set gives an estimate of the out-of-sample error its estimates are optimistic and variable

K-fold Cross Validation fixes it by giving a more stable error estimate

It essentially averages the errors over multiple independent folds

The Bagging (**B**ootstrap **Agg**regat**ing**) has a similar motivation

What is Bootstrap Sampling?

Averaging and all makes sense, but what do we average on? We only have a single training set at our disposal

Bootstrap sampling comes to the rescue here (an old idea in Statistics)

Create a collection of training sets of size n, each obtained by sampling examples with replacement from the given dataset

Essentially we are using empirical distribution of our data to estimate the true unknown data-generating distribution

Note that the sampling happens with replacement.

This means that there is a chance that we'll pick some examples multiple times

And not all training examples will appear in each sample

Bootstrap Sampling Continued

If we sample uniformly then we can compute the probability that a given example from a dataset of size n does not get chosen as a member of the bootstrap sample

$$P(not\ chosen) = \left(1 - \frac{1}{n}\right)^n$$

Which is asymptotically equivalent to

$$\frac{1}{e} \approx 0.368 \text{ as } n \to \infty$$

$$\text{Thus } P(chosen) = 1 - \left(1 - \frac{1}{n}\right)^n \approx 0.632$$

In other words, each Bootstrap sample has around $63.2\,\%$ off the observed data examples

What happens to the examples that randomly get left out?

Bagging (**B**ootstrap **Agg**regat**ing**)

Assume we are working with the classification tree

Given a training set $\mathscr{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$

Bagging averages the prediction from classification trees over a collection of bootstrap samples

For $b = 1,\ldots B$, get a bootstrap sample of size n from the training data:

$$(x_i^{*b}, y_i^{*b}) \forall i = 1,\ldots, n$$

Fit the classification tree g_b^{tree} on each sample b

A new sample x_0 is classified by taking a plurality vote over the predictions of B trees

$$\hat{y}_{x_0}^{bagging} = \arg \max_{k=1,\ldots,K} \sum_{b=1}^{B} I\left(g_b^{tree}(x_0) = k\right)$$

Typically when we “Bag” trees we are less worried about accuracy of individual trees

Bagging (**B**ootstrap **Agg**regat**ing**)

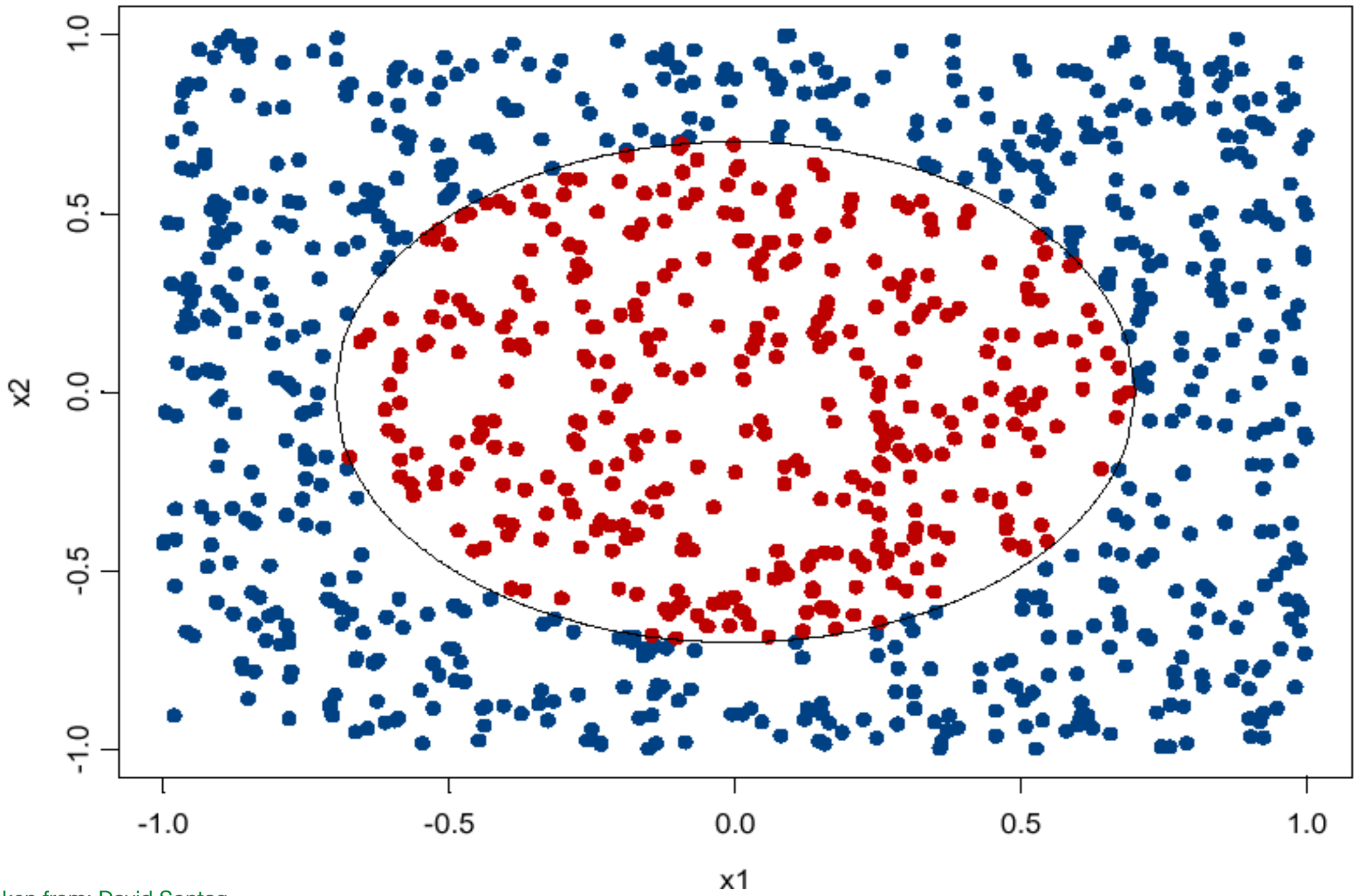


Figure taken from: David Sontag

Bagging (**B**ootstrap **Agg**regat**ing**)

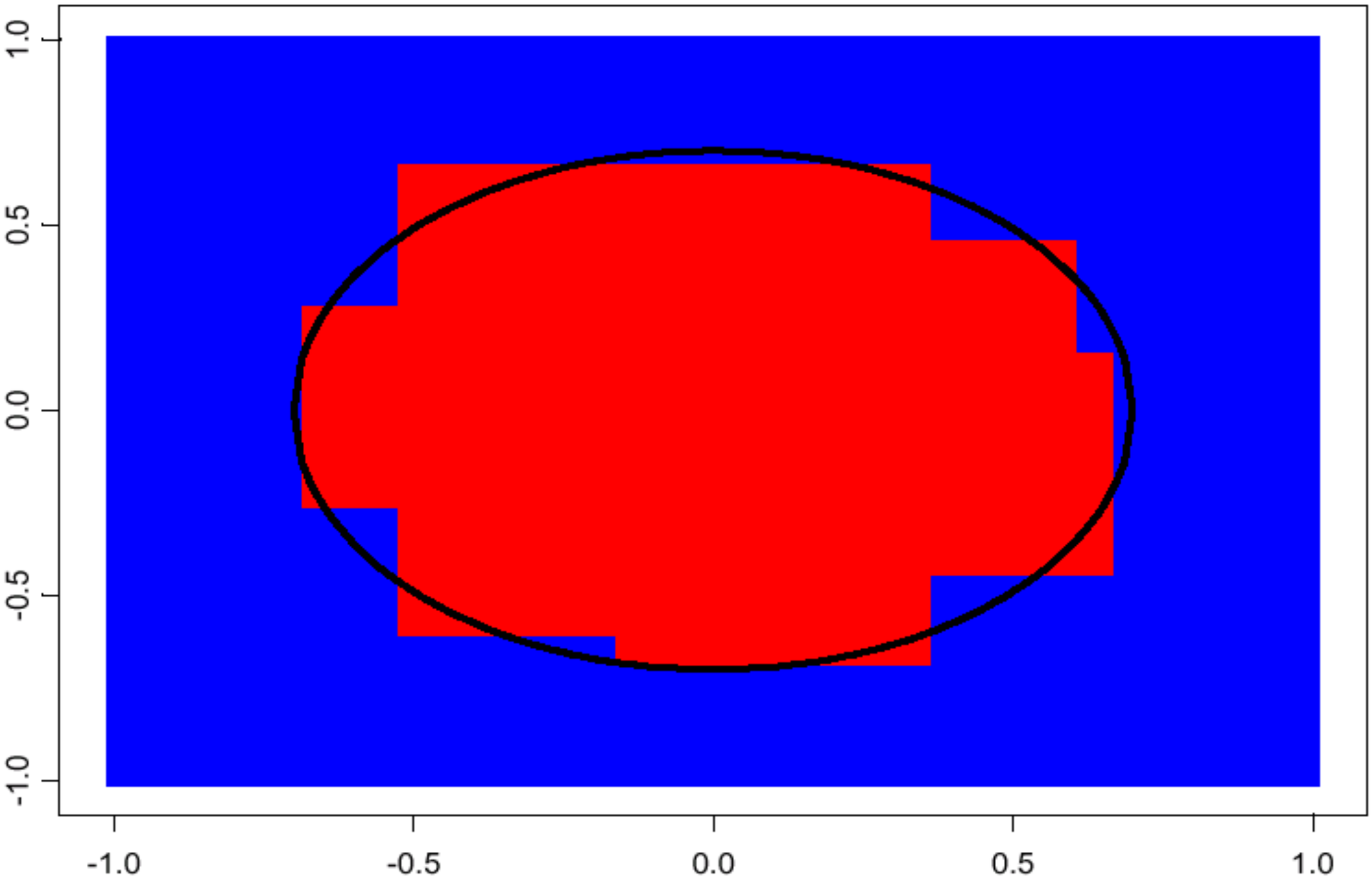


Figure taken from: David Sontag

Bagging (**B**ootstrap **Agg**regat**ing**)

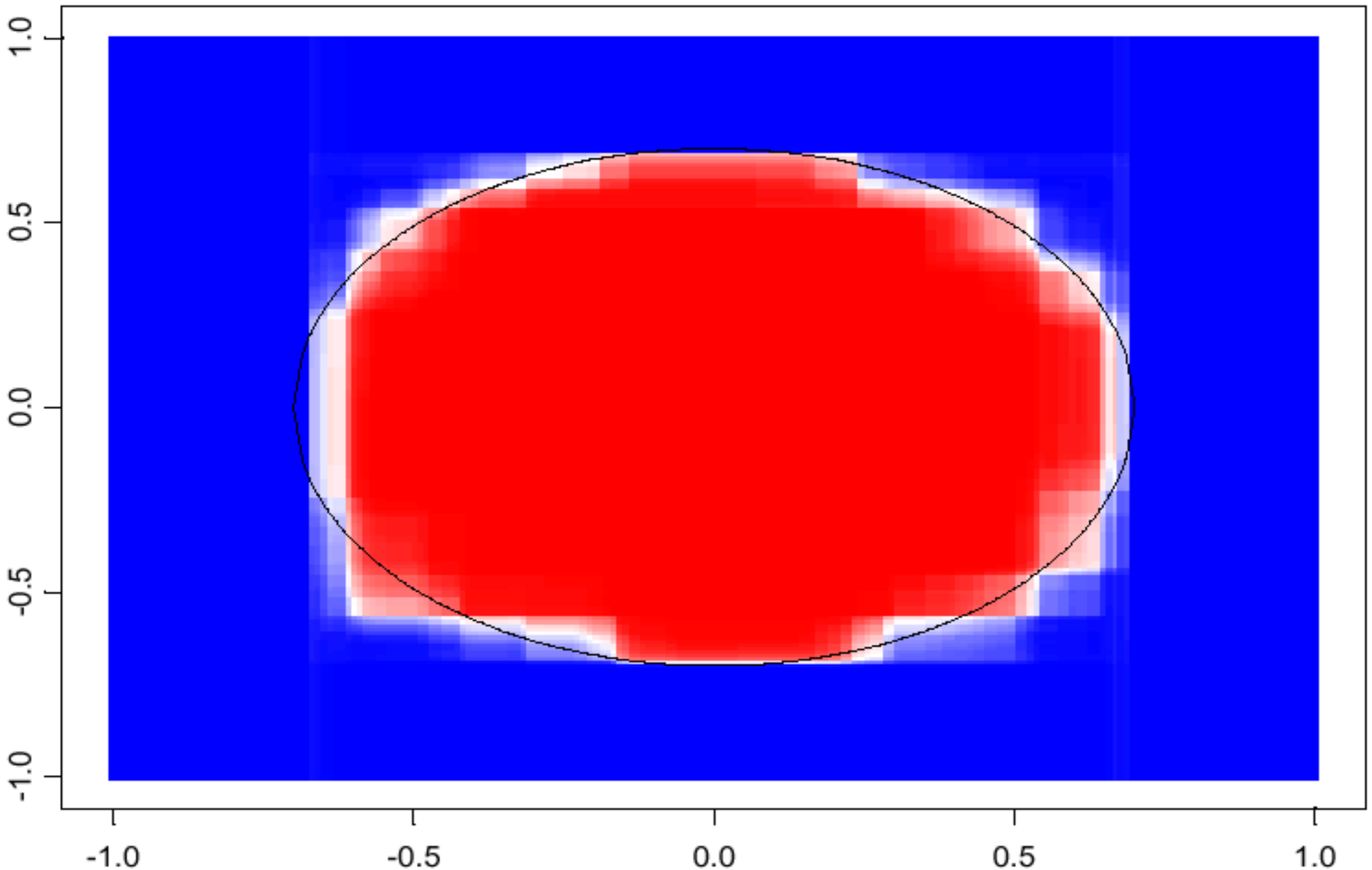


Figure taken from: Elements of Statistical Learning

Bagging (**B**ootstrap **Agg**regat**ing**): Another way of combining outcomes

What if we want probability estimates for each class instead of the class with the larges number of votes?

Instead of combining like

$$\hat{y}_{x_0}^{bagging} = \arg \max_{k=1,\ldots,K} \sum_{b=1}^{B} I\left(g_b^{tree}(x_0) = k\right)$$

We combine the probabilities directly

$$\hat{p}_k^{bagging}(x_0) = \frac{1}{B} \sum_{i=1}^{B} \hat{p}_k^i(x_0) \quad for\, k = 1,\ldots,K$$

Where $\hat{p}_k^i(x_0)$ is the predicted probability of the k-th class by the tree fitted on sample i

Then the final decision is provided by

$$\hat{Y}_{x_0}^{bagging} = \arg \max_{k=1,\ldots,K} \hat{p}_k^{bagging}(x_0)$$

This is generally a preferred form of bagging. More well behaved.

Bagging (**B**ootstrap **Agg**regat**ing**): Another way of combining outcomes

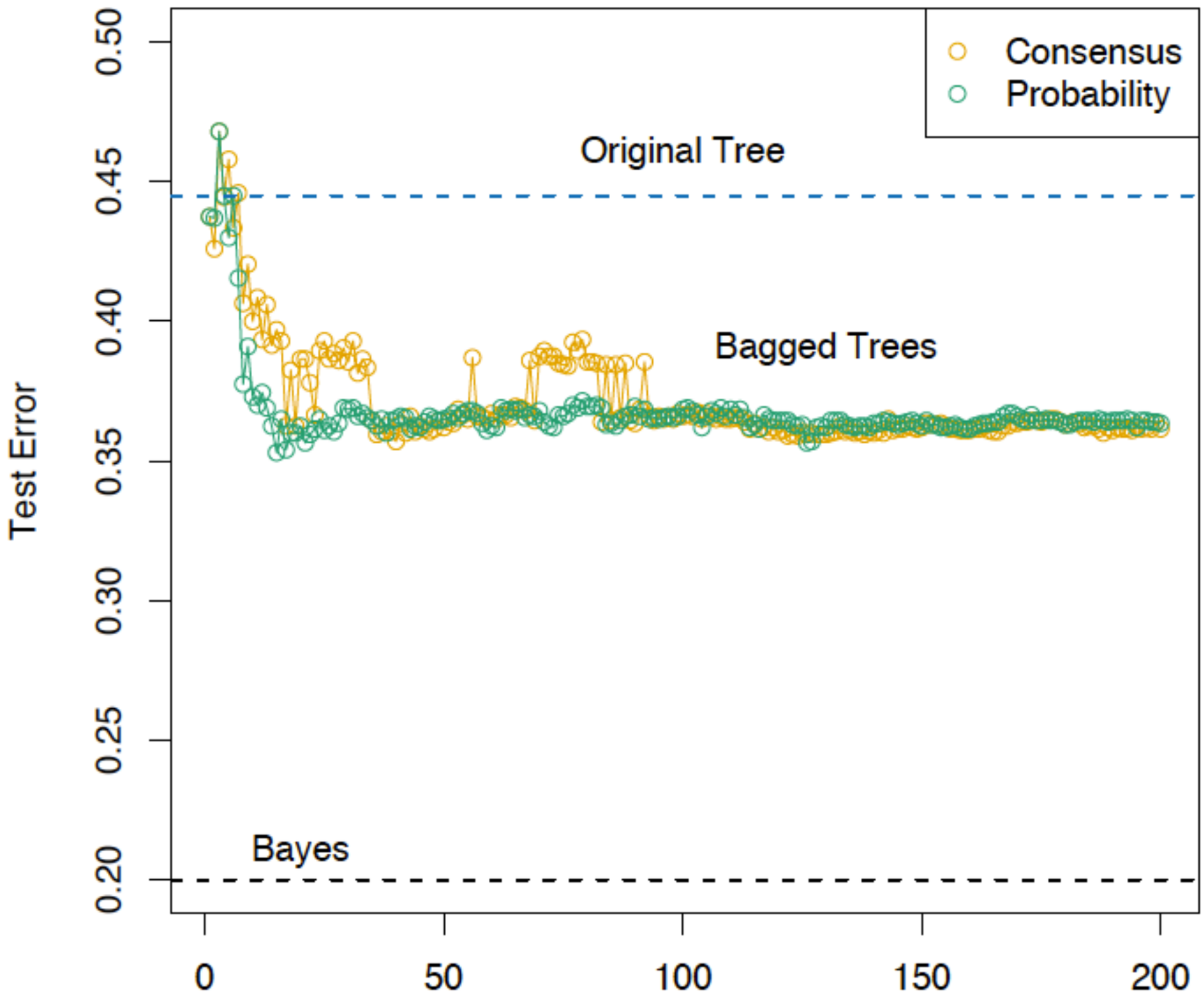


Figure taken from: Alexander Chouldechova

Test error stops improving after we hit a certain threshold

Out-of-Bag (OOB) Error Estimation

Recall that each bootstrap sample contains roughly only 62.3% (approx. 2/3) of the training samples

The remaining (approx. 1/3) samples are not chosen for training and can be see as test examples and are called out-of-bag (OOB) examples

In other words, each example is an OOB for roughly B/3 trees

To form the OOB estimate of test error

Predict the response for the i-th observation using each of the trees for which the i-th sample was OOB. From above we will get roughly B/3 predictions for each observation

Calculate the error of each OOB prediction

Average all of the errors

Random Forests

Random forests provide an improvement over bagged trees by providing a small tweak that decorrelates the individual trees

This further reduces the variance

It introduces two sources of randomness: Bagging and Random feature selection at every step

Build the tree in the same way as before

However while building the tree at each node, select a random subset of m predictors and split along the best predictor among that subset

A fresh selection of m random predictors are selected for each node of the tree

A good choice is $m \approx \sqrt{P}$

Random Forests

1. For $b = 1$ to B:

   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.

   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.

      i. Select m variables at random from the p variables.

      ii. Pick the best variable/split-point among the m.

      iii. Split the node into two daughter nodes.

2. Output the ensemble of trees $\{T_b\}_1^B$.

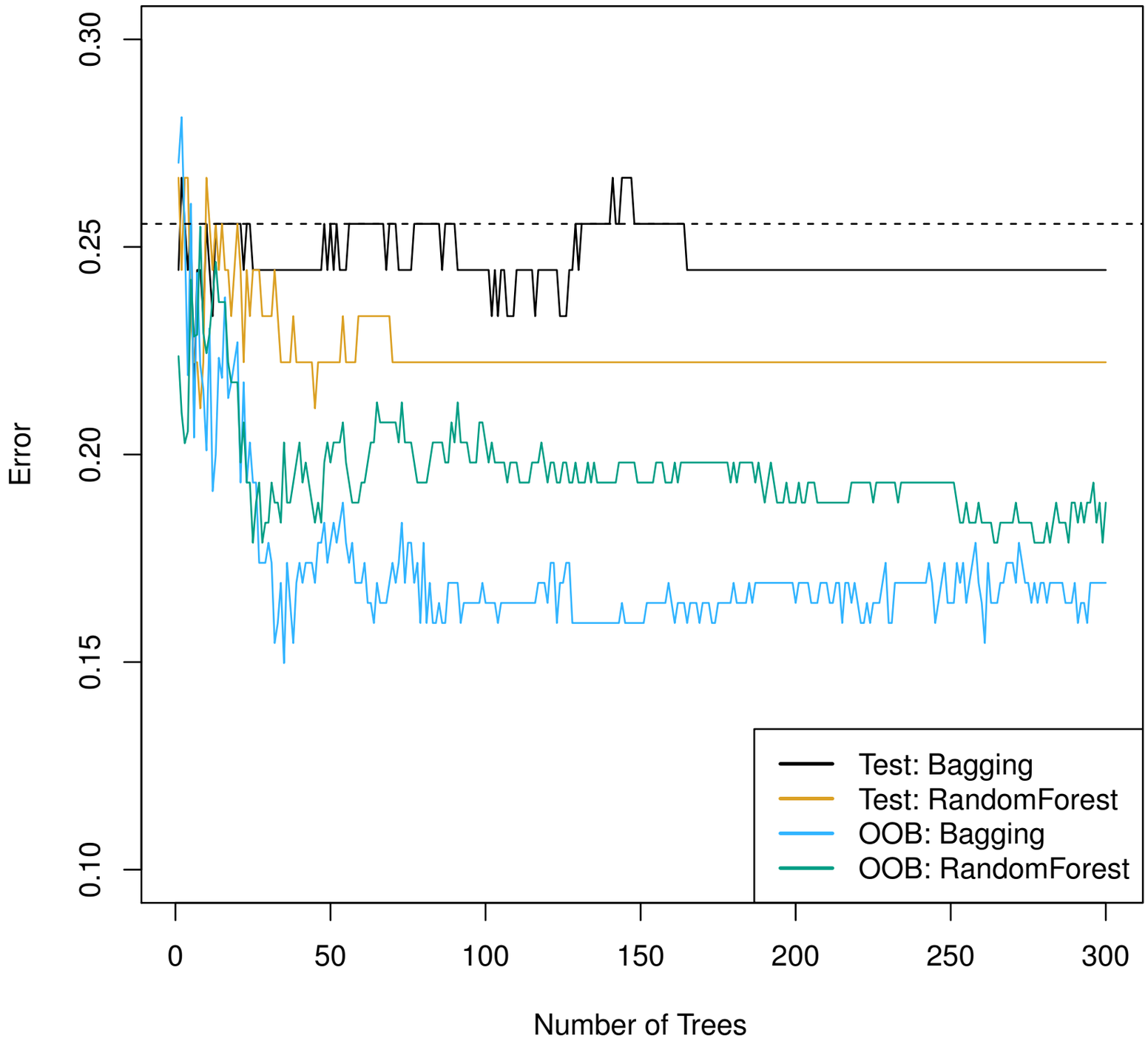
To make a prediction at a new point x:

Regression: $\hat{f}_{\text{rf}}^B(x) = \frac{1}{B}\sum_{b=1}^{B} T_b(x)$.

Classification: Let $\hat{C}_b(x)$ be the class prediction of the bth random-forest tree. Then $\hat{C}_{\text{rf}}^B(x) = \textit{majority vote}\ \{\hat{C}_b(x)\}_1^B$.

Random Forest Algorithm

Random forests are better predictors than bagged trees



Boosting

Weak learners: a classifier which has an accuracy of little more than 50%

Examples: a decision tree stump or a linear classifier

Boosting is an iterative process

The idea is to re-weight the training set based on the performance of the weak learner

Combine the recently learnt weak learner into the ensemble of previously trained learners

Repeat

How the training sets are weighted and how in the end the weak learners are combined depends on the algorithm

Boosting Algorithm Overview

Initialize a weight vector with uniform weights: one weight per training sample

Loop (until convergence):

Apply the weak learner to the weighted training set: instead of working with original training set, draw a Bootstrap sample with the weighted probabilities

Increase the weights of misclassified examples

To make a prediction on a new sample, do a (weighted) majority voting on trained classifiers

Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: Journal of computer and system sciences 55.1 (1997), pp. 119–139.

AdaBoost

Adaptive Boosting

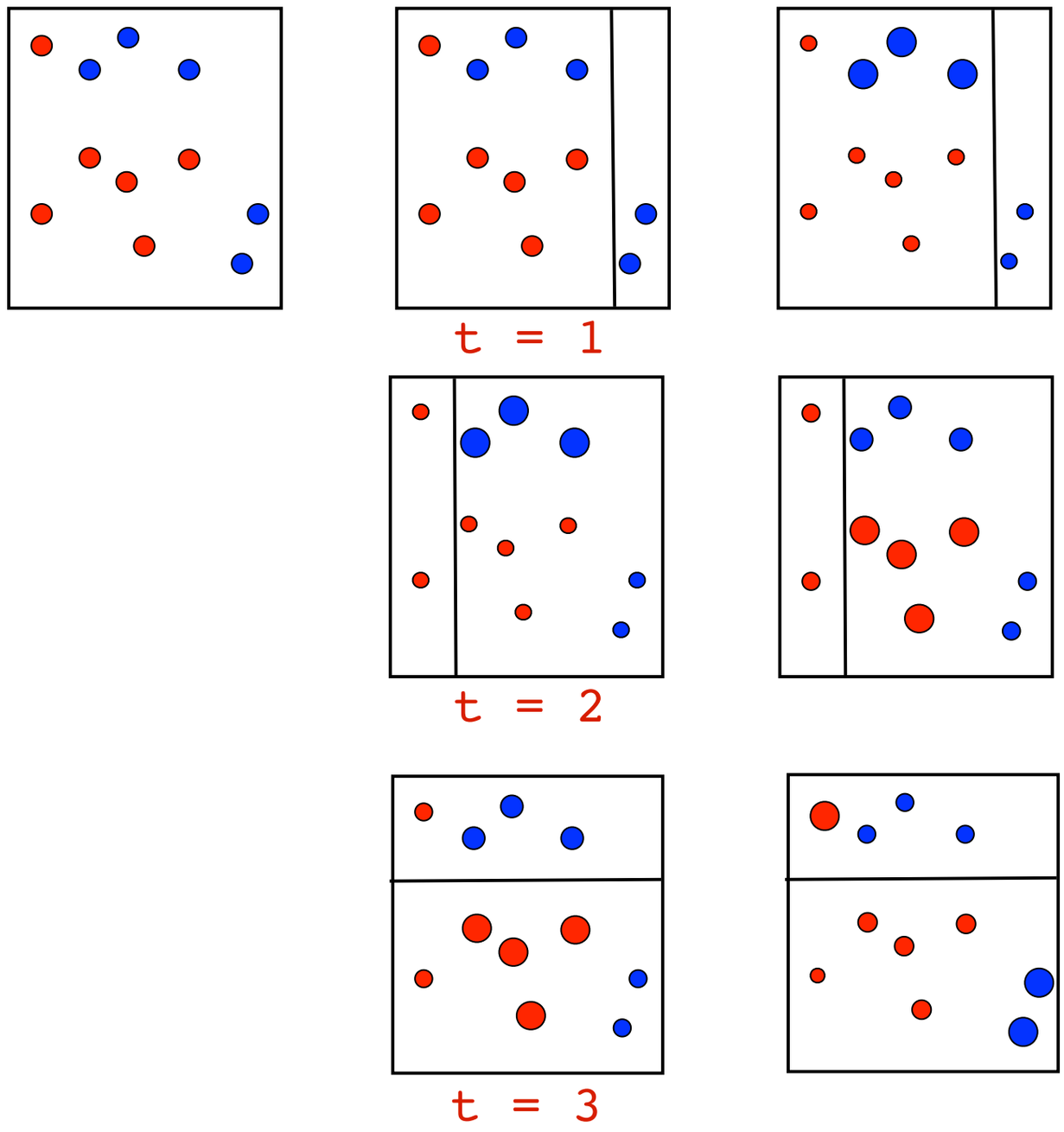
Algorithm 1 AdaBoost

Initialize k: the number of AdaBoost rounds
Initialize $\mathcal{D}$: the training dataset, $\mathcal{D} = \{\langle \mathbf{x}^{[1]}, y^{[1]} \rangle, ..., \mathbf{x}^{[n]}, y^{[n]} \rangle\}$
Initialize $w_1(i) = 1/n, \quad i = 1, ..., n, \ \mathbf{w}_1 \in \mathbb{R}^n$

for r=1 to k **do**
    For all $i : \mathbf{w}_r(i) := w_r(i) / \sum_i w_r(i)$ [normalize weights]
    $h_r := FitWeakLearner(\mathcal{D}, \mathbf{w}_r)$
    $\epsilon_r := \sum_i w_r(i) \ \mathbf{1}(h_r(i) \neq y_i)$ [compute error]
    if $\epsilon_r > 1/2$ then stop
    $\alpha_r := \frac{1}{2} \log[(1 - \epsilon_r)/\epsilon_r]$ [small if error is large and vice versa]
    $$w_{r+1}(i) := w_r(i) \times \begin{cases} e^{-\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) = y^{[i]} \\ e^{\alpha_r} & \text{if } h_r(\mathbf{x}^{[i]}) \neq y^{[i]} \end{cases}$$
Predict: $h_k(\mathbf{x}) = \arg\max_j \sum_r \alpha_r \mathbf{1}[h_r(\mathbf{x}) = j]$

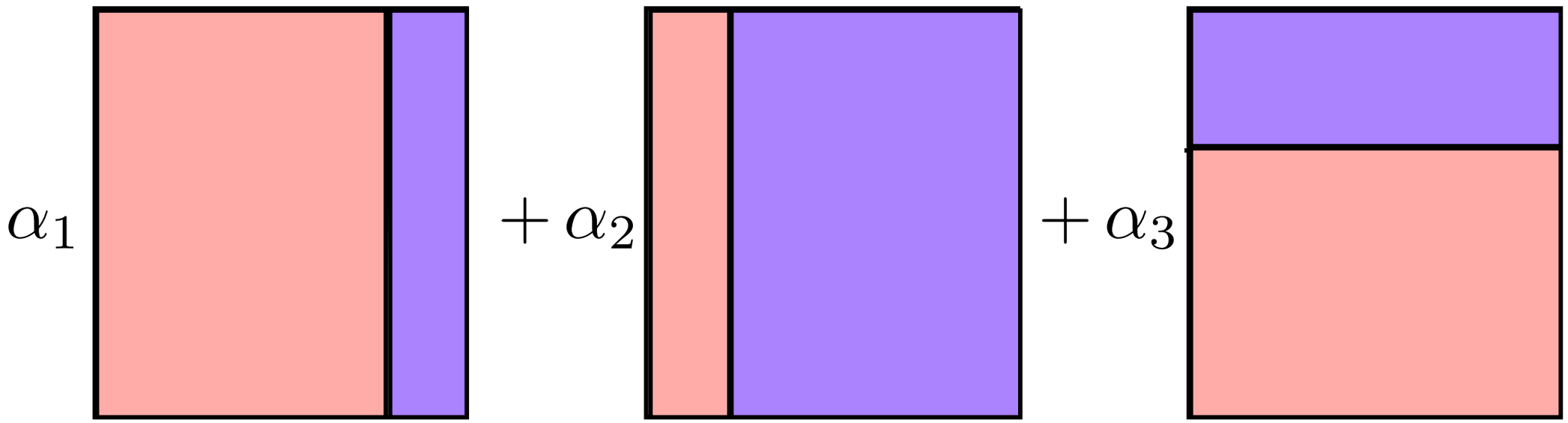
Yoav Freund and Robert E Schapire. “A decision-theoretic generalization of on-line learning and an application to boosting”. In: Journal of computer and system sciences 55.1 (1997), pp. 119–139.

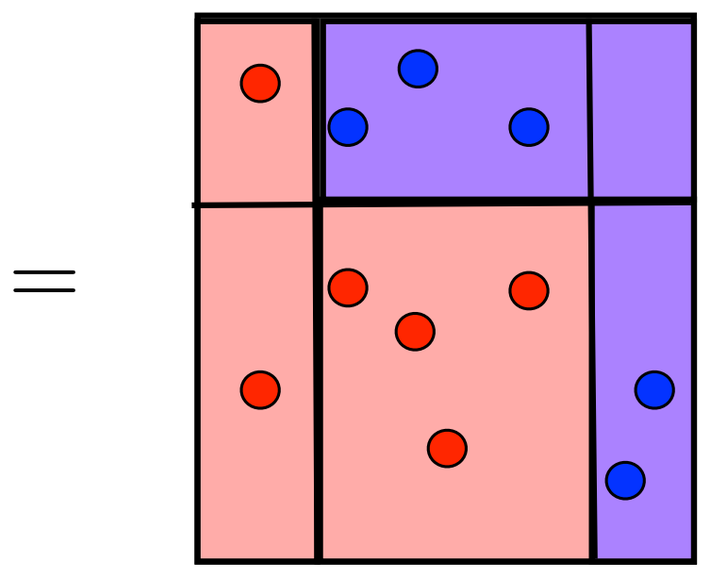
Illustration of Boosting



Slide taken from Mehryar Mohri

Illustration of Boosting





Slide taken from Mehryar Mohri

Idea Behind Gradient Boosting

You are given the training set $\mathscr{D} = \{(x_1, y_1), \ldots, (x_n, y_n)\}$ and your goal is come up with the best model that can minimize a square loss over $\mathscr{D}$

Suppose I give you a model F that fits well on the data but it is not perfect. That is it makes mistakes

Rules of the game:

1. You are not allowed to delete anything from F or change any of its parameters
2. You can only add an additional model to F so that the new model $F_{new}(x_i) = F(x_i) + h(x_i)$

How will you go about it?

Idea Behind Gradient Boosting

In other words you wish to improve the model such that

$$F(x_1) + h(x_1) = y_1$$
$$F(x_2) + h(x_2) = y_2$$
$$\vdots$$
$$F(x_n) + h(x_n) = y_n$$

Or equivalently you want

$$h(x_1) = y_1 - F(x_1)$$
$$h(x_2) = y_2 - F(x_2)$$
$$\vdots$$
$$h(x_n) = y_n - F(x_n)$$

Can a regression tree solve this?

One can certainly approximate this solution though

Idea Behind Gradient Boosting

Fit a new regression tree on a new training set given by

$$\mathscr{D}' = \{(x_1, [y_1 - F(x_1)]), (x_2, [y_2 - F(x_2)]), \ldots, (x_n, [y_n - F(x_n])\}$$

Idea Behind Gradient Boosting

Residuals of the model

Fit a new regression tree on a new training set given by

$$\mathscr{D}' = \{(x_1, [y_1 - F(x_1)]), (x_2, [y_2 - F(x_2)]), \ldots, (x_n, [y_n - F(x_n])\}$$

Residuals are the parts that existing model F cannot do well on

The role of h now is to compensate for the shortcomings of the model F

So the new model will be $F + h$

If the results are still not satisfactory then we can add repeat the procedure to add another model

We can keep repeating till we are satisfied with the fit

Idea Behind Gradient Boosting

How is this related to gradients?

Note that our loss function is a squared loss: $L(y_i, F(x_i)) = (y_i - F(x_i))^2$

And we want to minimize $\mathscr{L} = \sum_{i=1}^{n} L(y_i, F(x_i))$ by adjusting the values of $F(x_1), F(x_2), \ldots, F(x_n)$

These are just numbers which can be viewed as parameters. Hence we can take the gradient of the loss with respect to them:

$$\frac{\partial \mathscr{L}}{\partial F(x_i)} = \frac{\partial \sum_{i=j}^{n} L(y_j, F(x_j))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = -F(x_i) - y_i$$

Thus the residual can be interpreted as the negative of the gradient

Idea Behind Gradient Boosting

Thus the update rule is

$$
\begin{aligned}
F_{t+1}(x_i) &= F_t(x_i) + h(x_i) \\
&= F_t(x_i) + [y_i - F(x_i)] \\
&= F_t(x_i) - \frac{\partial \mathscr{L}}{\partial F(x_i)}
\end{aligned}
$$

$$
\theta_i = \theta_i - \eta \frac{\partial \mathscr{L}}{\partial \theta_i}
$$

Idea Behind Gradient Boosting

Summary of the algorithm

Define the negative gradient as

$$-g(x_i) = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = y_i - F(x_i)$$

Start with an initial model, say $F_1(x) = \frac{\sum_i y_i}{n}$

Iterate until convergence:

Calculate the negative gradients $-g(x_i)$

Fit a regression tree h to the negative gradients $-g(x_i)$

Update the mode: $F_{t+1} = F_t + \eta h$, where $\eta = 1$

The benefit of formulating this in terms of gradients is that it allows us to play with other loss functions and derive corresponding algorithms in the same way

End of Lecture 11