MY474: Applied Machine Learning for Social Science

Lecture 8: Ensemble Methods, Bagging, Random Forests, Boosting

14 March 2022

Weeks 9 - 11

1. **Bagging, random forests, and boosting**
2. Unsupervised learning
3. Neural networks

Before we start

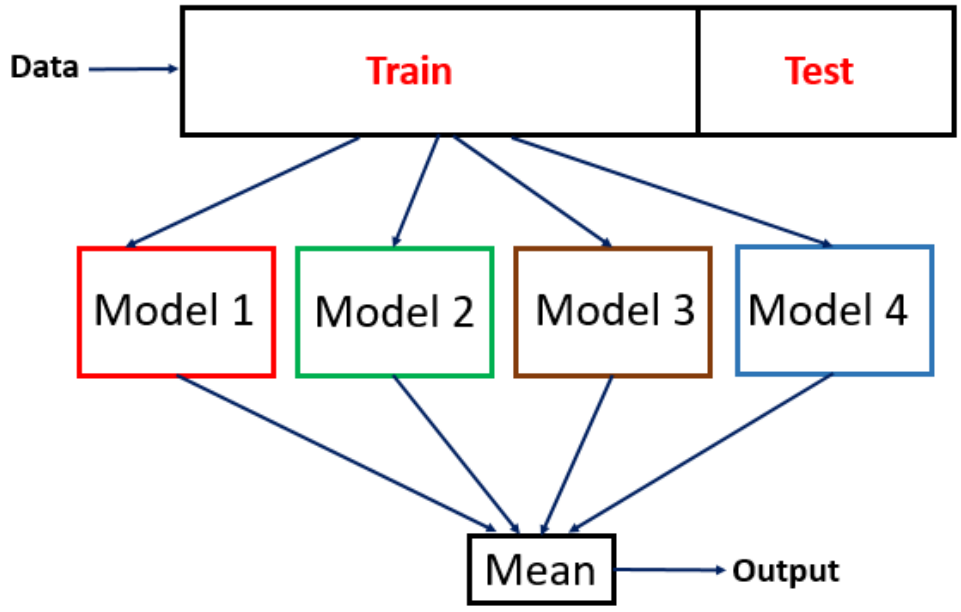
- The last week will be an introduction to neural networks
- As this is a large topic for one week, please have a look at the following "homework" before next week
- The videos from the channel 3Blue1Brown are exceptionally good and build important intuition before we start with the lecture
- Make sure to watch videos one to three, the fourth one is optional
- https://www.youtube.com/playlist?list=PLZHQObOWTQDNU6R1_67000Dx_ZCJB-3pi

Today

1. Ensembles
2. Bagging
3. Random Forests
4. Boosting
5. Guided coding

Ensembles

Simple Ensembles



- The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.
- Ensembles are constructed using a **committee** of learners whose output are combined to formulate a decision
- For a regression task, this might involve taking the **mean** of each learner's output $\hat{y}_1 \ldots \hat{y}_C$
- For a classification task, we would take the **majority vote** of each learner's output

Ensemble Applied to Trees

- Classification trees are simple but often unstable. Solution: combine many trees
- Bagging (Breiman 1996): Fit many large trees to bootstrap resampled versions of the training data.
- Random forests (Breiman 1999): Improvements over bagging with some randomization on features.
- Boosting (Freund & Schapire 1996): Sequentially fit many large or small trees to reweighted versions of the training data.

Bagging

Bagging

- **Bootstrap aggregation**, or **bagging**, is a general-purpose procedure for reducing the variance of a statistical learning method; we introduce it here because it is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n independent observations $Z_1, \ldots, Z_n$, each with variance σ^2, the variance of the mean $\bar{Z}$ of the observations is given by σ^2/n.
- In other words, **averaging a set of observations reduces variance**. Of course, this is not practical because we generally do not have access to multiple training sets.

Bagging— continued

- Instead, we can bootstrap, by taking repeated samples from the (single) training data set.
- In this approach we generate B different bootstrapped training data sets. We then train our method on each of the b bootstrapped training set in order to get $\hat{f}^{*b}(x)$, the prediction at a point x. For regression, we then average all the predictions to obtain

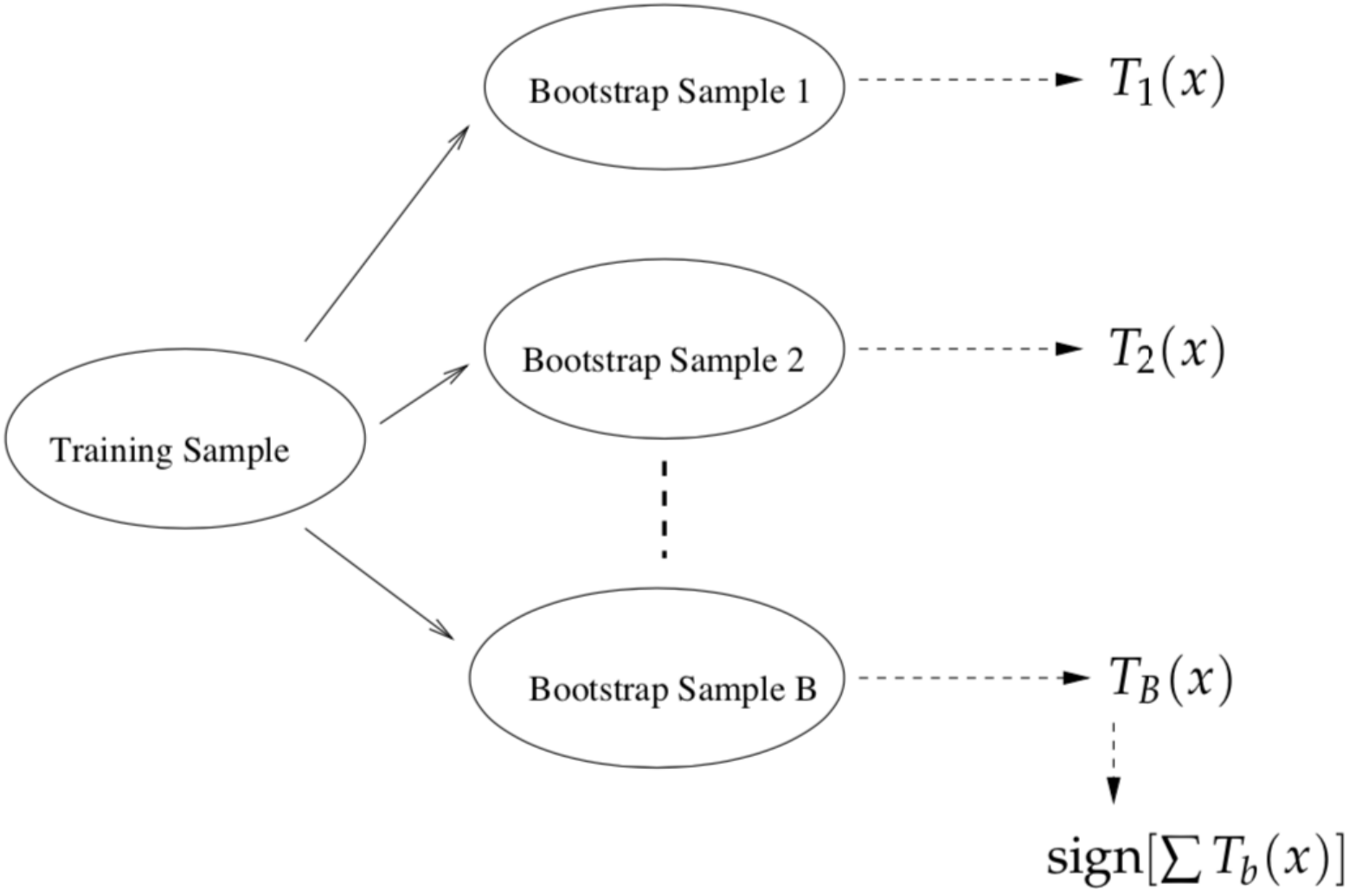
$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^{*b}(x)$$

- For classification we take majority vote

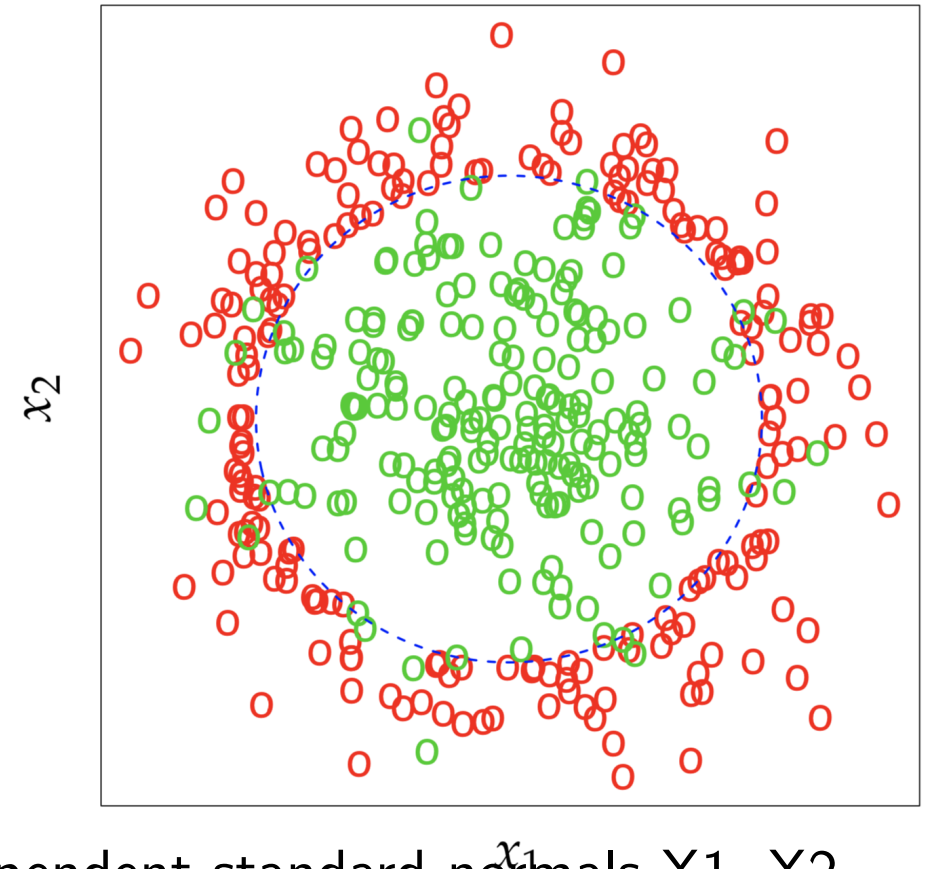
Schematics of Bagging with Trees

- Sample with replacement from the training data $(x_1, y_1), \ldots, (x_n, y_n)$ to obtain a bootstrap sample $(x_1^*, y_1^*), \ldots, (x_n^*, y_n^*)$.
- Construct a new tree T_1^*.
- Repeat everything B times, obtaining B trees $T_1^*, \ldots, T_B^*$.
- Given a new point x, predict average or classify by majority vote among $T_1^*(x), \ldots, T_B^*(x)$.
- Each individual tree is not pruned, so has high variance. Averaging the B trees reduces the variance.

Schematics of Bagging with Trees

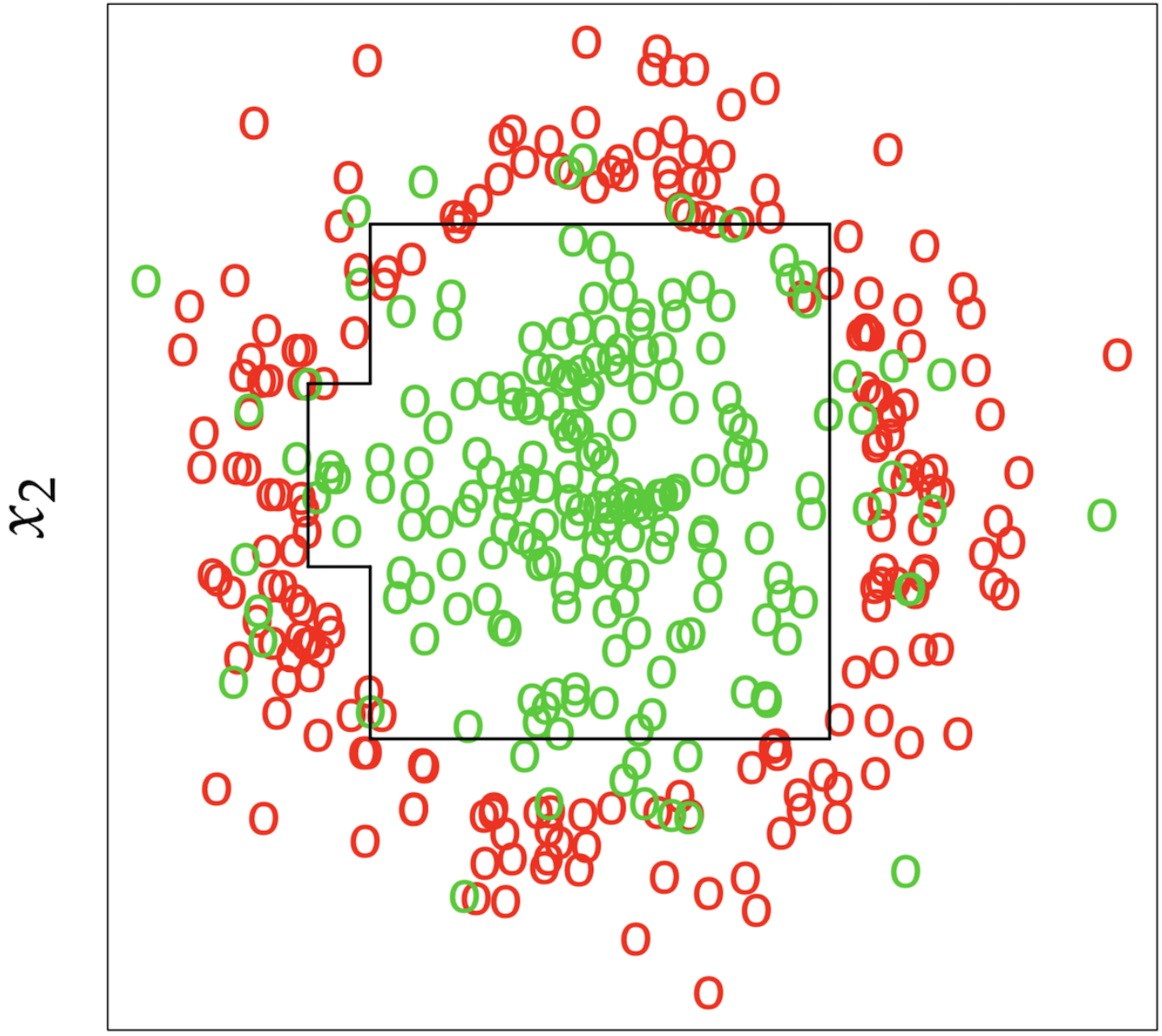


Example: Donut Data

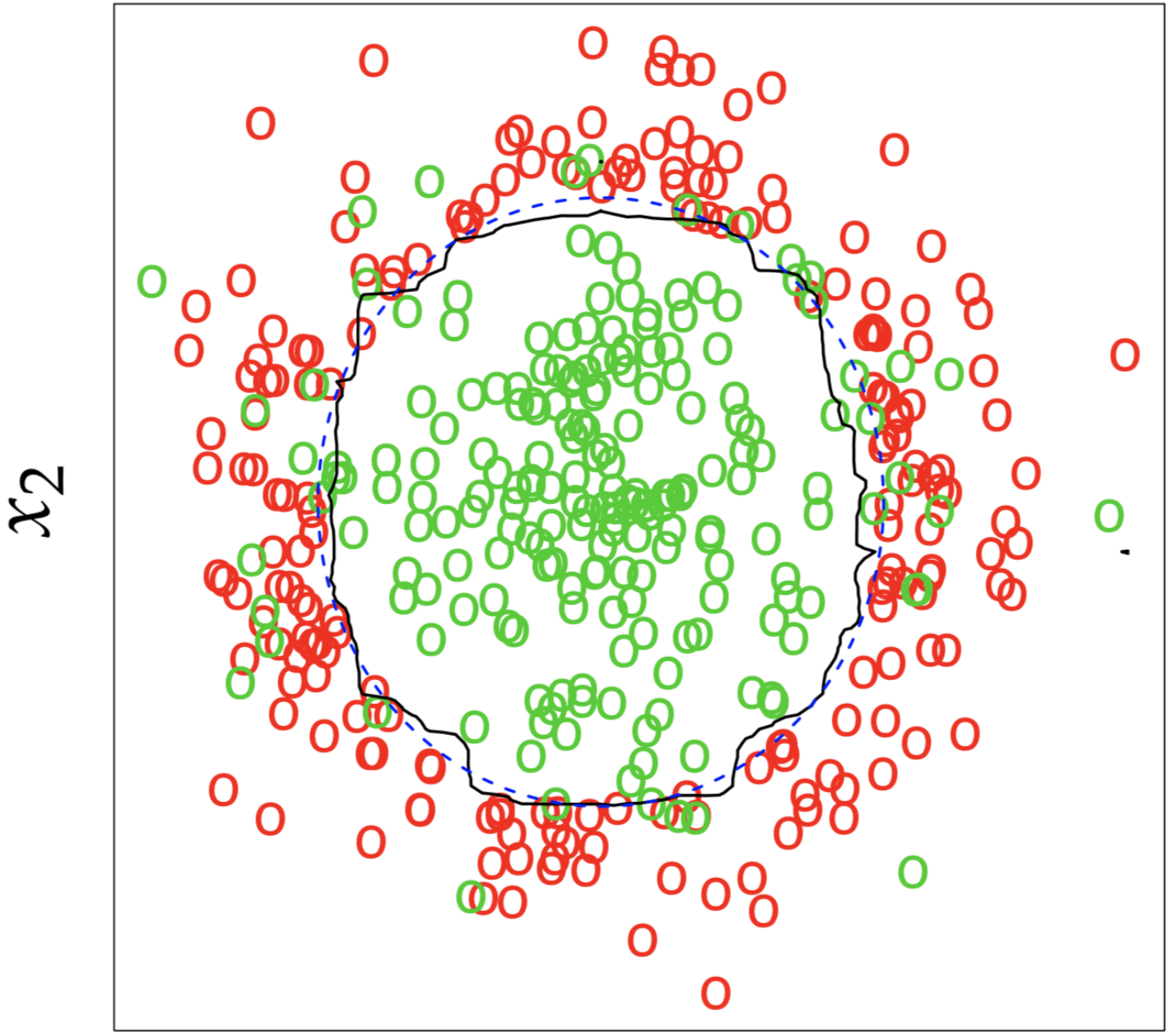


- Two independent standard normals X1, X2
- Green class: $X_1^2 + X_2^2 < 4.6$
- Red class: $X_1^2 + X_2^2 \geq 4.6$
- Plus some noise

Donut Data: Single tree decision boundary



Donut Data: Bagging decision boundary



Bagging the heart data

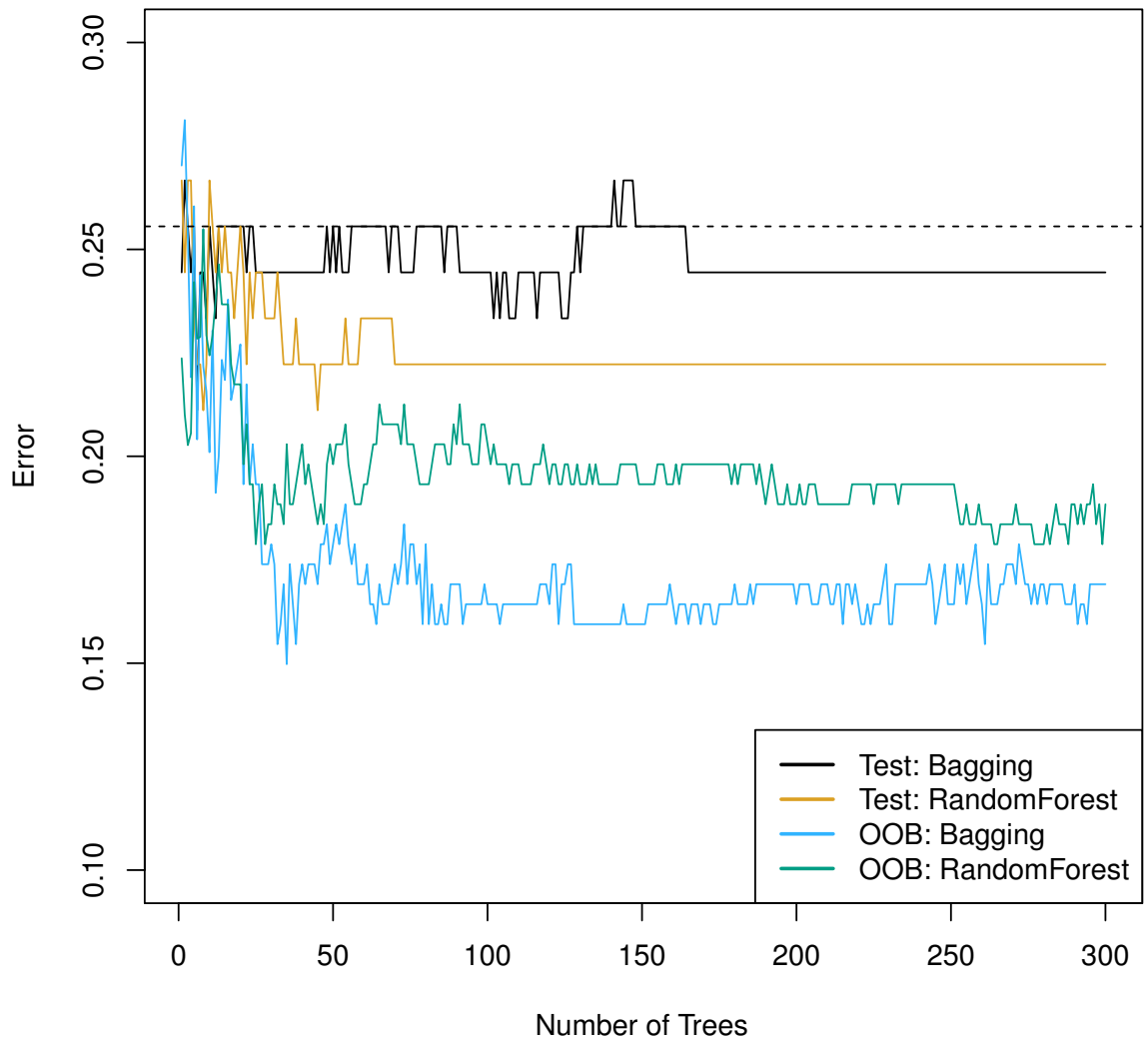


Figure 1: Bagging and random forest results for the Heart data.

Details of previous figure

- The test error (black and orange) is shown as a function of B, the number of bootstrapped training sets used.
- More about random forests in a bit.
- The dashed line indicates the test error resulting from a single classification tree.
- The green and blue traces show the **OOB error**, which in this case is considerably lower

Out-of-Bag Error Estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- The remaining one-third of the observations not used to fit a given bagged tree are referred to as the **out-of-bag (OOB)** observations.
- We can predict the response for the *ith* observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the *ith* observation, which we average.
- This estimate is essentially the LOO cross-validation error for bagging, if B is large.

Remarks on Bagging

- Bagging reduces the variance of individual high variance classifiers such as decision trees.
- Interpretability of inidividual classifiers is lost.

Random forests

Random forests: Main idea

- Create B boostrapped training sets as in bagging.
- Grow a decision tree on each bootstrapped training data set, but consider a **random subset of variables** at each split. The trees are not pruned.

Random Forests

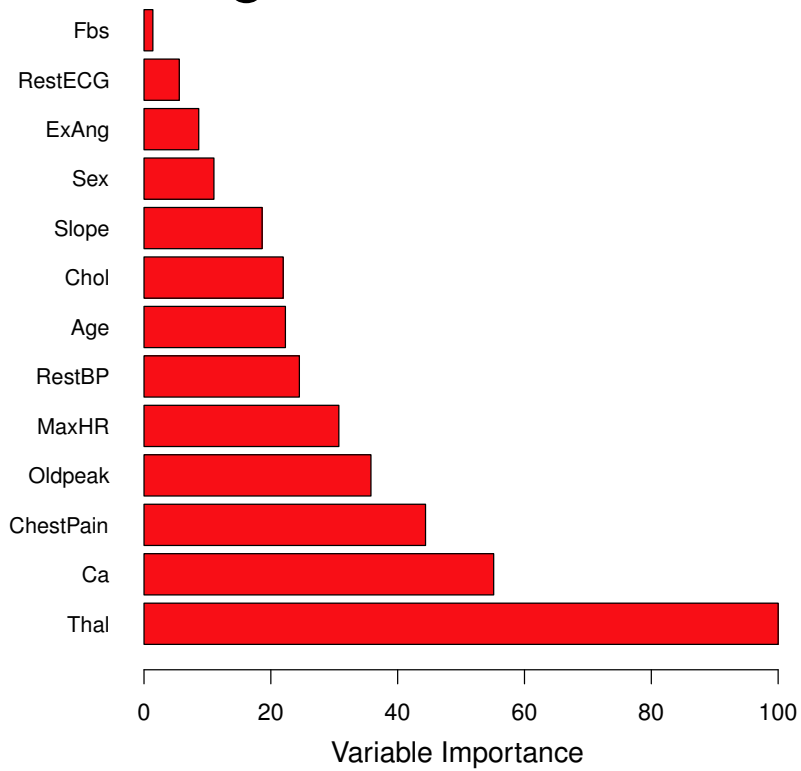
- **Random forests** provide an improvement over bagged trees by way of a small tweak that further **decorrelates** the trees on the individual samples. This reduces the variance when we average the trees.
- As in bagging, we build a number of decision trees on bootstrapped training samples.
- But when building these decision trees, each time a split in a tree is considered, **a random selection of m predictors** is chosen as split candidates from the full set of p predictors. The split is allowed to use only one of those m predictors. In other words, out of only a random subset m $<$ p predictors at each split, the algorithm chooses the best predictor.
- A fresh selection of m predictors is taken at each split.

Choosing m

- Good baseline value for classification: choose $m \approx \sqrt{p}$ — that is, the number of predictors considered at each split is approximately equal to the square root of the total number of predictors (4 out of the 13 for the Heart data).
- Baseline for regression: $m \approx p/3$
- Can also be chosen through cross validation

Variable importance measure

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Sidenote: A general variable importance measure

- A model-independent variable importance measure: Permute/shuffle the observations in a variable, rerun the prediction, and record the change in some outcome such as RMSE for regression. Do this for all variables and compare.
- Example for (classification) trees:
  - For each b, permute the values of the variable in question in the OOB sample at random.
  - Classify the original OOB sample and the permuted OOB sample using the tree T_b.
  - Measure drop in classification performance (using either classification error or Gini).
  - Average over all bootstrap samples B.
  - Repeat for all variables.
- Final output: Increase in error if individual variables are randomly permuted.

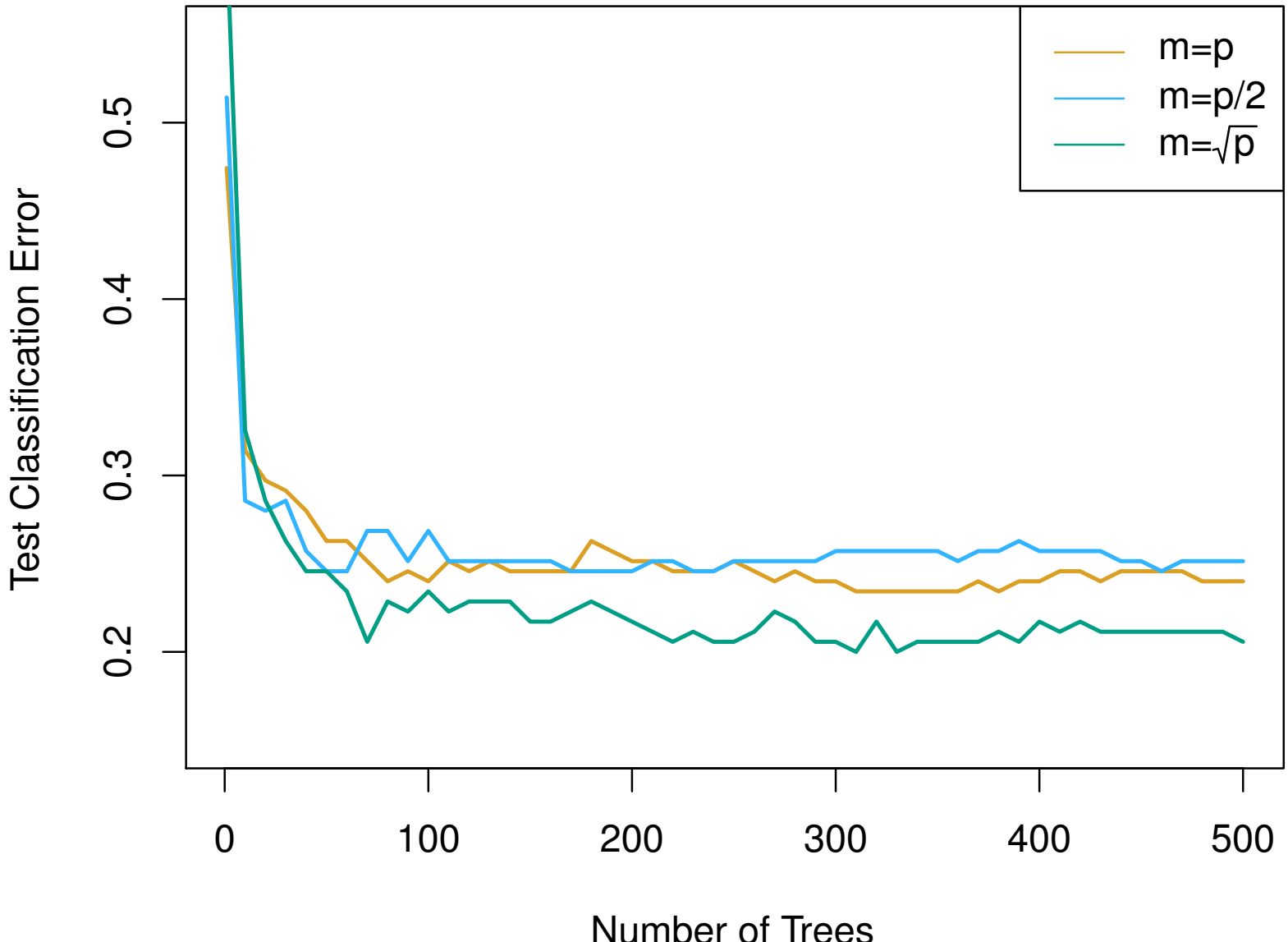
Variable importance measures caveats

- One caveat for both tree specific and plain-vanilla permutation importance measures: Correlations between variables.
- For example, if two variables are highly correlated, removing one would actually imply that the other one picks up (part of) its job when the model is re-estimated. Hence simple variable importance measure can overestimate the importance of variables.
- See also e.g. Strobl et al. (2008) for a discussion and a more advanced importance measure.
- Take variable importance output as a general guideline, but do not over-interpret ranking.

Example: Gene expression data

- We applied random forests to a high-dimensional biological data set consisting of expression measurements of 4,718 genes measured on tissue samples from 349 patients.
- There are around 20,000 genes in humans, and individual genes have different levels of activity, or expression, in particular cells, tissues, and biological conditions.
- Each of the patient samples has a qualitative label with 15 different levels: either normal or one of 14 different types of cancer.
- We use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.
- We randomly divided the observations into a training and a test set, and applied random forests to the training set for three different values of the number of splitting variables m.

Results: gene expression data



Details of previous figure

- Results from random forests for the fifteen-class gene expression data set with $p = 500$ predictors.
- The test error is displayed as a function of the number of trees. Each colored line corresponds to a different value of m, the number of predictors available for splitting at each interior tree node.
- Random forests ($m < p$) lead to a slight improvement over bagging ($m = p$). A single classification tree has an error rate of 45.7%.

Remarks on random forests

- Random forests are considered one of the most competitive classifiers and are popular.
- Random selection of variables further controls overfitting.
- For most data sets results seem not too sensitive to m, the number of variables used for splitting.
- While averaged models lack interpretability, variables can be ranked by importance.
- However, importance rankings can be much more variable than the classification results themselves.

Boosting

Boosting

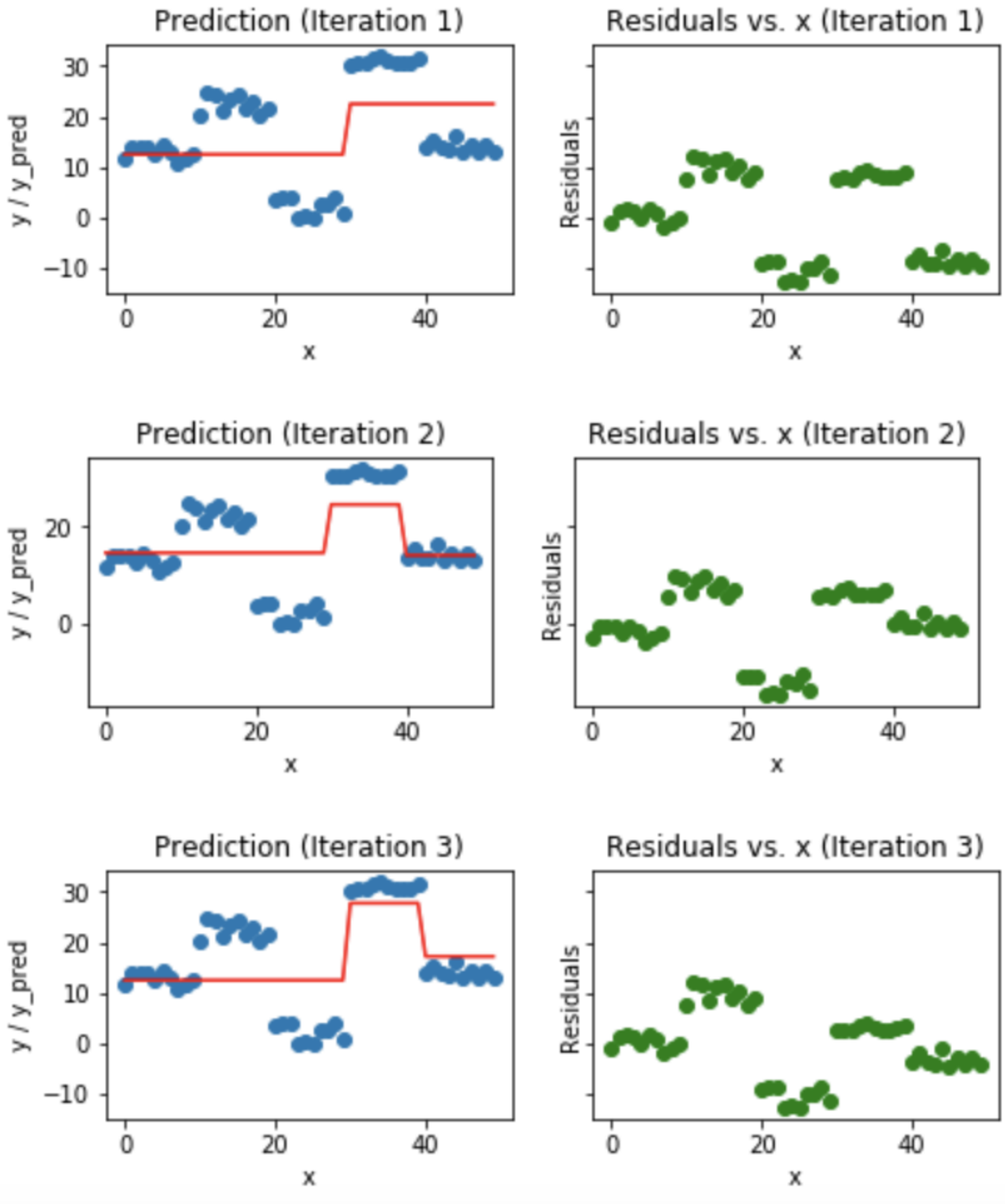
- Like bagging, boosting is a general approach that can be applied to many statistical learning methods for regression or classification. We only discuss boosting for decision trees.
- Recall that bagging and random forest involve creating multiple bootstrap samples of the original training data set, fitting a separate decision tree to each each of them, and then combining all of the trees in order to create a single predictive model.
- Notably, each tree is built on a bootstrap data set, independent of the other trees.
- In boosting trees are grown sequentially: Each tree is grown using information from previously grown trees.

Boosting algorithm for regression trees

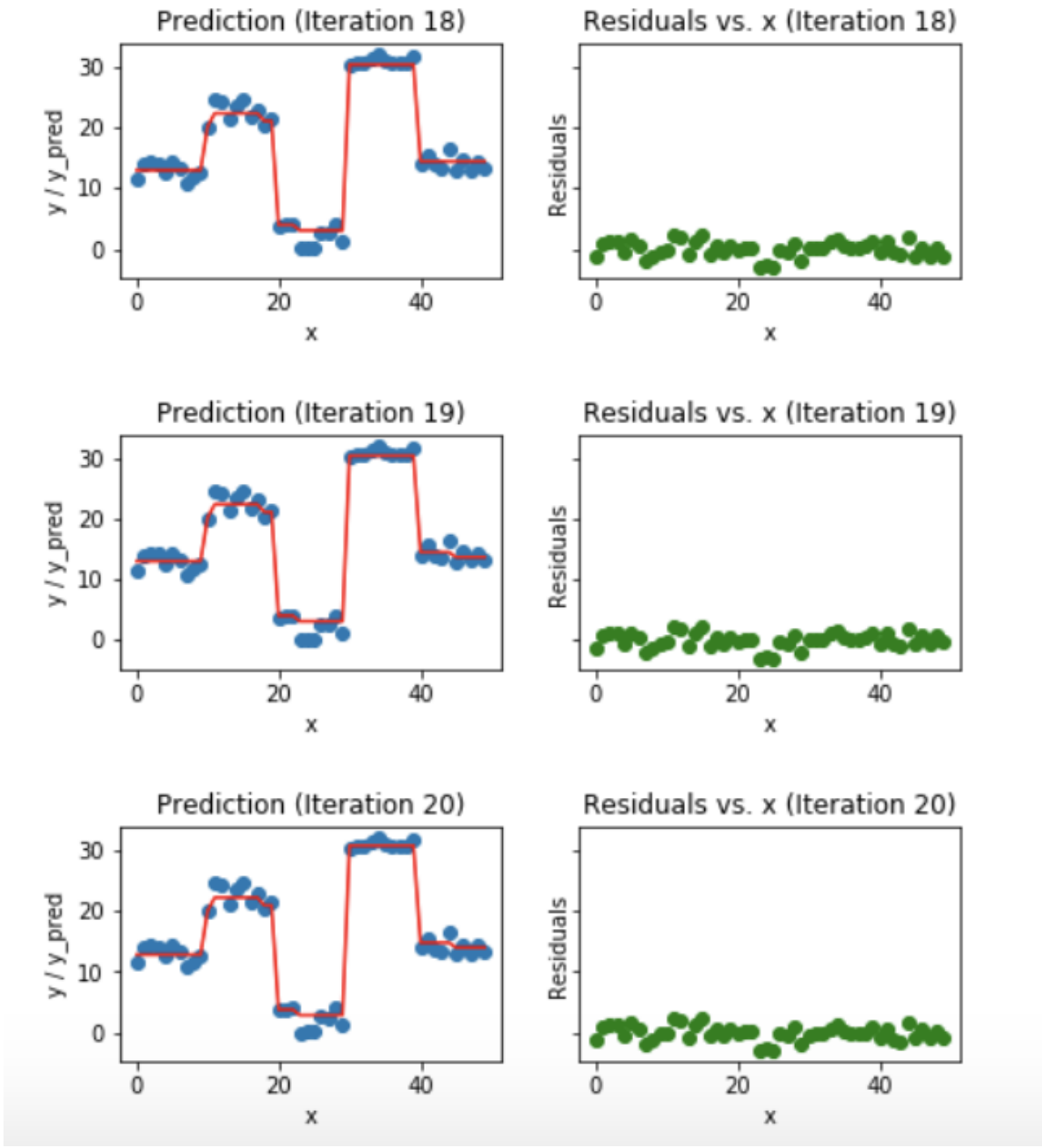
1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \ldots, B$, repeat:
   - Fit a tree $\hat{f}_b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   - Update $\hat{f}$ by adding in a shrunken version of the new tree: $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$.
   - Update the residuals, $r_i \leftarrow r_i - \lambda \hat{f}_b(x_i)$.
3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}_b(x)$$

Visualizing Boosting



Visualizing Boosting



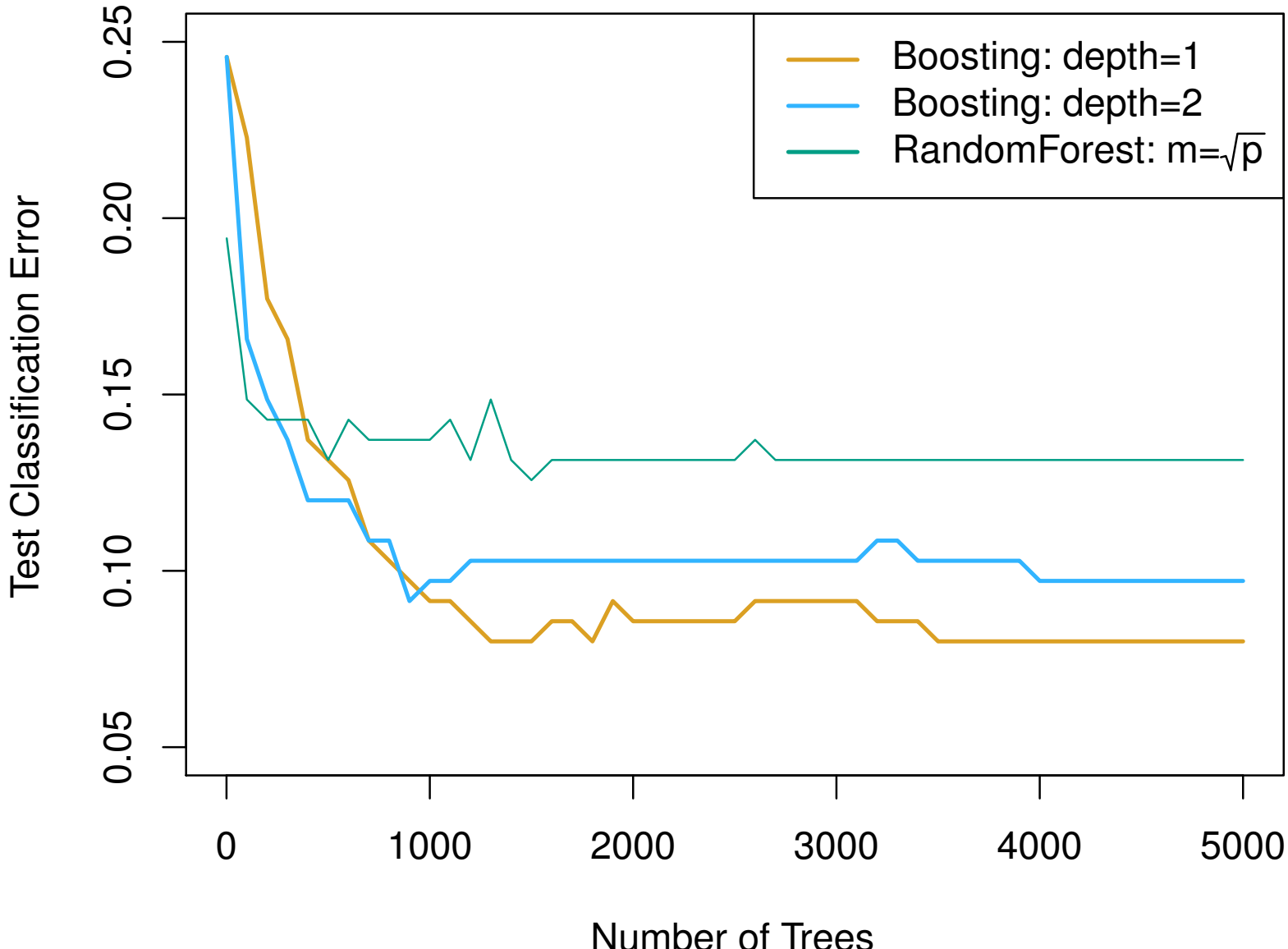
What is the idea behind this procedure?

- Unlike fitting a single large decision tree to the data, which amounts to **fitting the data hard** and potentially overfitting, the boosting approach instead **learns slowly**.
- Given the current model, we fit a decision tree to the residuals from the model. We then add this new decision tree into the fitted function in order to update the residuals.
- Each of these trees can be rather small, with just a few terminal nodes, determined by the parameter d in the algorithm.
- By fitting small trees to the residuals, we slowly improve $\hat{f}$ in areas where it does not perform well. The shrinkage parameter λ slows the process down even further, allowing more and different shaped trees to attack the residuals.
- Convert many "weak" learners into a complex predictor.

Boosting for classification

- Boosting for classification is similar in spirit to boosting for regression, but is a bit more complex. We will not go into detail here, nor do we in the text book.
- Students can learn about the details in **Elements of Statistical Learning, chapter 10**.
- The R package `gbm` (gradient boosted models) handles a variety of regression and classification problems.

Gene expression data continued



Details of previous figure

- Results from performing boosting and random forests on the fifteen-class gene expression data set in order to predict cancer versus normal.
- The test error is displayed as a function of the number of trees. For the two boosted models, $\lambda = 0.01$. Depth-1 trees slightly outperform depth-2 trees, and both outperform the random forest, although the standard errors are around 0.02, making none of these differences significant.
- The test error rate for a single tree is 24%.

Tuning parameters for simple boosting model

1. The **number of trees** B. Unlike bagging and random forests, boosting can overfit if B is too large, although this overfitting tends to occur slowly if at all. We use cross-validation to select B.
2. The **shrinkage parameter** λ, a small positive number. This controls the rate at which boosting learns. Typical values are 0.01 or 0.001, and the right choice can depend on the problem. Very small λ can require using a very large value of B in order to achieve good performance.
3. The **number of splits** d in each tree, which controls the complexity of the boosted ensemble. Often $d = 1$ works well, in which case each tree is a **stump**, consisting of a single split and resulting in an additive model. More generally d is the **interaction depth**, and controls the interaction order of the boosted model, since d splits can involve at most d variables.

Remarks on boosting

- Boosting works well with trees, but can in principle be applied to any classifier.
- Boosting can overfit, hyper parameter tuning is important.
- In more current libraries, there are many more tuning parameters.
- Interpretable structure is also lost.

Boosting: Current algorithms

- Among the strongest algorithms from statistical machine learning today are some extensions of the simple benchmark boosting algorithm outlined above.
- Frequently win prediction competitions and are used for code in production.
- **Bold** words are links from here onwards.
- Prominent algorithms and packages are **XGBoost** and **LightGBM** which exist for R and Python (and other languages).
- Many options in regard to model flexibility, regularisation, loss functions, etc.

LightGBM key features

- Grows trees leaf-wise instead of level-wise (see e.g. the **documentation**)
- Very fast
- **Very flexible**, for example:
- Single and multi-class classification
- Regression with a range of loss functions (e.g. Huber loss which is more robust against outliers, or e.g. Gamma or Poisson for modeling some positive outcomes)
- Can handle missing values
- Can be used in R, Python, and C
- Full **paper** for technical details (NIPS, 2017)

Summary

- Decision trees are simple and interpretable models for regression and classification
- However they are often not competitive with other methods in terms of prediction accuracy
- Bagging, random forests and boosting are good methods for improving the prediction accuracy of trees. They work by growing many trees on the training data and then combining the predictions of the resulting ensemble of trees.
- The latter two methods— random forests and boosting— are among the state-of-the-art methods for supervised learning. However their results can be difficult to interpret.

Guided coding

- 01-bagging-and-rf.Rmd
- 02-lightgbm.Rmd