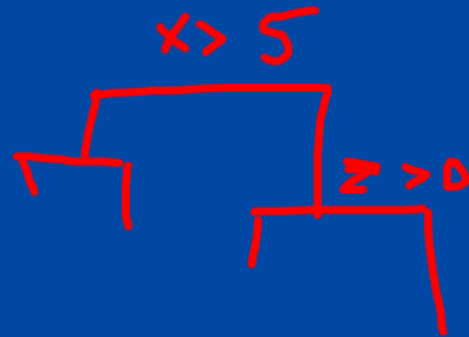




STAT5003

Week 9 : Tree and Ensemble methods

Dr. Justin Wishart



Readings and functions covered

-  readings
 - Tree methods covered in Chapter 8 in James, Witten, Hastie, and Tibshirani (2013)
-  functions
 - `tree::tree` (Create Regression and Decision trees)
 - `base::sample` (Bagging)
 - `ranger::ranger` (construct random forests)
 - `gbm::gbm` (Gradient boosting machines)

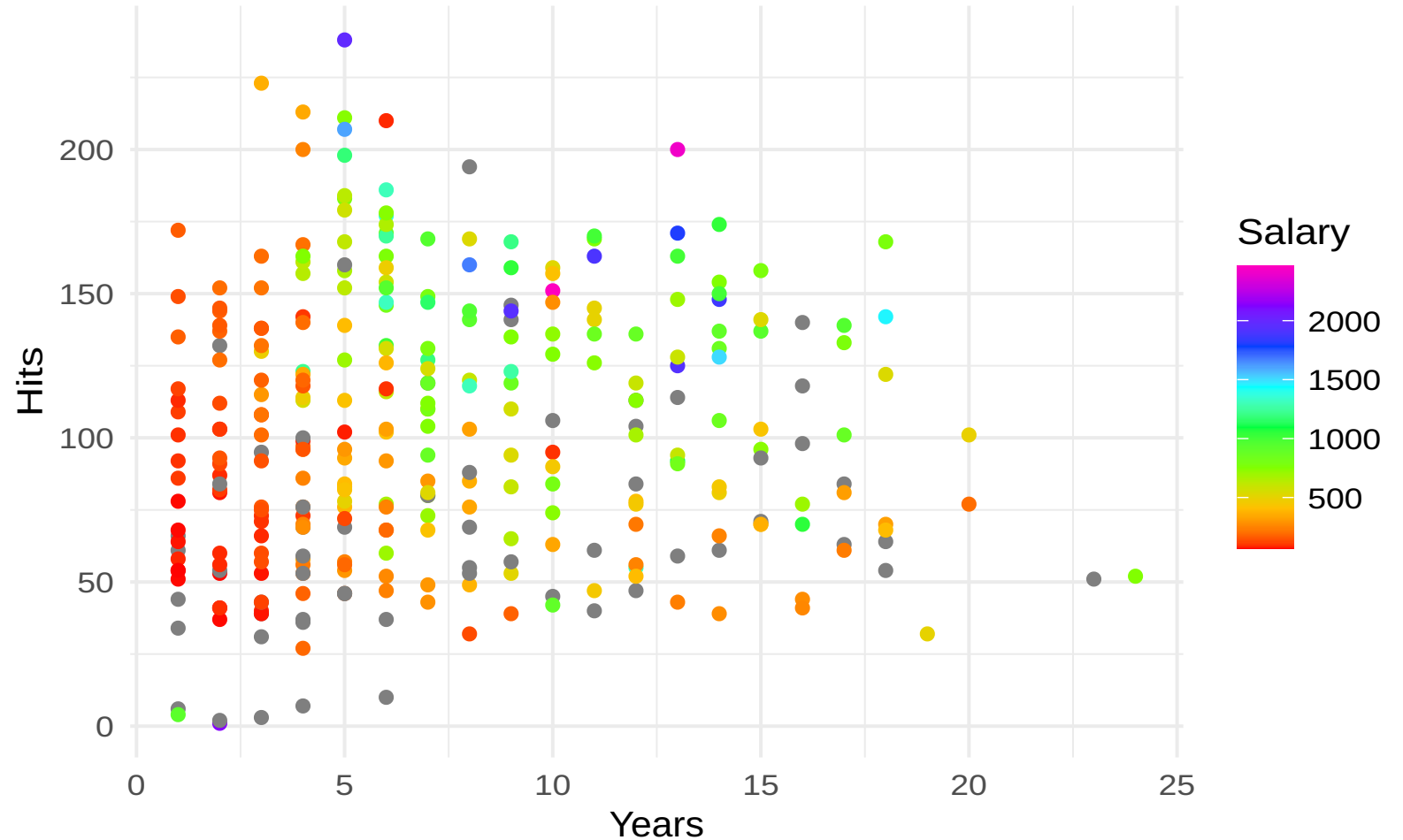
Regression/Decision Trees



THE UNIVERSITY OF
SYDNEY

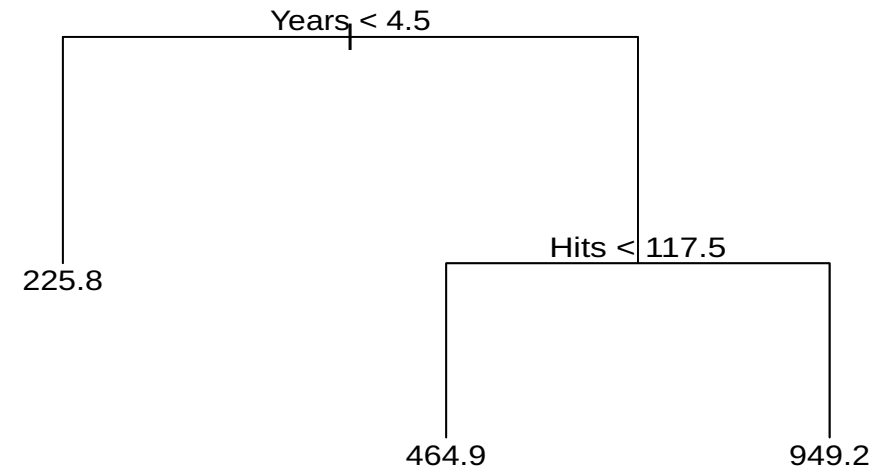
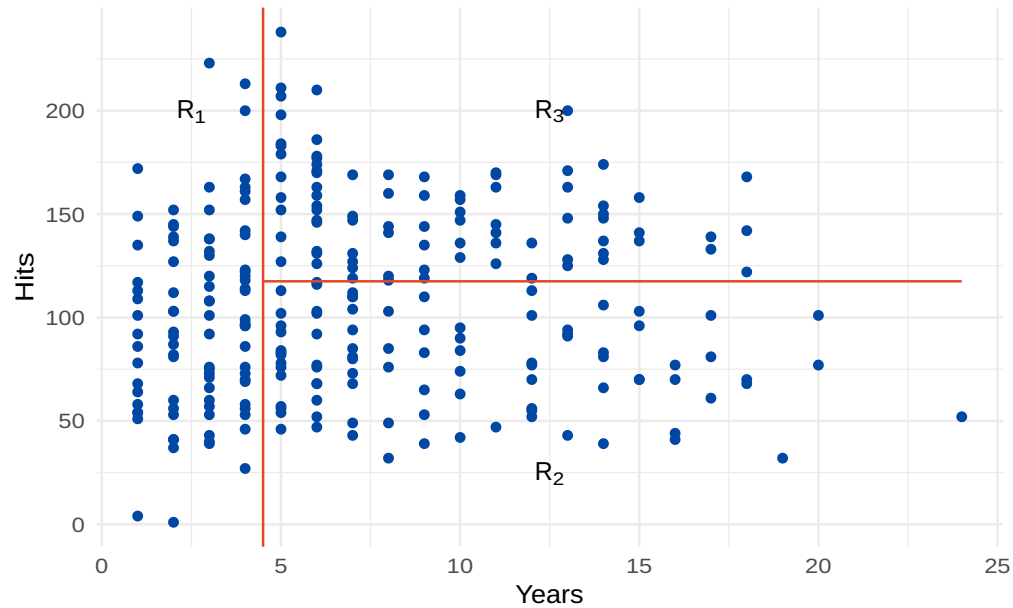
Decision tree for regression

- Baseball player salary data: how would you stratify it?
 - **Salary** is colour-coded (NA values coded grey).



Decision Tree

- Overall, the tree segments the players into three regions of predictor space:
 - $R_1 = \{X | \text{Years} < 4.5\}$
 - $R_2 = \{X | \text{Years} \geq 4.5, \text{Hits} < 117.5\}$
 - $R_3 = \{X | \text{Years} \geq 4.5, \text{Hits} \geq 117.5\}$



Terminology for trees

- In keeping with the tree analogy,
 - the regions R_1 , R_2 and R_3 are known as terminal nodes.
 - Decision trees are typically drawn upside down, in the sense that the leaves are at the bottom of the tree.
 - The points along the tree where the predictor space is split are referred to as internal nodes.
- In the Baseball player salary tree, the two internal nodes are indicated by the text Years < 4.5 and Hits > 117.5.

Interpretation of Results

- **Years** is the most important factor in determining **Salary**,
 - Players with less experience earn lower salaries than more experienced players.
- Given that a player is less experienced,
 - the number of **Hits** that he made in the previous year seems to play little role in his **Salary**.
- Among players who have been in the major leagues for five or more years,
 - the number of Hits made in the previous year does affect Salary,
 - players who made more Hits last year tend to have higher salaries.
- Obviously an over-simplification,
 - compared to some other classification models (such as a regression model), it is easy to display, interpret and explain.

Details on tree building process

- In theory, the decision regions could have any shape. However, we choose to divide the predictor space into high-dimensional rectangles, or boxes, for simplicity and for ease of interpretation of the resulting predictive model
- The goal is to find boxes R_1, \dots, R_J that minimizes the residual sum of squares (RSS), given by:

$$\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$$

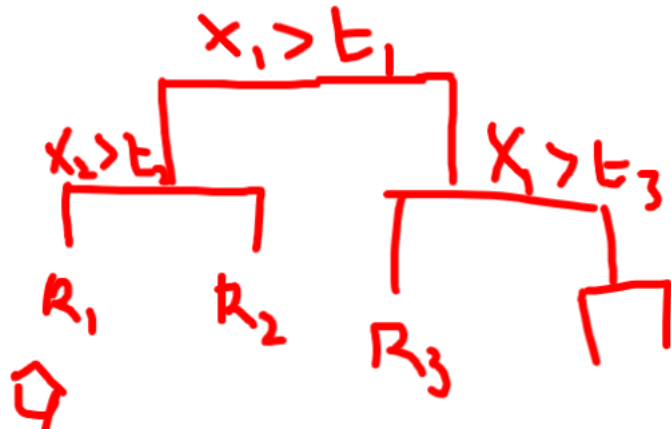
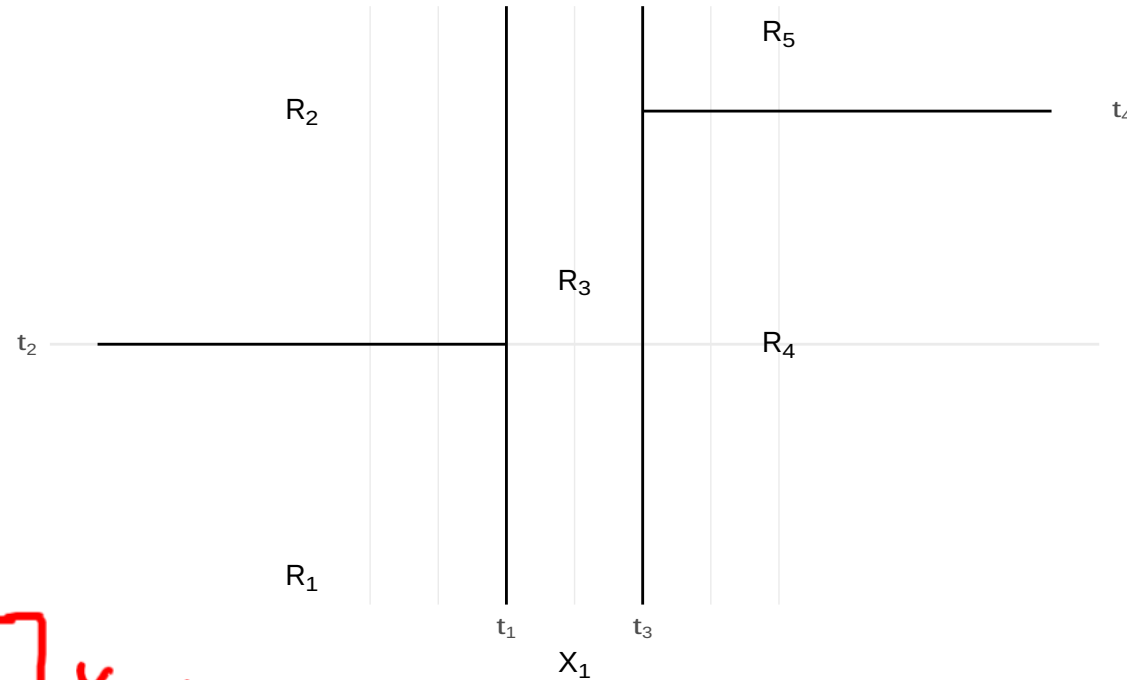
- where \hat{y}_{R_j} is the mean response for the training observations within the j^{th} box.

Tree building with recursive binary splitting

- It is computationally **infeasible** to consider every possible partition of the feature space into J boxes
- Take a top-down, **greedy** approach that is known as **recursive binary splitting**
- The approach is **top-down** because it begins at the top of the tree and then successively splits the predictor space; each split is indicated via two new branches further down on the tree.
- It is **greedy** because at each step of the tree-building process, the best split is made at that particular step,
 - rather than looking ahead and picking a split that will lead to a better tree in some future step.

Using Decision Trees for prediction

- We divide the predictor space
 - the set of possible values for X_1, X_2, \dots, X_p into J distinct and non-overlapping regions, R_1, R_2, \dots, R_J .
- For every observation that falls into the region R_j , we make the same prediction, which is simply the mean of the response values for the training observations in R_j .



Decision trees for classification

- Very similar to a regression tree,
 - exception being the prediction of a qualitative response rather than a quantitative one.
- For a classification tree,
 - Inspect the region that the observation belongs and predict the most commonly occurring class in that region.

Gini index

- Just as in the regression setting, we use recursive binary splitting to grow a classification tree.
- In the classification setting, RSS cannot be used as a criterion for making the binary splits
- Alternative measure such as Gini index is used instead.
- The Gini index is defined by

$$G = \sum_j \sum_{k=1}^K \hat{p}_{jk}(1 - \hat{p}_{jk})$$

- where \hat{p}_{jk} represents the proportion of training observations in the j^{th} region that are from the k^{th} class.

Gini index

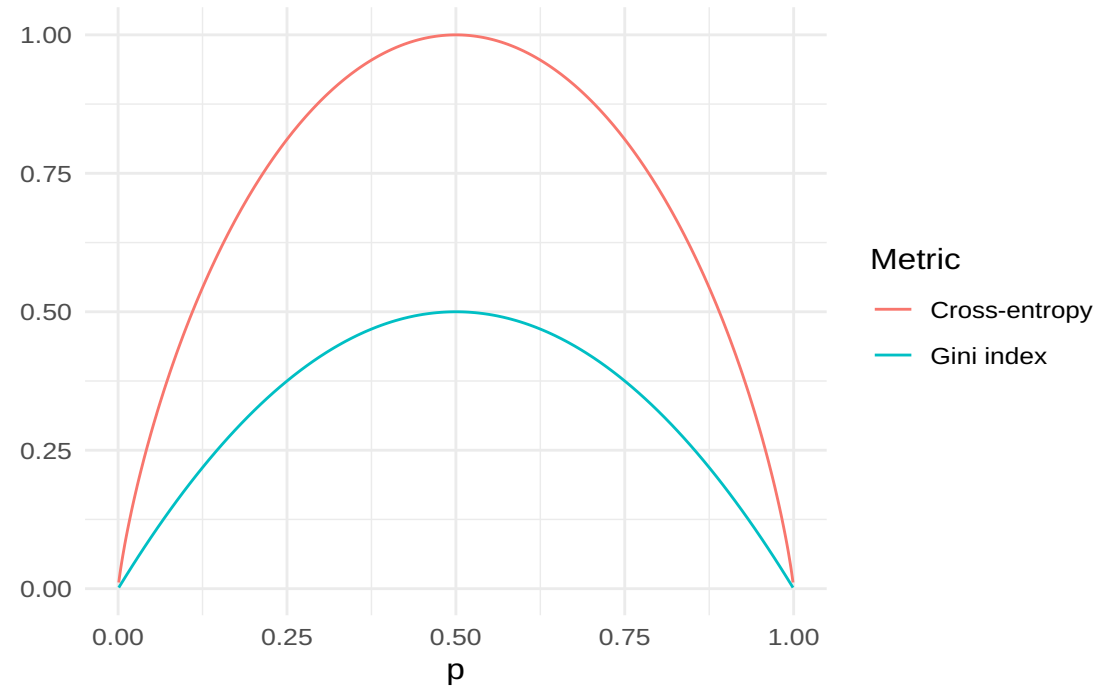
- The Gini index is a measure of total variance across the K classes.
 - It takes on a small value if all of the \hat{p}_{jk} values are close to zero or one.
 - This occurs when there is a clear majority class!
- For this reason the Gini index is referred to as a measure of node **purity**
 - a small value indicates that a node contains predominantly observations from a single class.

Cross-entropy

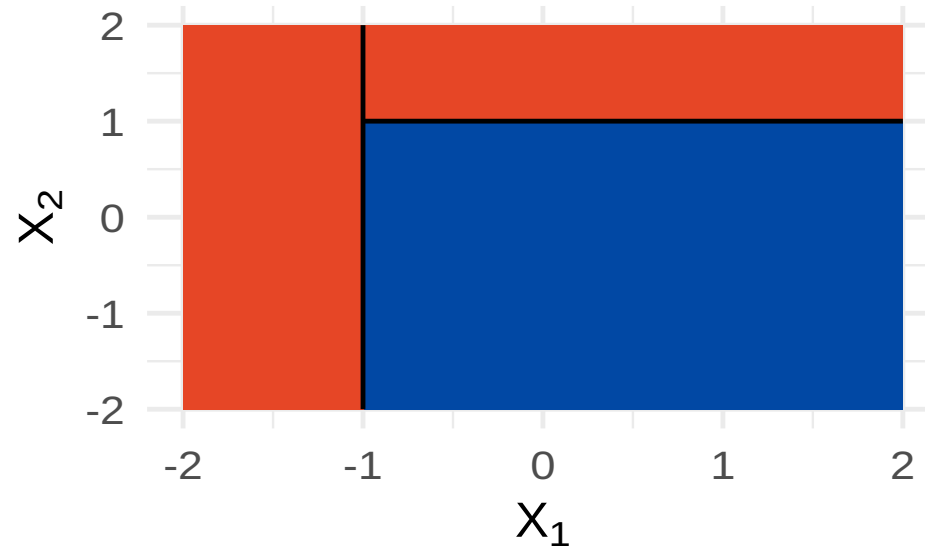
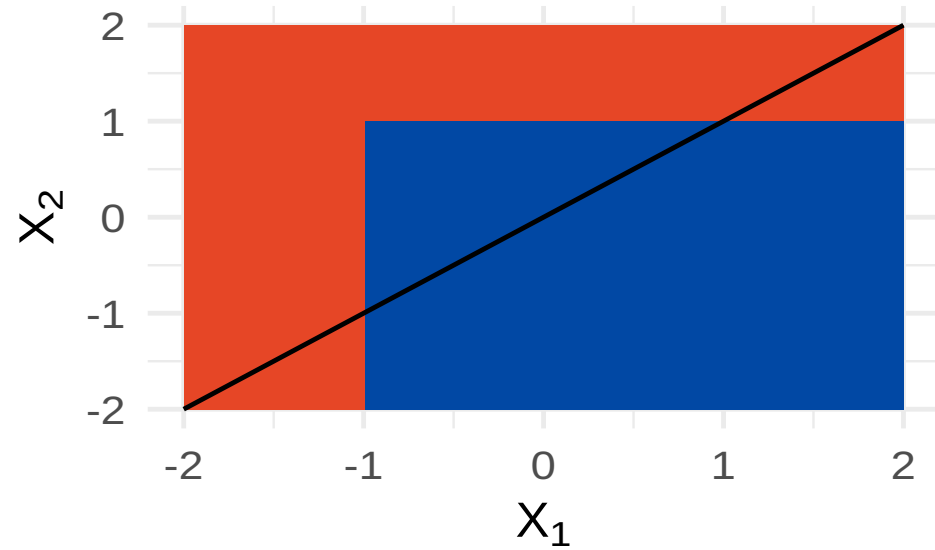
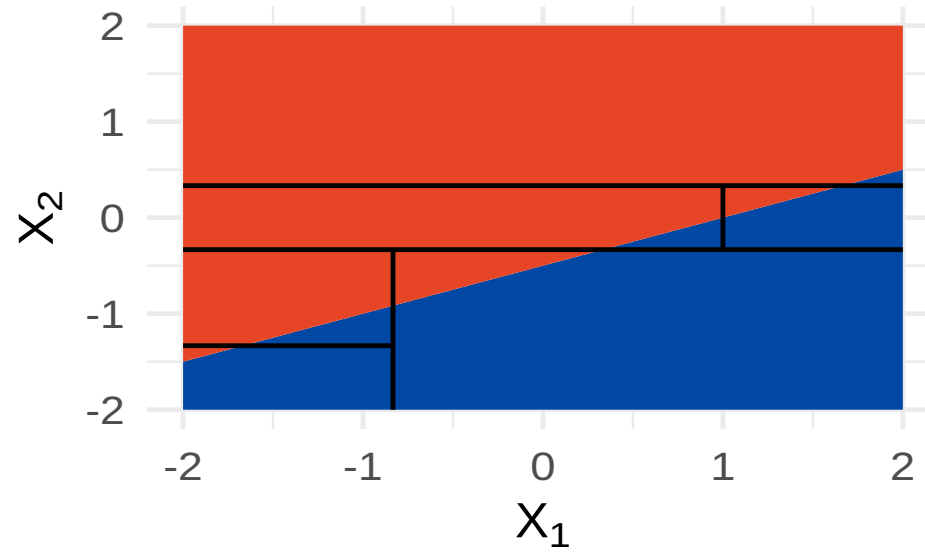
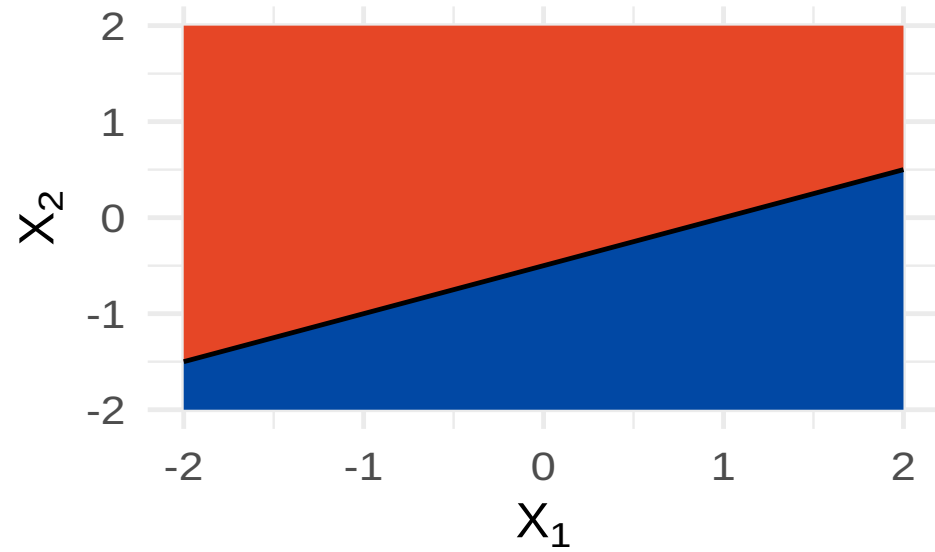
- An alternative to the Gini index is cross-entropy, given by

$$D = - \sum_m \sum_{k=1}^K \hat{p}_{mk} \log \hat{p}_{mk}$$

- It turns out that the Gini index and the cross-entropy are very similar numerically.



Tree vs linear model



Advantages and disadvantages of trees

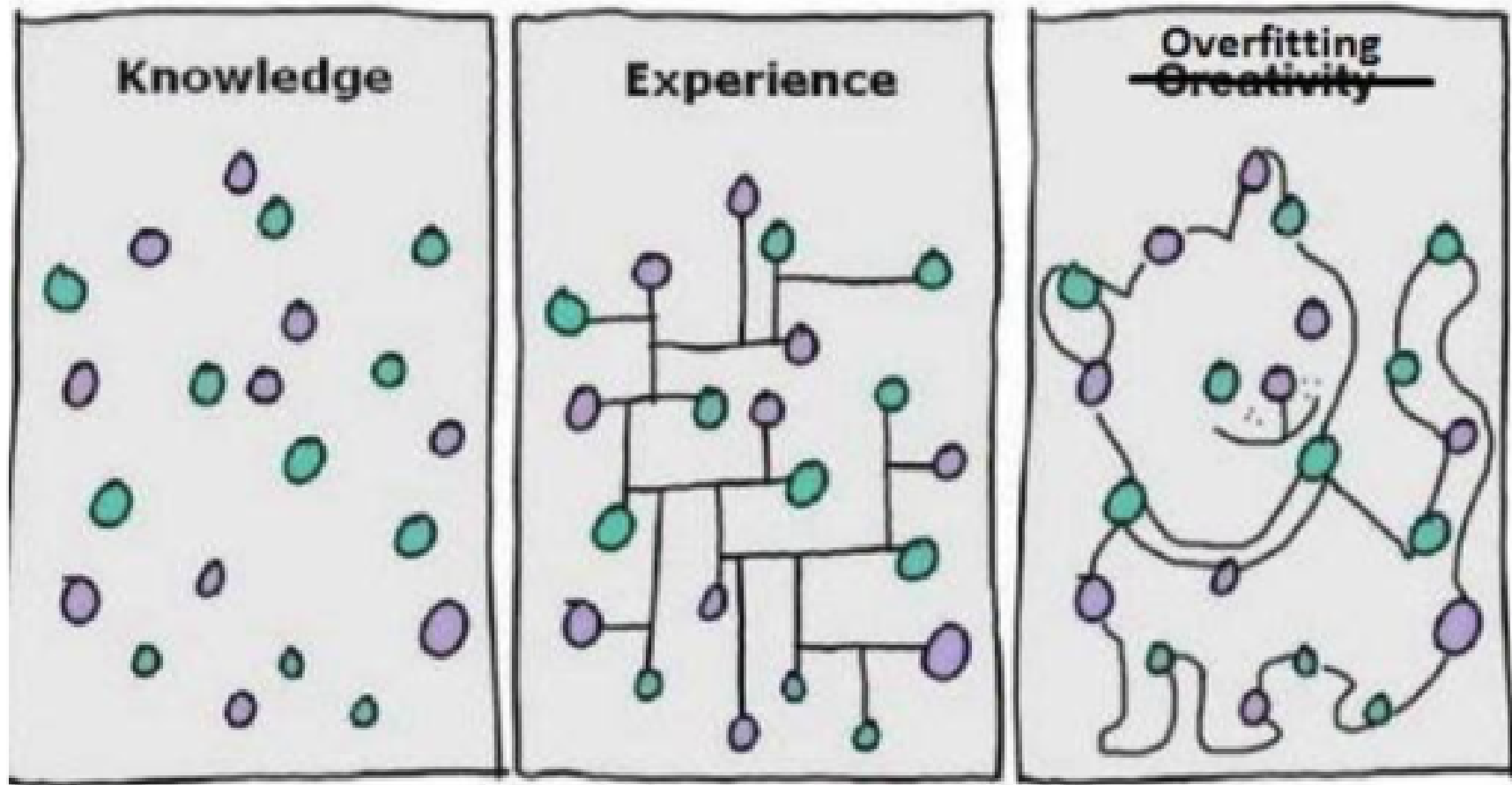
Advantages:

- Trees are very easy to explain to people
- Some people believe that decision trees closely relate to human decision-making
- Trees can be displayed graphically
- Can handle different data types and doesn't require scaling

Disadvantages:

- Trees do not have the same level of predictive accuracy as some of the other regression and classification approaches we have discussed.

A single decision tree is prone to over-fitting



Ensemble methods via bootstrapping



THE UNIVERSITY OF
SYDNEY

Ensemble of trees

- An alternative is available than just relying on one tree and hope we make the right decision at each split.

Ensemble Methods

- Take a sample of Decision Trees into account
- Calculate which features to use at each split
- Make final prediction model using the **aggregated** result from an **ensemble** of trees.

Bagging (Bootstrap Aggregation)

- Bootstrap aggregation, or bagging, is a general purpose procedure for **reducing the variance** of a statistical learning method. It is particularly useful and frequently used in the context of decision trees.
- Recall that given a set of n observations Z_1, \dots, Z_n each with a variance of σ^2 ,
 - the variance of the mean \bar{Z} is given by σ^2/n
- In other words, averaging a set of observations reduces variance.
- Since is it typically not possible to have access to multiple training sets,
 - we can use **bootstrapping** to create multiple training sets.

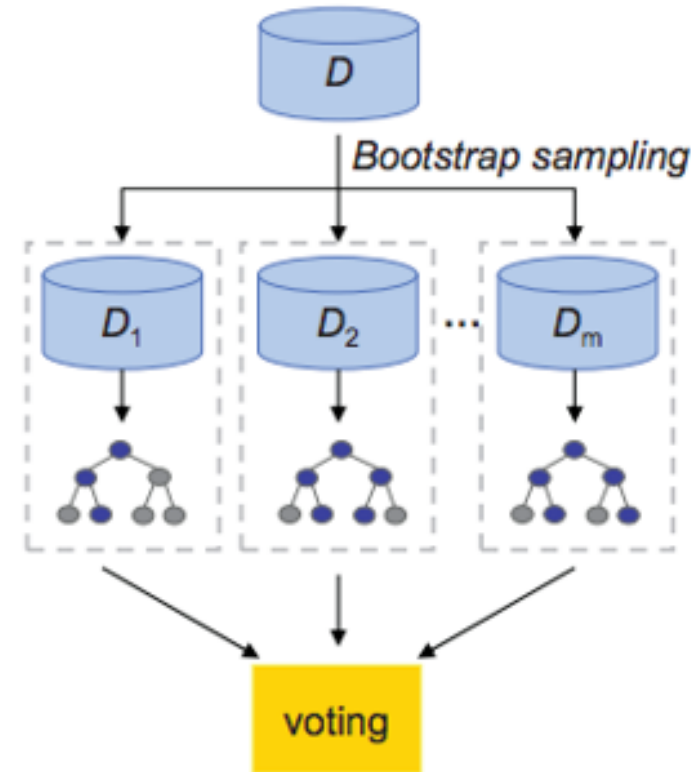
Bagging continued

- Use bootstrapping to take repeated samples from a (single) training data set.
- In this approach, we generate B different bootstrapped training data sets.
 - Train our method of the b^{th} bootstrapping set in order to obtain $\hat{f}_b^*(x)$, the prediction at a point x .

- Average all the observations to obtain:

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

- This is called bagging



Out of bag error estimation

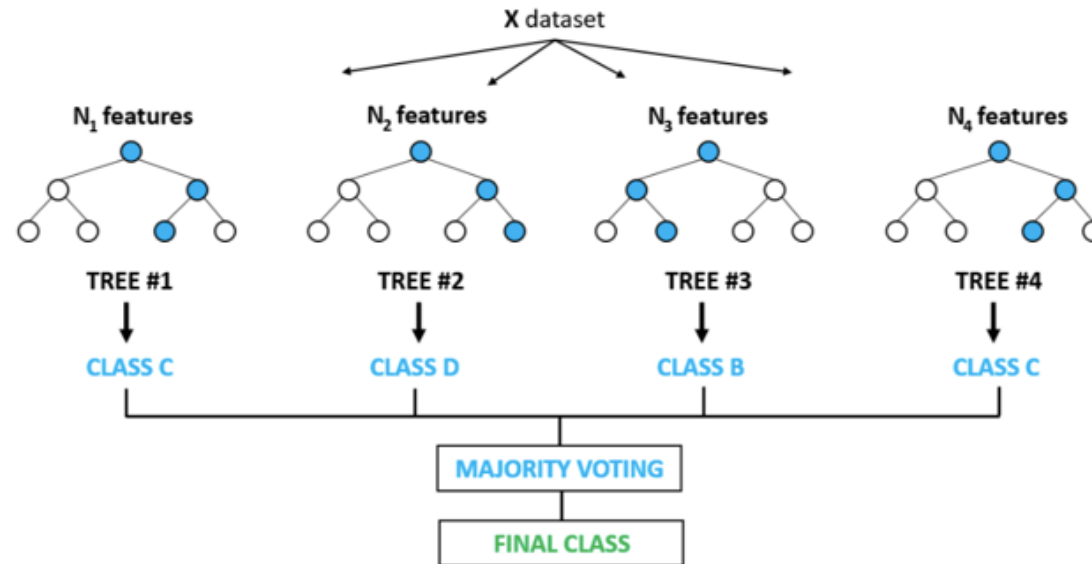
- It turns out that there is a very straightforward way to estimate the test error of a bagged model.
- Recall that the key to bagging is that trees are repeatedly fit to bootstrapped subsets of the observations. One can show that on average, each bagged tree makes use of around two-thirds of the observations.
- 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 i^{th} observation using each of the trees in which that observation was OOB. This will yield around $B/3$ predictions for the i^{th} observation, which we average. This estimate is essentially the LOO cross-validation error for bagging, if B is large.

From bagging to Random Forest

- Random forests (some times) provide an improvement over bagged trees by way of a small tweak that decorrelates the trees. 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.
- A fresh selection of m predictors is taken at each split, and typically we 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.

Random Forest algorithm pseudocode

```
Set number of models to build, B
for i = 1 to B
  Generate a bootstrap sample of the original data
  Train a tree model on this sample where
    for each split
      Randomly select  $m$  ( $< p$ ) of the original predictors
      select the best predictor among the  $k$  predictors and partition the data
    endfor
endfor
```



Ensemble methods via boosting



THE UNIVERSITY OF
SYDNEY

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 involves creating multiple copies of the original training data set using the bootstrap, fitting a separate decision tree to each copy, 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.
- Boosting works in a similar way, except that the trees are grown *sequentially*: each tree is grown using information from previously grown trees.

Bagging : independent

Boosting: sequentially

Bagging 这种有放回的采样策略，可以减少 over-fitting，而 Boosting 会修正那些错分类的样本，因此能提高准确率（但也可能导致 overfitting ）。

Bagging 由于样本之间没有关联，因此它的训练是可以并行的，比如 Random Forest 中，每一棵决策树都是可以同时训练的。Boosting 由于需要考虑上一步错分类的样本，因此需要顺序进行。

Idea behind boosting

- 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 which just a few terminal nodes.
- By fitting small trees to the residuals, we slowly improve $f(x)$ 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.

Boosting for regression trees

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
 - Here r_i denotes the i^{th} residual and y_i the outcome.
2. For $b = 1, 2, \dots, B$
 - Fit a tree \hat{f}_b with d splits ($d + 1$ terminal nodes) to the new training data (X, r) .
 - That is r is the new response value.
 - Update \hat{f} by adding in a shrunk version of the new tree
 - $\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}_b(x)$
 - Update the residuals
 - $r_i \leftarrow r_i - \lambda \hat{f}_b(x)$
3. Compute the final model

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

Parameters to tune in boosting

- Number of trees B
- The shrinkage parameter λ a small positive number.
 - Typical values are between 0.01 and 0.001.
- Number of split d in each tree.
 - Controls the complexity of the boosted ensemble.
 - If $d = 1$, then the tree is just a stump. This actually usually works quite well.

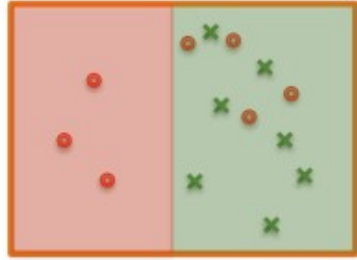
Other boosting algorithms

- AdaBoost
- Stochastic gradient boosting
- XGBoost

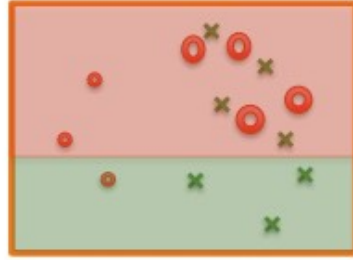
AdaBoost

- Short for Adaptive Boosting, one of the first boosting algorithms
- Convert a set of weak classifiers into a strong one
- Basic idea
 - at each iteration, reweight the data to place more weight on data points that the classifier got wrong
- At the end, combine all the weak classifiers by taking a weighted combination. Put more weight on the weak classifiers with the higher accuracies.

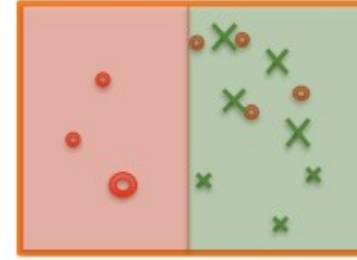
Adaboost plot



Iteration 1



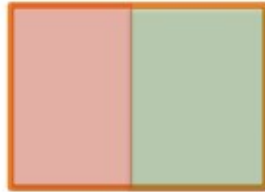
Iteration 2



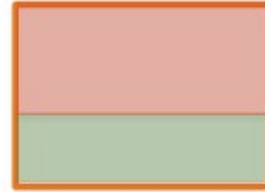
Iteration 3

Final Model

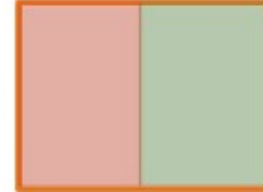
$$f(x) = 0.46$$



$$+ 0.29$$



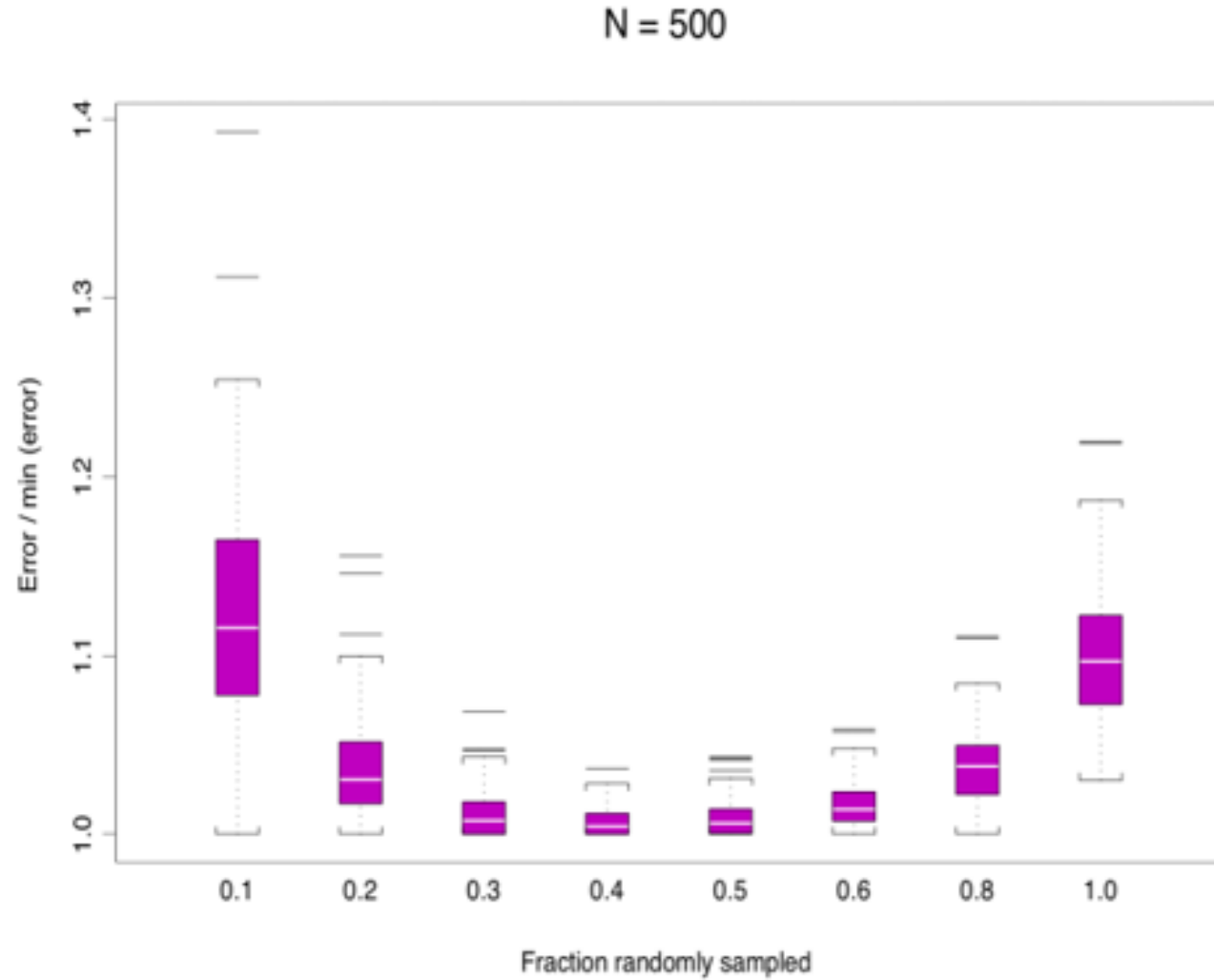
$$+ 0.46$$



Stochastic gradient boosting

- Uses the idea behind bagging
- In each iteration of stochastic boosting, a sample of the training set instead of the full training set.
- Instead of a bootstrap sample (with replacement), the algorithm samples a fraction of the training set.
- This introduced randomness can improve performance
- Paper by Friedman (2002) covers this technique.

Stochastic gradient boosting plot



- Taken from Friedman (2002)

XGBoost

- XGBoost is short for eXtreme Gradient Boosting
- It is a library for high performance gradient boosting models written in C++ with implementations in Python, R and Julia
- Designed for speed – up to 10 times faster than the gbm package
- Has been very popular in recent years and has won a number of machine learning competitions (especially on tabular data)
- Supports regularization
- Can handle missing data

Bagging vs Boosting

- Both ensemble methods get N learners from 1 learner
 - built independently for bagging
 - built sequentially for boosting
- Trees built in boosting are weak learners (sometimes just a stump) while trees in Random Forest have higher complexity
- Few parameters to tune in Random Forest, many more in Boosting (depending on which variation you are using)
- Both combine outputs from N trees

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

References

Friedman, J. H. (2002). "Stochastic gradient boosting". In: *Computational statistics & data analysis* 38.4, pp. 367-378.

James, G., D. Witten, T. Hastie, et al. (2013). *An introduction to statistical learning*. Vol. 112. Springer.

