Trees based methods

Decision Trees, Random Forests and Gradient Boosting

Hicham Zmarrou, PhD

2018-11-20

Aims of this lesson

- Understand what are decision trees (DT), random forests (RF) and gradient boosting (GB), how they works, and how to evaluate a DT a RF or a GB model.

Aims of this lesson

- Understand what are decision trees (DT), random forests (RF) and gradient boosting (GB), how they works, and how to evaluate a DT a RF or a GB model.
- Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) mostly used in classification problems.

Aims of this lesson

- Understand what are decision trees (DT), random forests (RF) and gradient boosting (GB), how they works, and how to evaluate a DT a RF or a GB model.
- Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) mostly used in classification problems.
- It works for both categorical and continuous input and output variables.

Aims of this lesson

- Understand what are decision trees (DT), random forests (RF) and gradient boosting (GB), how they works, and how to evaluate a DT a RF or a GB model.
- Decision tree is a type of supervised learning algorithm (having a pre-defined target variable) mostly used in classification problems.
- It works for both categorical and continuous input and output variables.
- In this technique, we split the population or sample into two or more homogeneous sets (or sub-populations) based on most significant splitter/differentiator in input variables.

Decision trees

A decision tree example

Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?

Figure 1: Playing tennis?

A decision tree example

Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

New data:

D15	Rain	High	Weak	?

Figure 2: Playing tennis?

A decision tree example

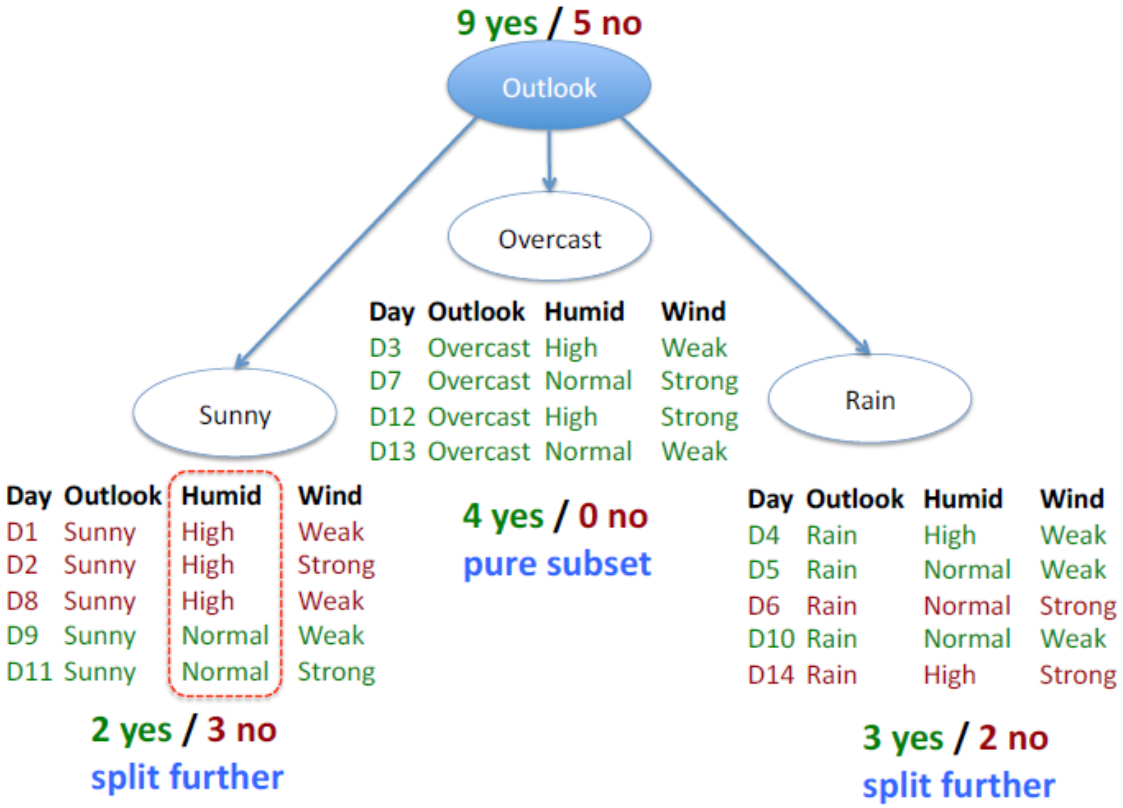


Figure 3: Playing tennis?

A decision tree example

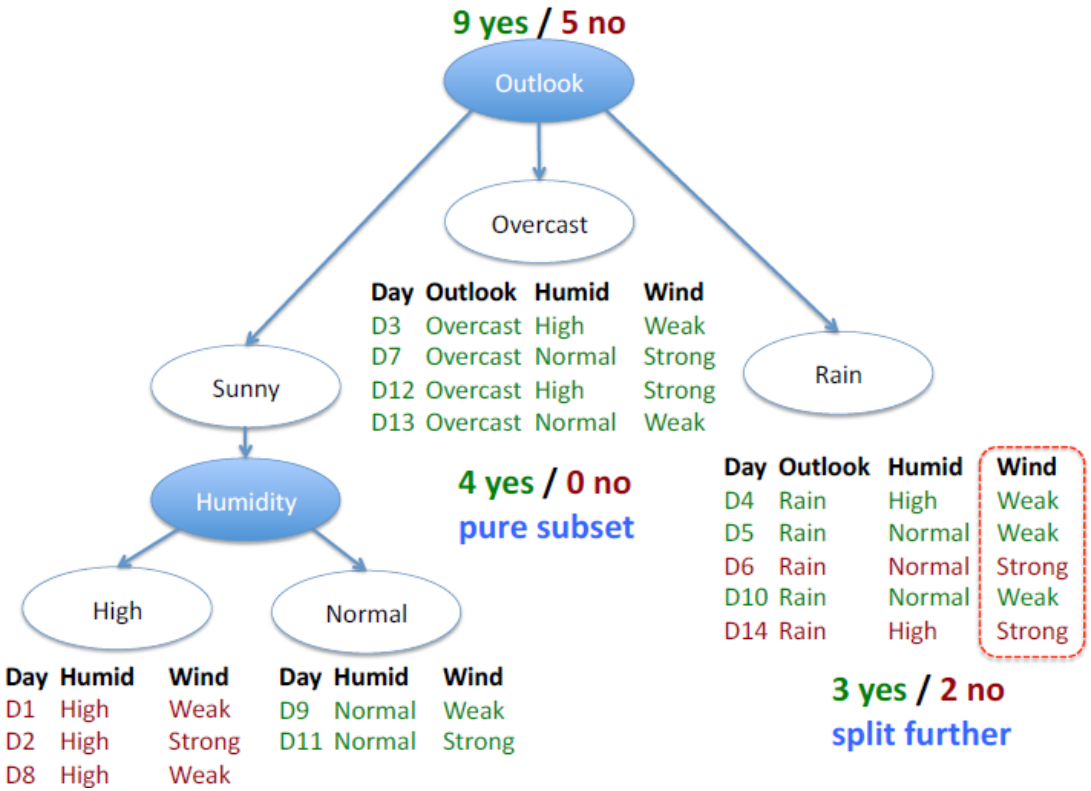


Figure 4: Playing tennis?

A decision tree example

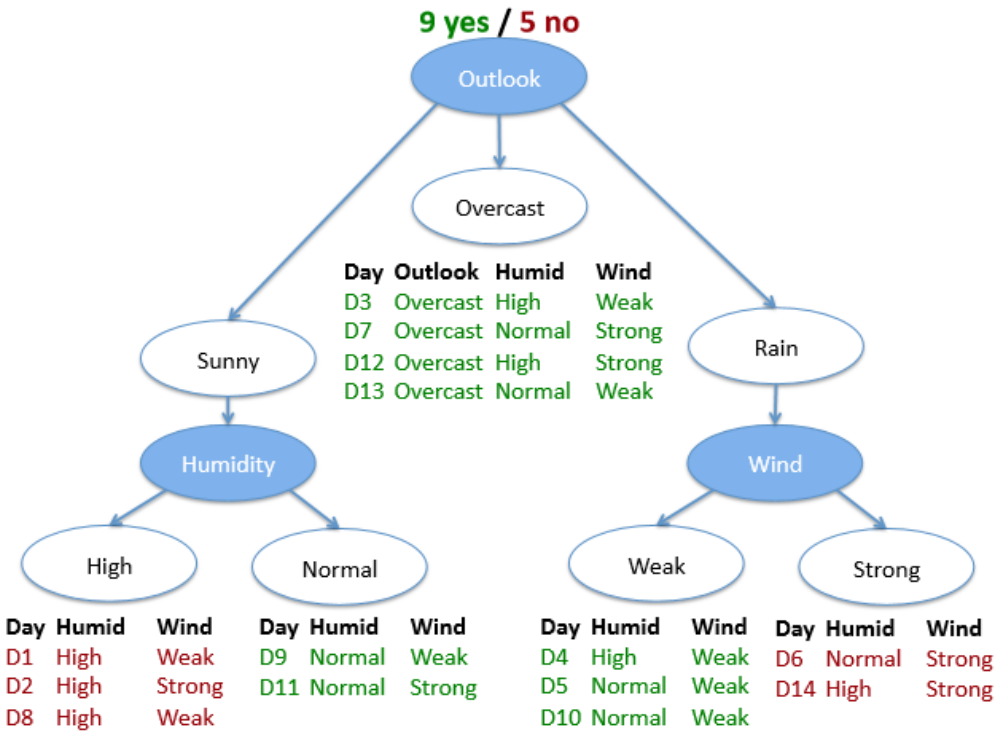


Figure 5: Playing tennis?

A decision tree example

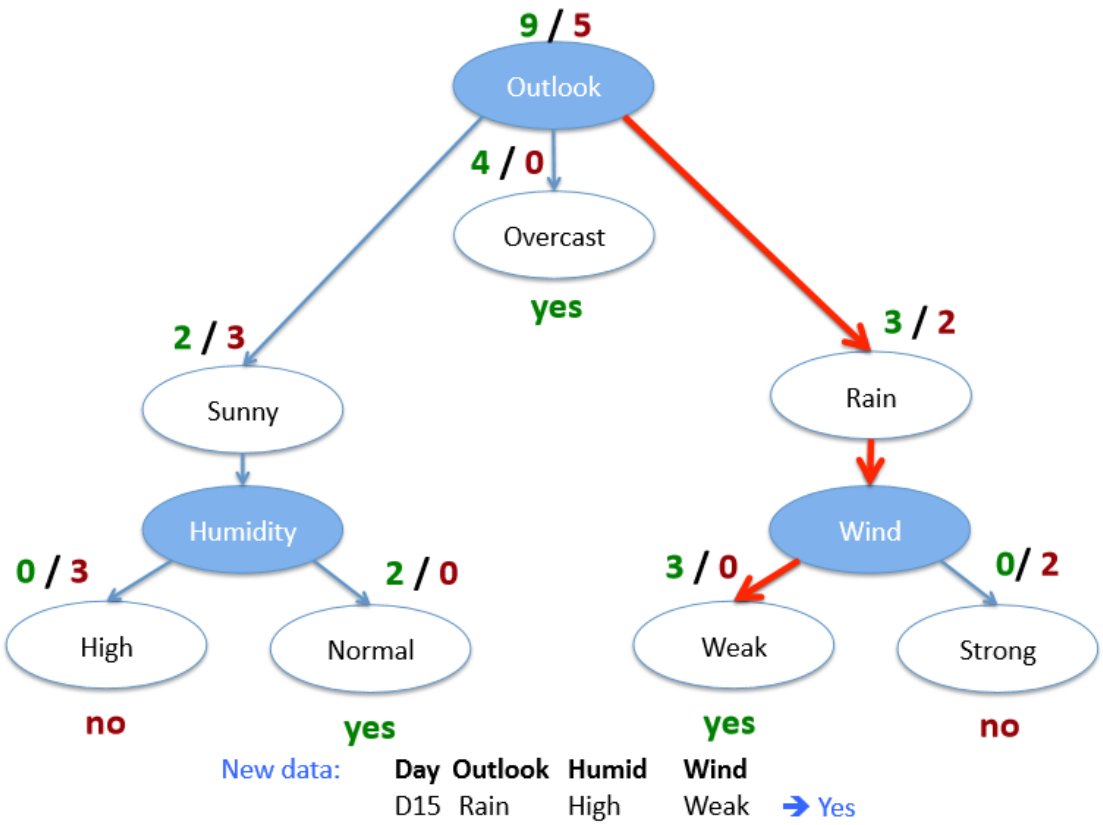


Figure 6: Playing tennis?

Types of decision trees

- **Classification decision tree:** Decision trees which have categorical target variable

Types of decision trees

- **Classification decision tree:** Decision trees which have categorical target variable
  - Models suitable for answering questions: Which category(ies)

Types of decision trees

- **Classification decision tree:** Decision trees which have categorical target variable
  - Models suitable for answering questions: Which category(ies)
- **Regression trees:** decision trees that have continuous target variable

Types of decision trees

- **Classification decision tree:** Decision trees which have categorical target variable
  - Models suitable for answering questions: Which category(ies)
- **Regression trees:** decision trees that have continuous target variable
  - Models suitable for answering questions: How mach, how many

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.

Terminology related to decision trees

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/ Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes where as

Terminology related to decision trees

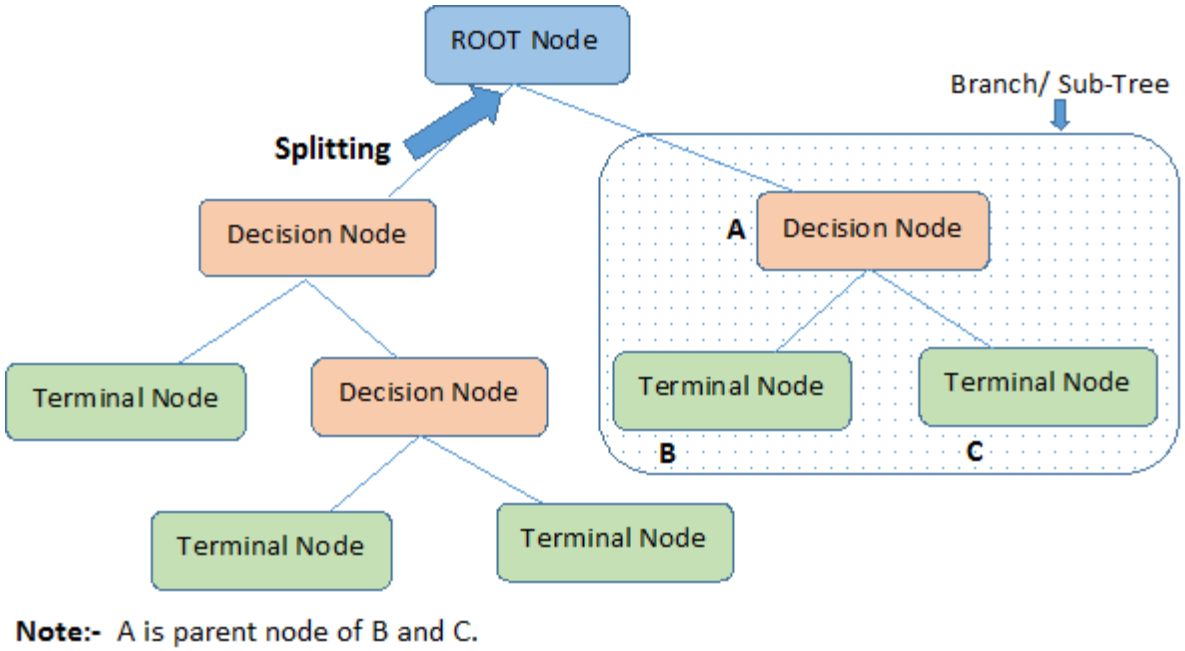


Figure 7: Trees Terminology

Advantages & disadvantages

Advantages

- Easy to Understand:

Disadvantages

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:

Disadvantages

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:
- Less data cleaning required.

Disadvantages

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:
- Less data cleaning required.
- Data type is not a constraint.

Disadvantages

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:
- Less data cleaning required.
- Data type is not a constraint.
- Non parametric method.

Disadvantages

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:
- Less data cleaning required.
- Data type is not a constraint.
- Non parametric method.

Disadvantages

- Over fitting

Advantages & disadvantages

Advantages

- Easy to Understand:
- Useful in data exploration:
- Less data cleaning required.
- Data type is not a constraint.
- Non parametric method.

Disadvantages

- Over fitting
- Not fit for continuous variables

How does a tree decide where to split?

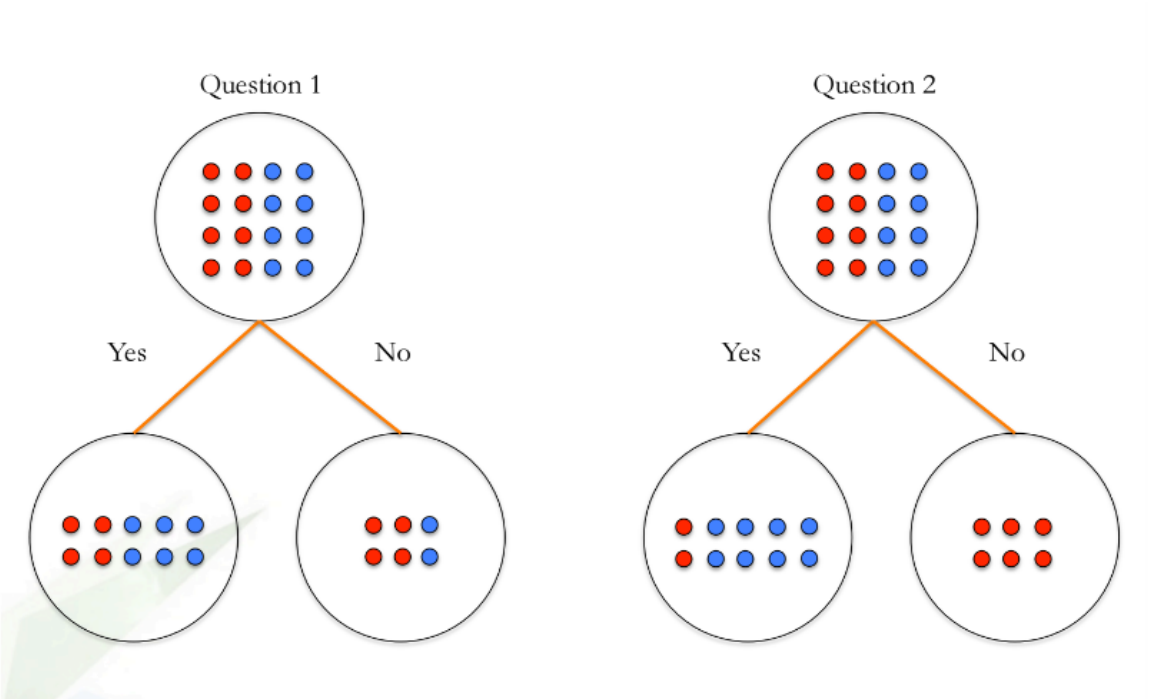


Figure 8: Tree spliting

How does a tree decide where to split?

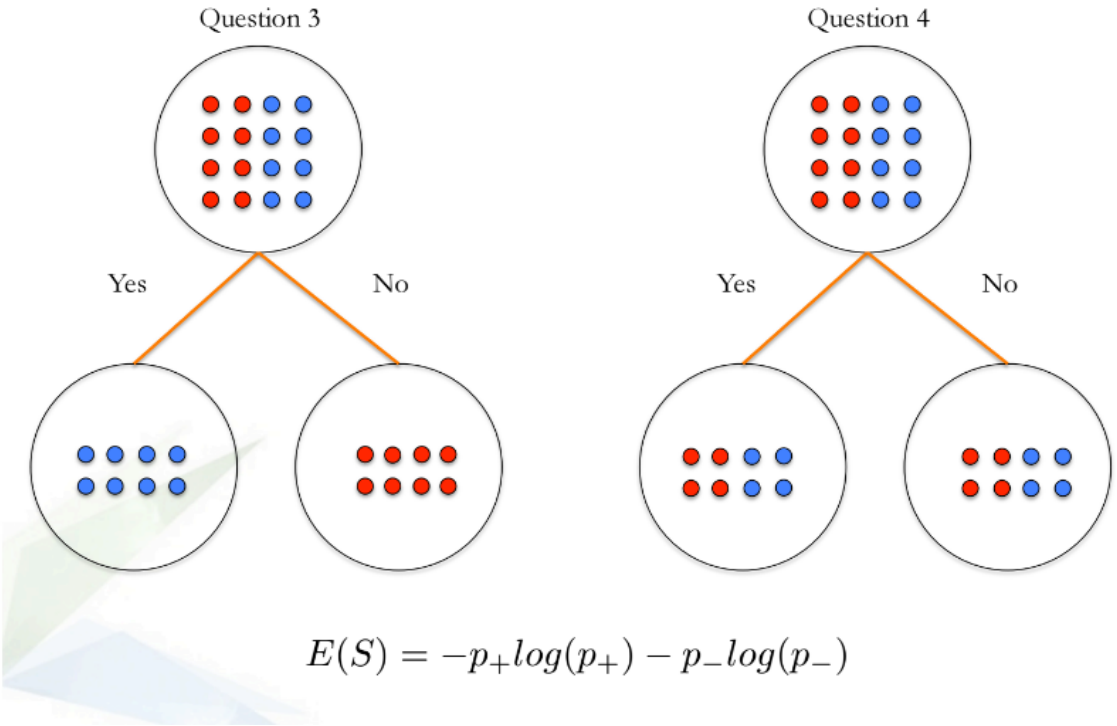


Figure 9: Compute the entropy

How does a tree decide where to split?

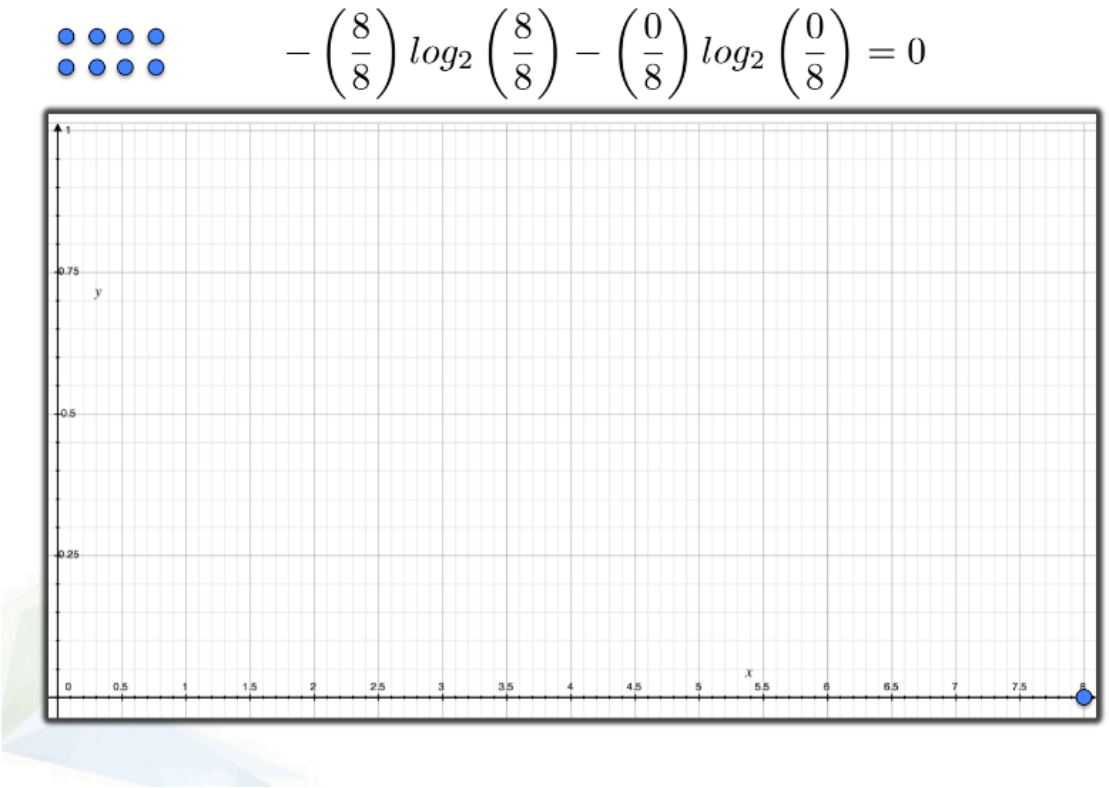


Figure 10: Compute the entropy

How does a tree decide where to split?

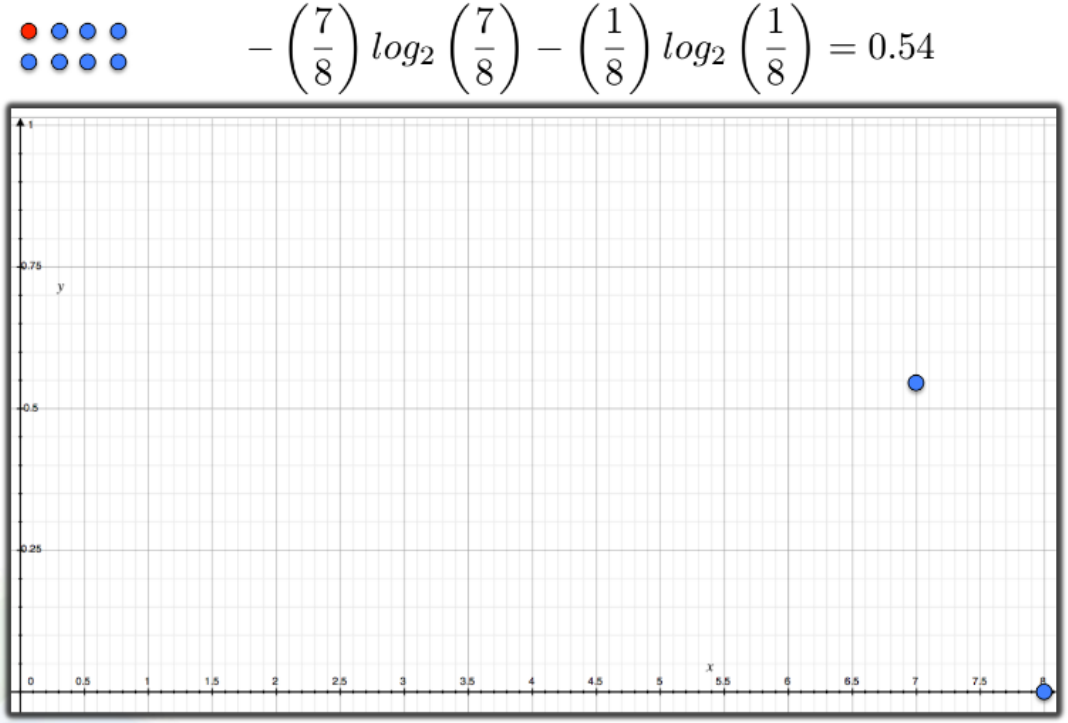


Figure 11: Compute the entropy

How does a tree decide where to split?

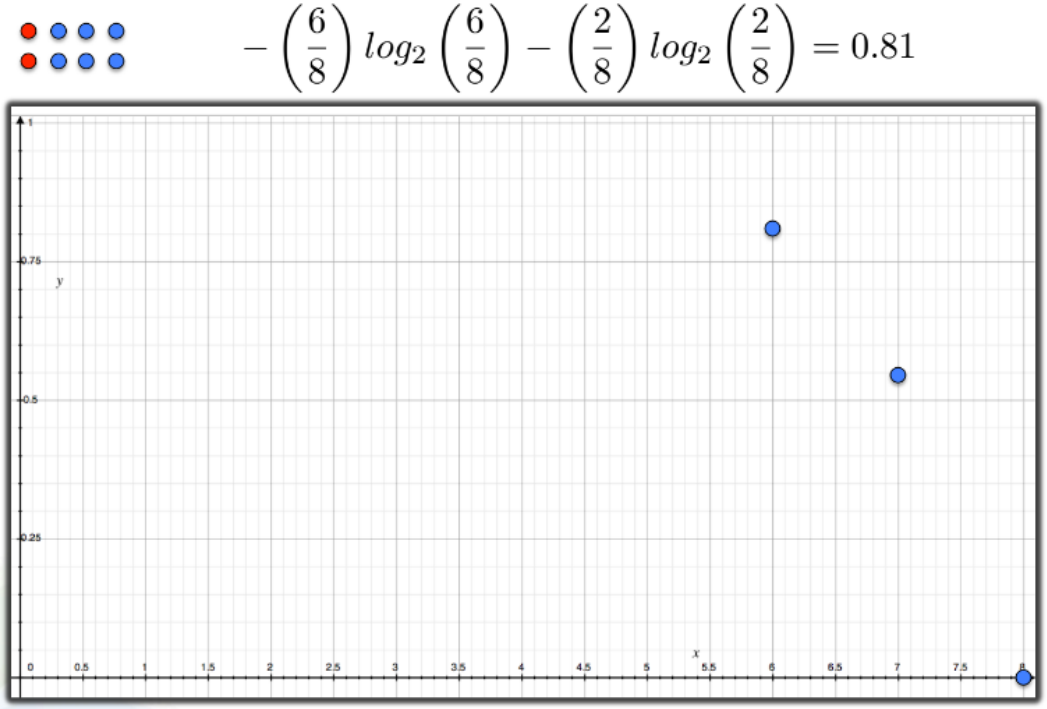


Figure 12: Compute the entropy

How does a tree decide where to split?

$$-\left(\frac{5}{8}\right) log_2\left(\frac{5}{8}\right) - \left(\frac{3}{8}\right) log_2\left(\frac{3}{8}\right) = 0.95$$

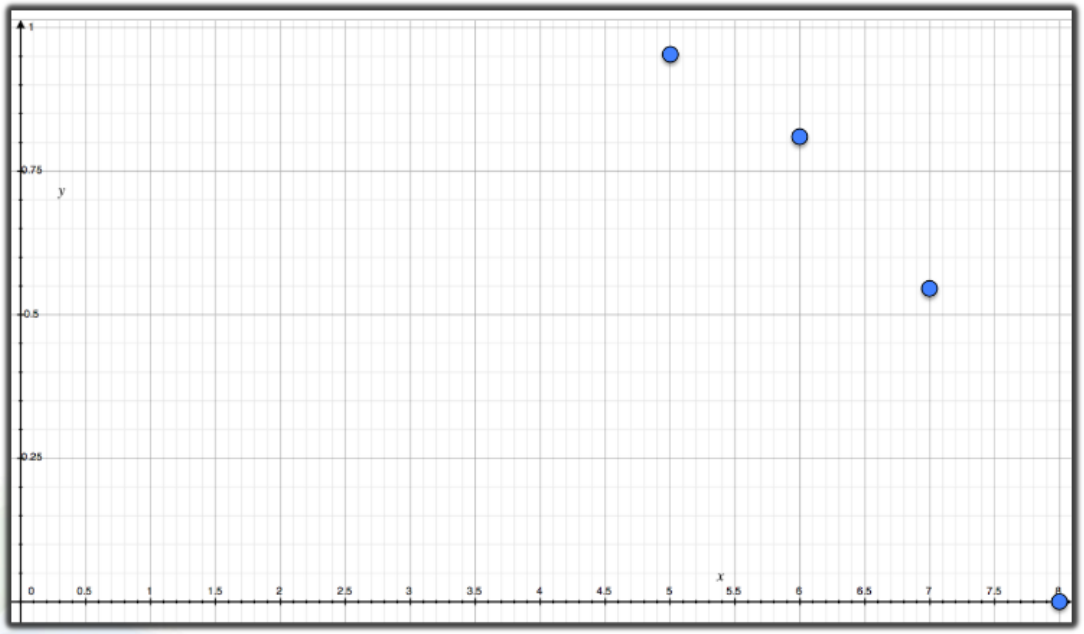


Figure 13: Compute the entropy

How does a tree decide where to split?

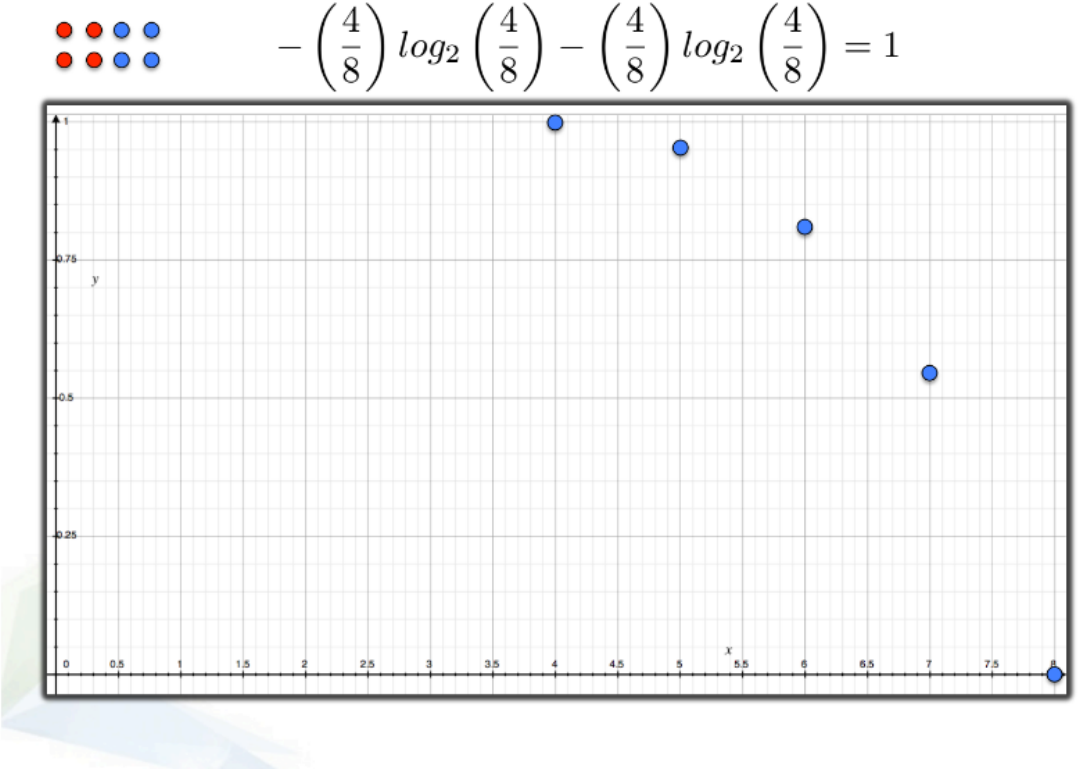


Figure 14: Compute the entropy

How does a tree decide where to split?

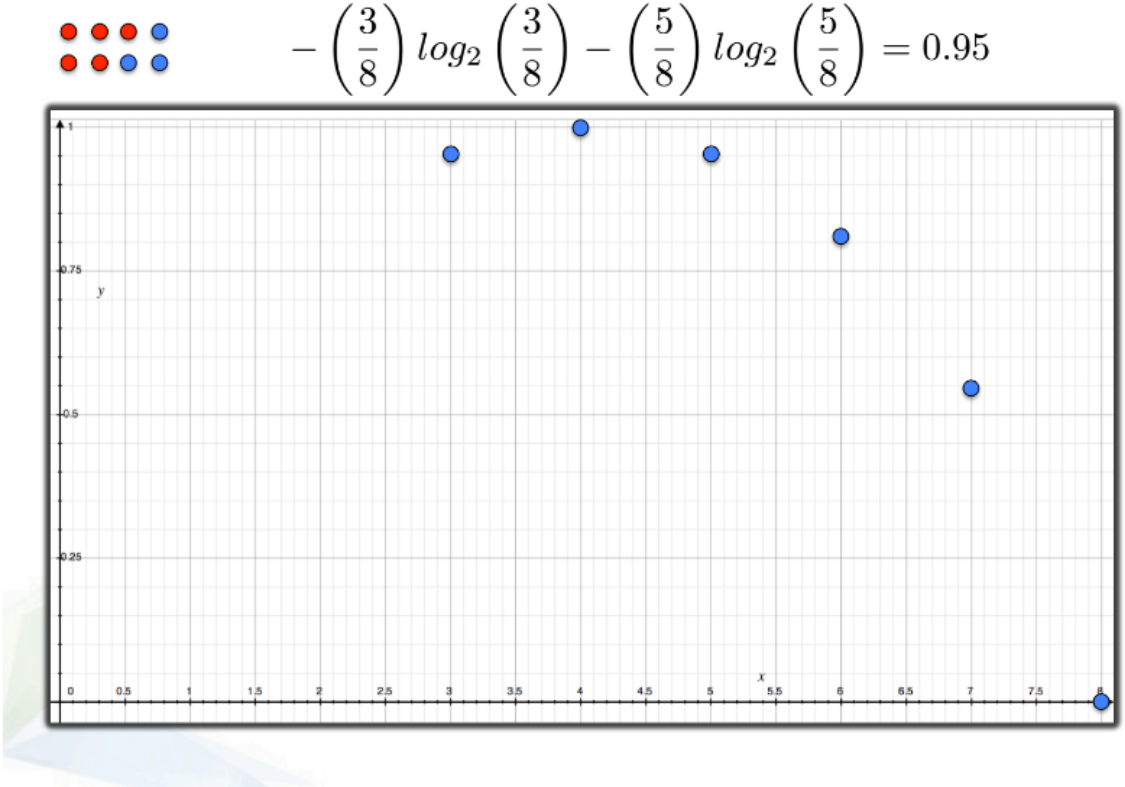


Figure 15: Compute the entropy

How does a tree decide where to split?

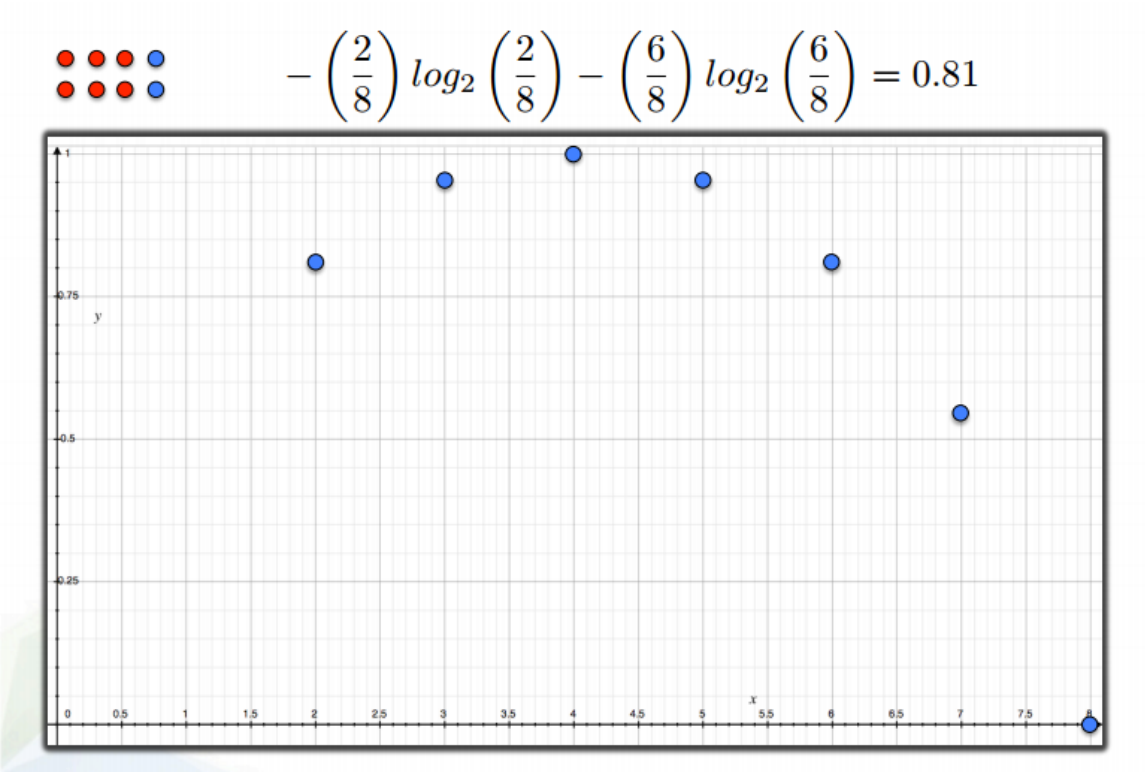


Figure 16: Compute the entropy

How does a tree decide where to split?

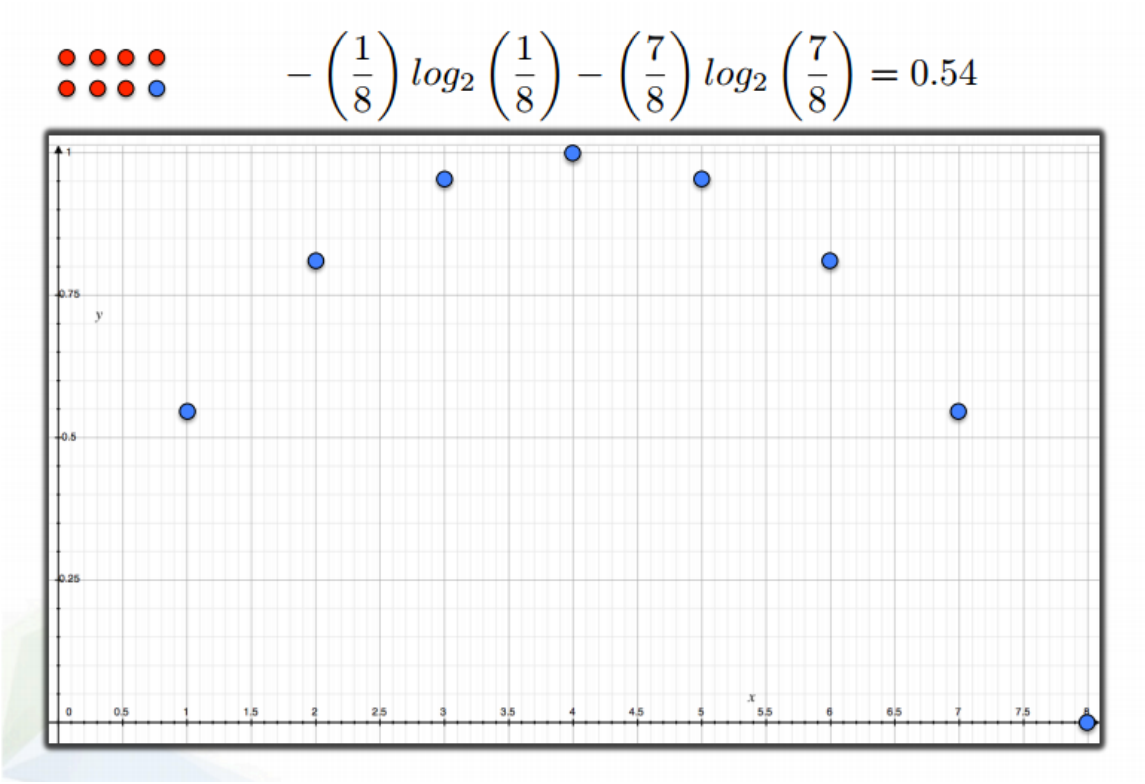


Figure 17: Compute the entropy

How does a tree decide where to split?

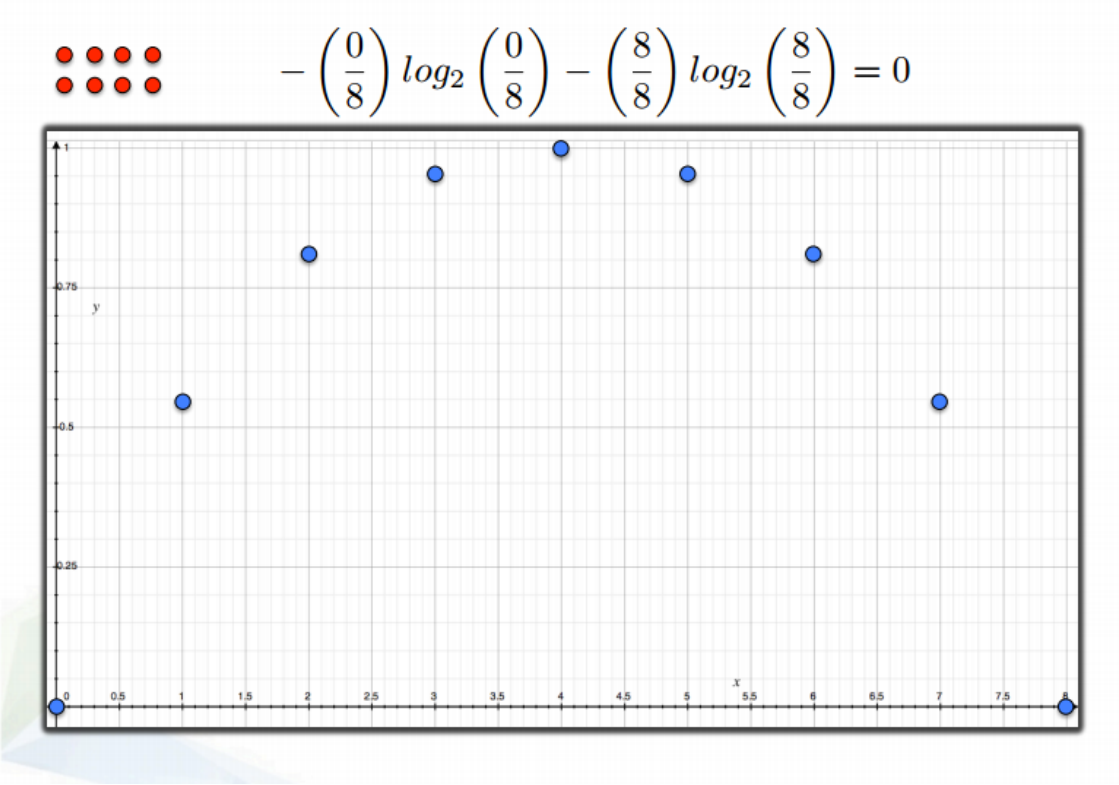


Figure 18: Compute the entropy

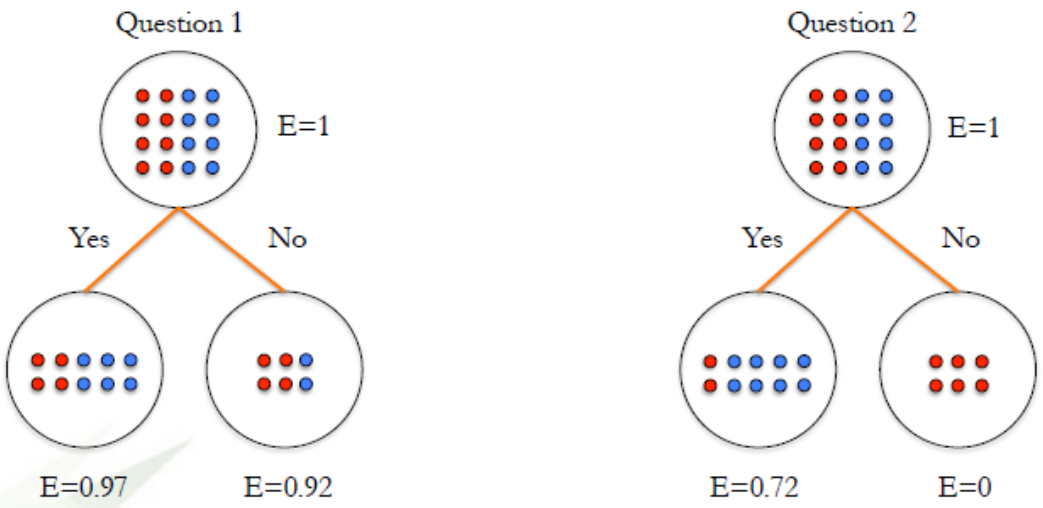
How does a tree decide where to split?

$$y = -\sum_{i=1}^{k} p_i log_k(p_i)$$

$$y = -\underbrace{\left[\left(\frac{1}{10}\right) log_4\left(\frac{1}{10}\right)\right]}_{\text{Red}} - \underbrace{\left[\left(\frac{3}{10}\right) log_4\left(\frac{3}{10}\right)\right]}_{\text{Green}} - \underbrace{\left[\left(\frac{2}{10}\right) log_4\left(\frac{2}{10}\right)\right]}_{\text{Blue}} - \underbrace{\left[\left(\frac{4}{10}\right) log_4\left(\frac{4}{10}\right)\right]}_{\text{Yellow}}$$

Figure 19: Compute the entropy

How does a tree decide where to split?

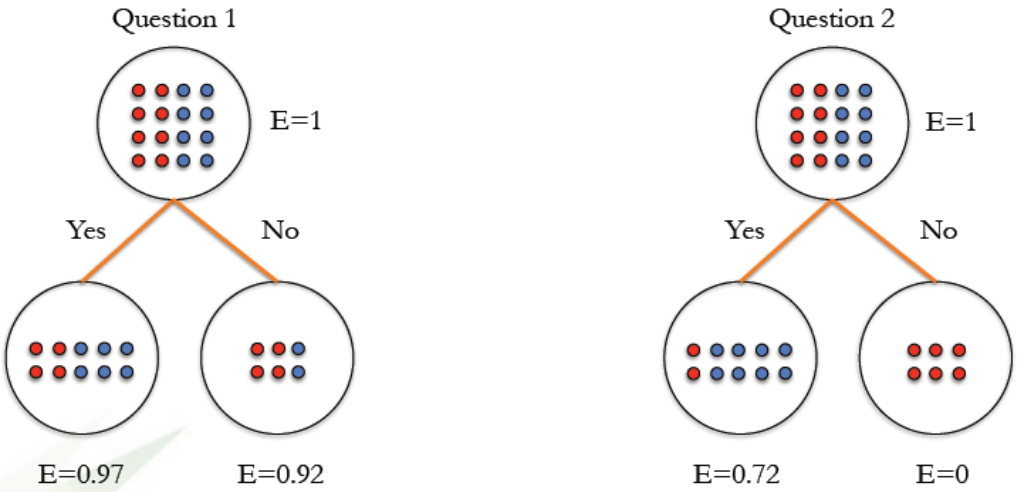


Information Gain

$$G(S,Q) = E(S) - \sum_{i=1}^{k} p_i E(S, Q_i)$$

Figure 20: Compute information gain

How does a tree decide where to split?



$$G(S, Q_1) = 1 - \left(\frac{10}{16}\right) \times 0.97 - \left(\frac{6}{16}\right) \times 0.92$$

$$G(S, Q_1) = 0.049$$

$$G(S, Q_2) = 1 - \left(\frac{10}{16}\right) \times 0.72 - \left(\frac{6}{16}\right) \times 0$$

$$G(S, Q_2) = 0.55$$

Figure 21: Compute information gain

How does a tree decide where to split?

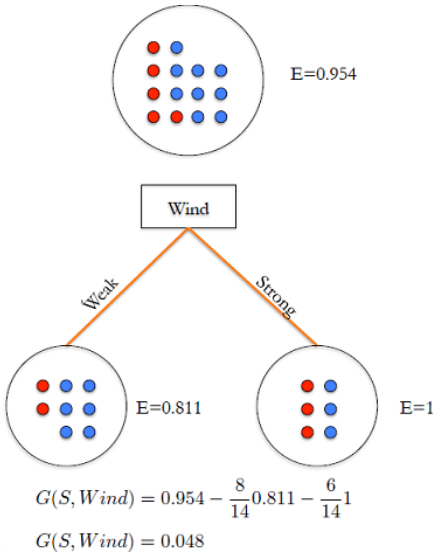


Figure 22: Compute information gain

How does a tree decide where to split?

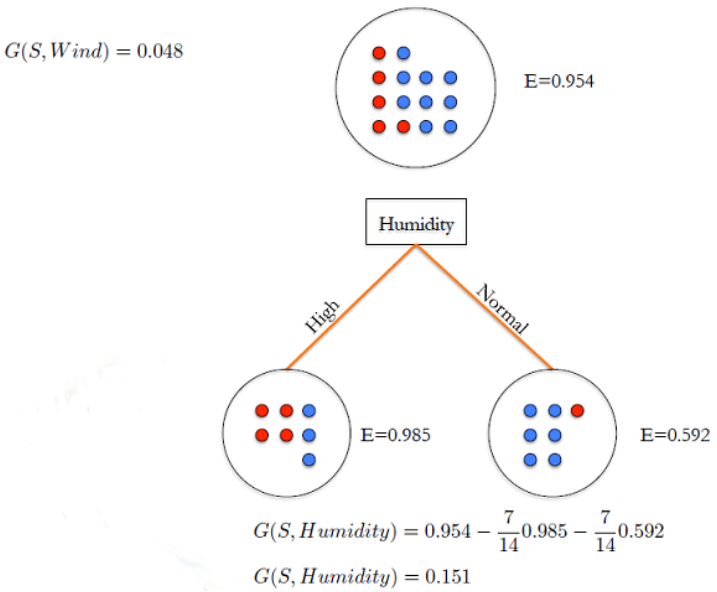


Figure 23: Compute information gain

How does a tree decide where to split?

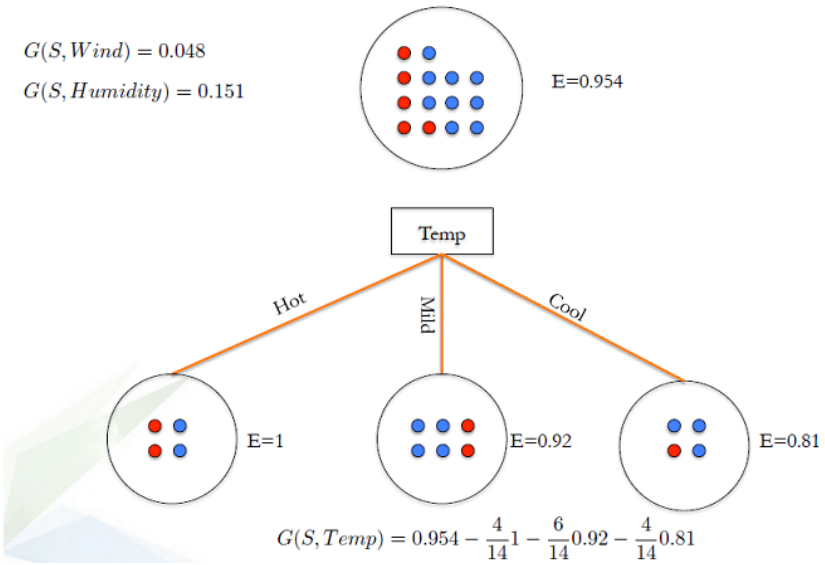


Figure 24: Compute information gain

How does a tree decide where to split?

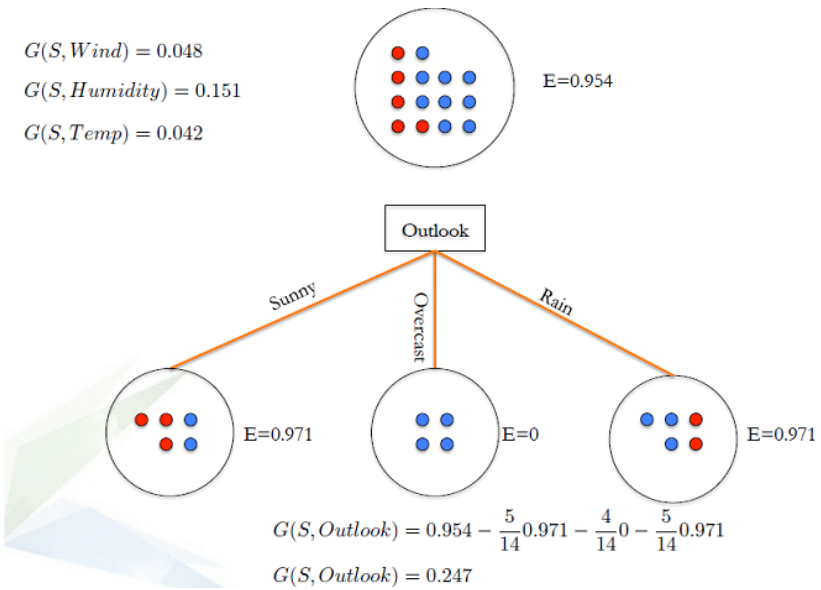


Figure 25: Compute information gain

key parameters of tree modeling

- Overfitting is one of the key challenges faced while modeling decision trees.

key parameters of tree modeling

- Overfitting is one of the key challenges faced while modeling decision trees.
- If no limit set, tree give you 100% accuracy on training set

key parameters of tree modeling

- Overfitting is one of the key challenges faced while modeling decision trees.
- If no limit set, tree give you 100% accuracy on training set
- Preventing overfitting is essential in fitting a decision tree and it can be done in 2 ways:

key parameters of tree modeling

- Overfitting is one of the key challenges faced while modeling decision trees.
- If no limit set, tree give you 100% accuracy on training set
- Preventing overfitting is essential in fitting a decision tree and it can be done in 2 ways:
  - Setting constraints on tree size

key parameters of tree modeling

- Overfitting is one of the key challenges faced while modeling decision trees.
- If no limit set, tree give you 100% accuracy on training set
- Preventing overfitting is essential in fitting a decision tree and it can be done in 2 ways:
  - Setting constraints on tree size
  - Tree pruning

Setting constraints on tree size

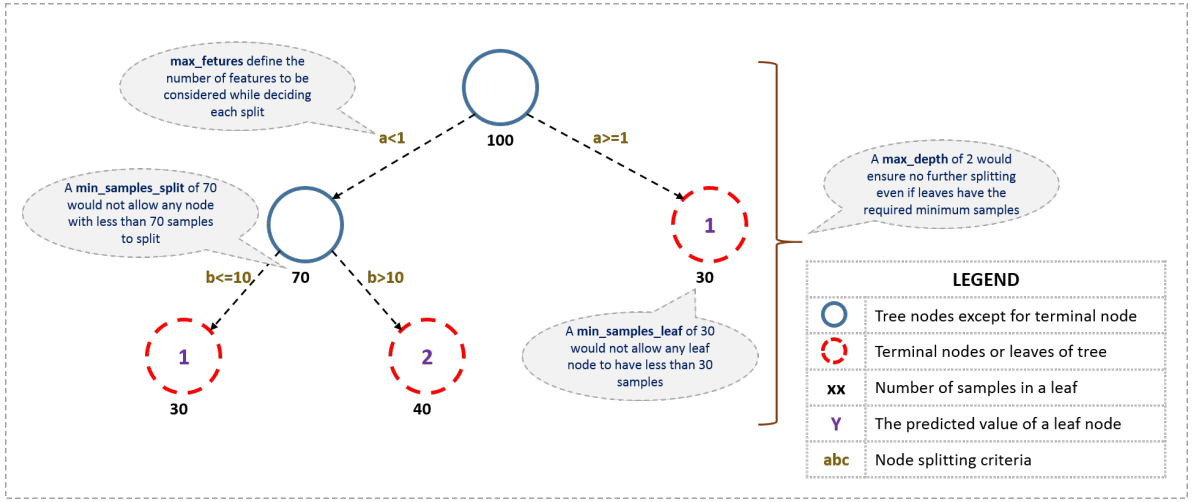


Figure 26: constraints on tree size

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.
3. Maximum depth of tree (vertical depth, `max_depth`)

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.
3. Maximum depth of tree (vertical depth, `max_depth`)
   - Control over-fitting Should be tuned using CV

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.
3. Maximum depth of tree (vertical depth, `max_depth`)
   - Control over-fitting Should be tuned using CV
4. Maximum number of terminal nodes

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.
3. Maximum depth of tree (vertical depth, `max_depth`)
   - Control over-fitting Should be tuned using CV
4. Maximum number of terminal nodes
   - Can be defined in place of `max_depth`. In a binary tree, a depth of 'n' would produce a maximum of $2^{n+1} - 1$ leaves.

Setting constraints on tree size

1. Minimum samples for a node split (`min_samples_split`)
   - Control over-fitting. Should be tuned using CV.
2. Minimum samples for a terminal node (leaf)
   - Control over-fitting similar to min_samples_split.
3. Maximum depth of tree (vertical depth, `max_depth`)
   - Control over-fitting Should be tuned using CV
4. Maximum number of terminal nodes
   - Can be defined in place of `max_depth`. In a binary tree, a depth of 'n' would produce a maximum of $2^{n+1} - 1$ leaves.
5. Maximum features to consider for split

Tree pruning

1. Make the decision tree to a large depth.

Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is $+10$ and keep both leaves.

Tree pruning

1. Make the decision tree to a large depth.
2. Start at the bottom and start removing leaves which are giving us negative IG when compared from the top.

Suppose a split is giving us a gain of say -10 (loss of 10) and then the next split on that gives us a gain of 20. A simple decision tree will stop at step 1 but in pruning, we will see that the overall gain is $+10$ and keep both leaves.

Are tree based models better than logistic models?

- If the relationship between feature and label is well approximated by a linear model, linear regression will outperform tree based model.

Are tree based models better than logistic models?

- If the relationship between feature and label is well approximated by a linear model, linear regression will outperform tree based model.
- If there is a high non-linearity andcomplex relationship between feature and label tree model will outperform a classical regression method.

Are tree based models better than logistic models?

- If the relationship between feature and label is well approximated by a linear model, linear regression will outperform tree based model.
- If there is a high non-linearity andcomplex relationship between feature and label tree model will outperform a classical regression method.
- If you need to build a model which is easy to explain to people, a decision tree model will always do better than a linear model. Decision tree models are even simpler to interpret than linear regression!

Working with decision trees in R

Go to the notebook

Evaluate model Performance

Predicting class labels for test data

```
predict(model, test_dataset)
```

```
predict(model, test_dataset, type = ---)
```

```
class_prediction <- predict(object  = restaurant_model, #model object
                            newdata = restaurant_test,  #test dataset
                            type = "class") #return classification labels
```

Figure 27: Model evaluation

Evaluation metrics for binary classification

- Accuracy

Evaluation metrics for binary classification

- Accuracy
- Confusion matrix

Evaluation metrics for binary classification

- Accuracy
- Confusion matrix
- Log-loss

Evaluation metrics for binary classification

- Accuracy
- Confusion matrix
- Log-loss
- AUC

Accuracy

$$accuracy = \frac{\#ofcorrect\ \ prediction}{\#\ of\ \ total\ \ data\ \ points}$$

Confusion matrix

Predicted class \ True class	Positive	Negative	Measures
Positive	True positive TP	False positive FP	Positive predictive value (PPV) $\frac{TP}{TP+FP}$
Negative	False negative FN	True negative TN	Negative predictive value (NPV) $\frac{TN}{FN+TN}$
Measures	Sensitivity $\frac{TP}{TP+FN}$	Specificity $\frac{TN}{FP+TN}$	Accuracy $\frac{TP+TN}{TP+FP+FN+TN}$

Figure 28: Confusion matrix

Confusion matrix

true positives (TP): These are cases in which we predicted yes (Defalut), and they are Defaulted.

true negatives (TN): We predicted will NOT default, and they will NOT default.

false positives (FP): We predicted Defaulted, but they don't actually Defaulted. (Also known as a "Type I error.")

false negatives (FN): We predicted NOT Defaulted, but they actually do NOT Defaulted. (Also known as a "Type II error.")

Confusion matrix

We fit a simple tree model `default ~ balance+income+student` to the Default dataset from the ISLR package

```
library(ISLR)
library(rpart)
data(Default)

## 75% of the sample size
smp_size <- floor(0.75 * nrow(Default))

## set the seed to make your partition reproducible
set.seed(123)
train_ind <- sample(seq_len(nrow(Default)), size = smp_size)
# make a training and test set
train_df <- Default[train_ind, ]
test_df <-  Default[-train_ind, ]
# fit a tree to the default data
model = rpart(default ~ balance+income+student, data = train_df, method = "class")
class_prediction <- predict(object  = model, #model object
                            newdata = test_df,  #test dataset
                            type = "class") #return classification labels
```

Confusion matrix

Let calculate some metrics such as accuracy the hard way.

```
mean(class_prediction == test_df$default)
```

```
[1] 0.9736
```

We may use the `table()` and `confusionMatrix()` from the `caret` library to quickly obtain many more metrics.

```
library(caret)
test_tab = table(predicted = class_prediction , actual = test_df$default)
con_mat = confusionMatrix(test_tab, positive = "Yes")
c(con_mat$overall["Accuracy"],
  con_mat$byClass["Sensitivity"],
  con_mat$byClass["Specificity"])
#>    Accuracy Sensitivity Specificity
#>   0.9722000   0.2738095   0.9964818
```

Evaluation metrics for binary classification

Please complete "compute_confusion_matrix_lab.Rmd"

Ensemble methods - Random Forests

What are ensemble methods in tree based modeling ?

en-sem-ble

A unit or group of complementary parts that contribute to a single effect, especially:

- A coordinated outfit or costume.

What are ensemble methods in tree based modeling ?

en-sem-ble

A unit or group of complementary parts that contribute to a single effect, especially:

- A coordinated outfit or costume.
- A coordinated set of furniture.

What are ensemble methods in tree based modeling ?

en-sem-ble

A unit or group of complementary parts that contribute to a single effect, especially:

- A coordinated outfit or costume.
- A coordinated set of furniture.
- A group of musicians, singers, dancers, or actors who perform together

Bootstrapping

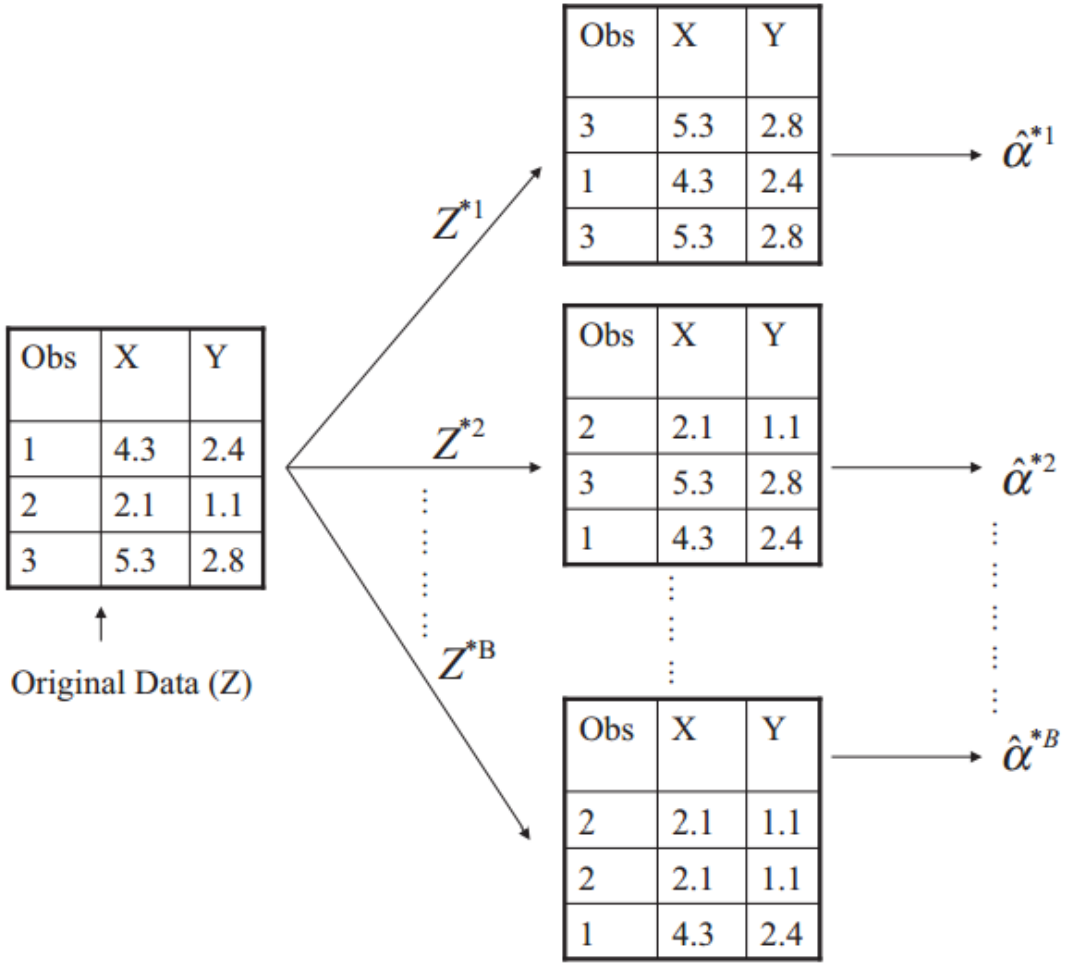


Figure 30: Bootstraping

What is bagging? how does it work?

1. **Create multiple data sets through bootstrapping:** Sampling is done with replacement on the original data and new datasets are formed. The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting

What is bagging? how does it work?

1. **Create multiple data sets through bootstrapping:** Sampling is done with replacement on the original data and new datasets are formed. The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
2. **Build multiple classifiers:** Classifiers are built on each data set. Generally the same classifier is modeled on each data set and predictions are made.

What is bagging? how does it work?

1. **Create multiple data sets through bootstrapping:** Sampling is done with replacement on the original data and new datasets are formed. The new data sets can have a fraction of the columns as well as rows, which are generally hyper-parameters in a bagging model Taking row and column fractions less than 1 helps in making robust models, less prone to overfitting
2. **Build multiple classifiers:** Classifiers are built on each data set. Generally the same classifier is modeled on each data set and predictions are made.
3. **Combine classifiers:** The predictions of all the classifiers are combined using a mean, median or mode value depending on the problem at hand. The combined values are generally more robust than a single model.

What is bagging? how does it work?

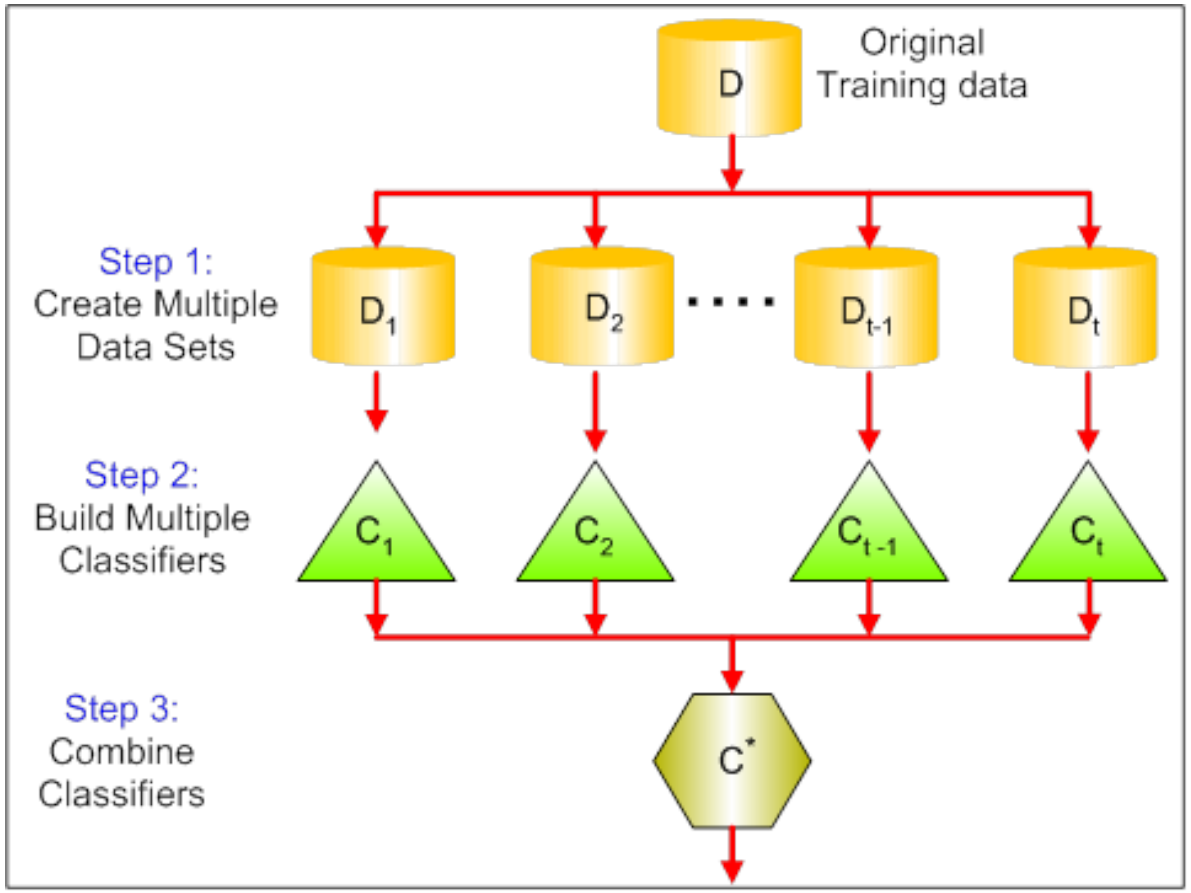


Figure 31: Bagging

Out-of-Bag error estimation

- It turns out that there is a very straightforward way to estimate the test error of a bagged model.

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.

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.

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 *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.

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 *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.

Bagging the heart data

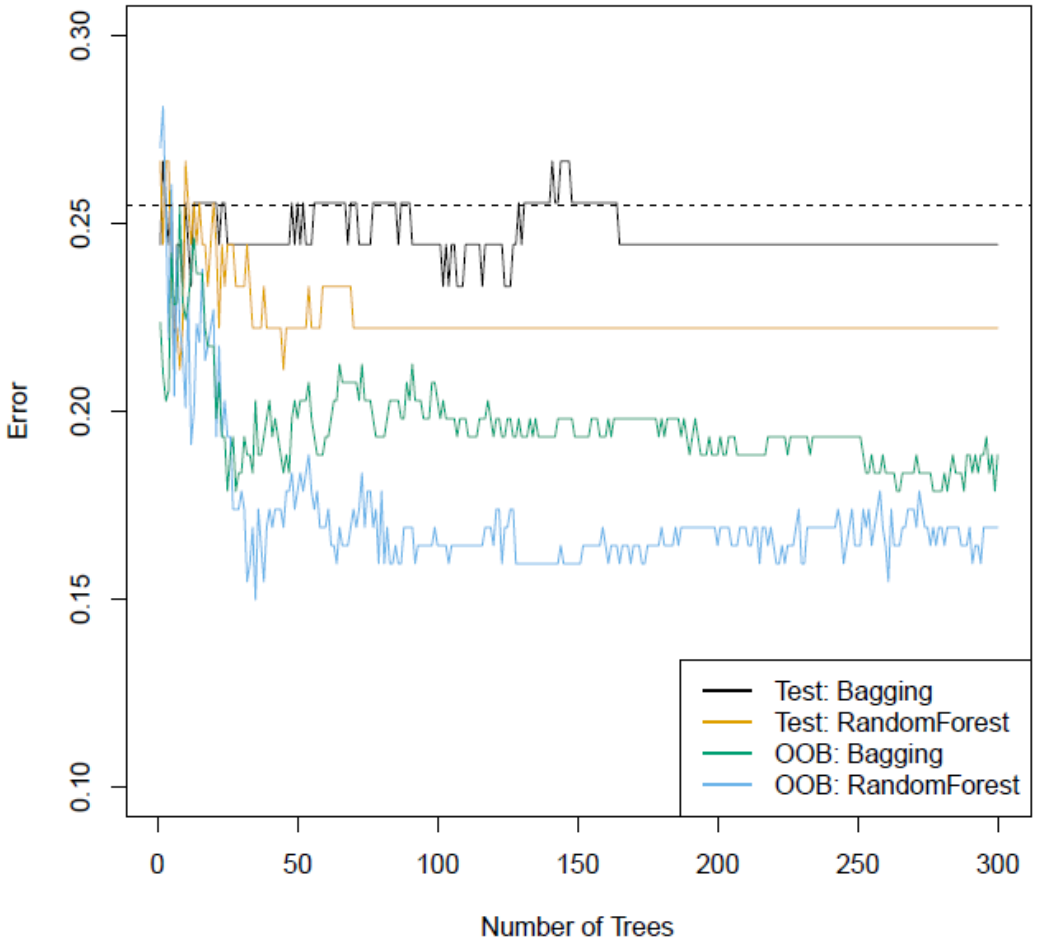


Figure 32: Bagging

What is Random Forest ? How does it work?

1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.

What is Random Forest ? How does it work?

1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m < M$ is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.

What is Random Forest ? How does it work?

1. Assume number of cases in the training set is N. Then, sample of these N cases is taken at random but with replacement. This sample will be the training set for growing the tree.
2. If there are M input variables, a number $m < M$ is specified such that at each node, m variables are selected at random out of the M. The best split on these m is used to split the node. The value of m is held constant while we grow the forest.
3. Each tree is grown to the largest extent possible and there is no pruning. Predict new data by aggregating the predictions of the ntree trees (i.e., majority votes for classification, average for regression).

What is Random Forest ? How does it work?

Advantages of Random Forest

1. Random forest can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.

Advantages of Random Forest

1. Random forest can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.
2. Random forest can handle large data set with higher dimensionality. Further, RF models output Importance of variable, which can be a very usefull feature (on some random data set).

Advantages of Random Forest

1. Random forest can solve both type of problems i.e. classification and regression and does a decent estimation at both fronts.
2. Random forest can handle large data set with higher dimensionality. Further, RF models output Importance of variable, which can be a very usefull feature (on some random data set).
3. Computation of the out-of-bag error estimate removes the need for a set aside test set.

Bagged trees vs. Random Forest

Bagged trees vs. Random Forest

What is the main difference between bagged trees and the Random Forest algorithm?

Answer the question 50 XP

Possible Answers

- In Random Forest, the decision trees are trained on a random subset of the rows, but in bagging, they use all the rows. press 1
- In Random Forest, only a subset of features are selected at random at each split in a decision tree. In bagging, all features are used. press 2
- In Random Forest, there is randomness. In bagging, there is no randomness. press 3

Submit Answer

Take Hint (-15 XP)

Random forrest 'R' implementation (randomForrest)

```
library(randomForest)
```

```
?randomForest
```

randomForest {randomForest} R Documentation

Classification and Regression with Random Forest

Description

`randomForest` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

Usage

```
## S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
## S3 method for class 'randomForest'
print(x, ...)
```

Random forrest 'R' implementation (`randomForrest`)

```
library(randomForest)
```

```
?randomForest
```

randomForest {randomForest} R Documentation

Classification and Regression with Random Forest

Description

`randomForest` implements Breiman's random forest algorithm (based on Breiman and Cutler's original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points.

Usage

```
## S3 method for class 'formula'
randomForest(formula, data=NULL, ..., subset, na.action=na.fail)
## Default S3 method:
randomForest(x, y=NULL,  xtest=NULL, ytest=NULL, ntree=500,
             mtry=if (!is.null(y) && !is.factor(y))
             max(floor(ncol(x)/3), 1) else floor(sqrt(ncol(x))),
             replace=TRUE, classwt=NULL, cutoff, strata,
             sampsize = if (replace) nrow(x) else ceiling(.632*nrow(x)),
             nodesize = if (!is.null(y) && !is.factor(y)) 5 else 1,
             maxnodes = NULL,
             importance=FALSE, localImp=FALSE, nPerm=1,
             proximity, oob.prox=proximity,
             norm.votes=TRUE, do.trace=FALSE,
             keep.forest=!is.null(y) && is.null(xtest), corr.bias=FALSE,
             keep.inbag=FALSE, ...)
## S3 method for class 'randomForest'
print(x, ...)
```

Please complete "intro_randomForest01.Rmd"

Understanding the Random Forest model output

```r
# Print  model output
print(credit_model)
```

```
Call:
 randomForest(formula = default ~ ., data = credit_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 30.53%
Confusion matrix:
    1  2 class.error
1 457 68   0.1295238
2 161 64   0.7155556
```

Out-of-bag error matrix

```r
# Grab OOB error matrix & take a look

err <- credit_model$err.rate
head(err)
```

```
           OOB         1         2
[1,] 0.4075472 0.2842105 0.7200000
[2,] 0.3863636 0.2619808 0.6929134
[3,] 0.4082734 0.2748092 0.7300613
[4,] 0.3754045 0.2397260 0.7055556
[5,] 0.3588589 0.2271762 0.6769231
[6,] 0.3748191 0.2438525 0.6896552
```

Out-of-bag error estimate

```
tail(err, n=1)
```

```
         OOB         1         2
[500,] 0.3053333 0.1295238 0.7155556
```

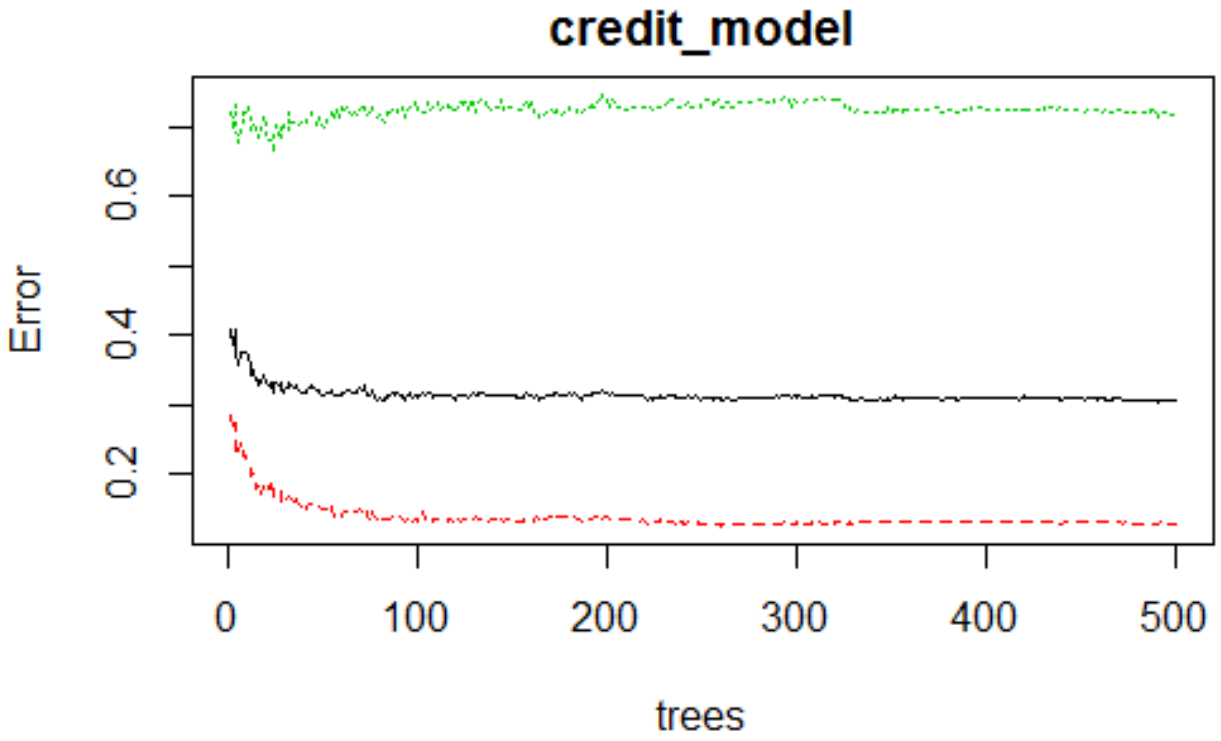
Hide

```
# Print  model output
print(credit_model)
```

```
Call:
 randomForest(formula = default ~ ., data = credit_train)
               Type of random forest: classification
                     Number of trees: 500
No. of variables tried at each split: 2

        OOB estimate of  error rate: 30.53%
Confusion matrix:
    1  2 class.error
1 457 68   0.1295238
2 161 64   0.7155556
```

Out-of-bag error estimate



Please complete "evaluate_out_of_bag_error.Rmd"

OOB error vs. test set error

Advantages

- Can evaluate your model without a separate test set

Disadvantages

OOB error vs. test set error

Advantages

- Can evaluate your model without a separate test set
- Compute automatically by the randomForest function

Disadvantages

OOB error vs. test set error

Advantages

- Can evaluate your model without a separate test set
- Compute automatically by the randomForest function

Disadvantages

- OOB error only estimate error (not other performance metrics such as AUC, log-loss, etc)

OOB error vs. test set error

Advantages

- Can evaluate your model without a separate test set
- Compute automatically by the randomForest function

Disadvantages

- OOB error only estimate error (not other performance metrics such as AUC, log-loss, etc)
- can't compare RF performance to other types of models

Advantage of OOB error

Advantage of OOB error

What is the main advantage of using OOB error instead of validation or test error?

Answer the question 50 XP

Possible Answers

- Tuning the model hyperparameters using OOB error will lead to a better model. press 1
- If you evaluate your model using OOB error, then you don't need to create a separate test set. press 2
- OOB error is more accurate than test set error. press 3

Submit Answer

Take Hint (-15 XP)

Tuning a Random Forest model

- *ntree*: number of trees

Tuning a Random Forest model

- *ntree*: number of trees
- *mtry*: number of predictors sampled for spliting at each node.

Tuning a Random Forest model

- *ntree*: number of trees
- *mtry*: number of predictors sampled for spliting at each node.
- *sampsize*: number of samples to train on

Tuning a Random Forest model

- *ntree*: number of trees
- *mtry*: number of predictors sampled for spliting at each node.
- *sampsize*: number of samples to train on
- *nodesize*: minimum size (number of samples) of the terminal nodes

Tuning a Random Forest model

- *ntree*: number of trees
- *mtry*: number of predictors sampled for spliting at each node.
- *sampsize*: number of samples to train on
- *nodesize*: minimum size (number of samples) of the terminal nodes
- *maxnodes*: maximum number of terminal nodes

Tuning mtry with tuneRF()

```
# Execute the tuning process

set.seed(1)
res <- tuneRF(x = train_predictor_df,
              y = train_response_vector,
              ntreeTry = 500)
```

```
# Look at results
print(res)
```

Results table:

```
      mtry OOBError
2.00B    2   0.2475
4.00B    4   0.2475
8.00B    8   0.2425
```

Please complete "tuning_random_forest.Rmd"

Ensemble methods - Gradient Boosting

Predict a person's age example

We want to predict a person's age based on whether they play video games, enjoy gardening, and their preference on wearing hats. Our objective is to minimize squared error. We have these nine training samples to build our model.

PersonID	Age	LikesGardening	PlaysVideoGames	LikesHats
1	13	FALSE	TRUE	TRUE
2	14	FALSE	TRUE	FALSE
3	15	FALSE	TRUE	FALSE
4	25	TRUE	TRUE	TRUE
5	35	FALSE	TRUE	TRUE
6	49	TRUE	FALSE	FALSE
7	68	TRUE	TRUE	TRUE
8	71	TRUE	FALSE	FALSE
9	73	TRUE	FALSE	TRUE

Predict a person's age example

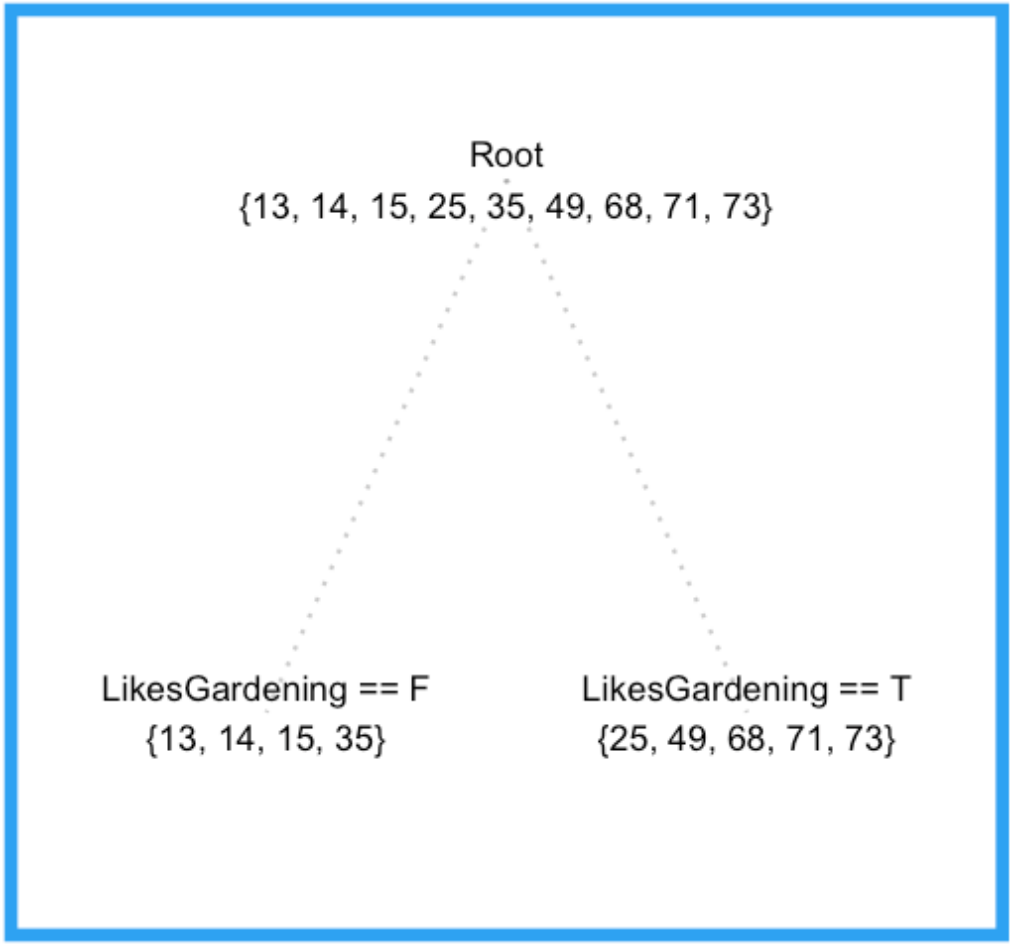
Intuitively, we might expect - The people who like gardening are probably older - The people who like video games are probably younger - `LikesHats` is probably just random noise

We can do a quick and dirty inspection of the data to check these assumptions:

Feature	FALSE	TRUE
LikesGardening	{13, 14, 15, 35}	{25, 49, 68, 71, 73}
PlaysVideoGames	{49, 71, 73}	{13, 14, 15, 25, 35, 68}
LikesHats	{14, 15, 49, 71}	{13, 25, 35, 68, 73}

Now let's model the data with a regression tree. To start, we'll require that terminal nodes have at least three samples. With this in mind, the regression tree will make its first and last split on `LikesGardening`.

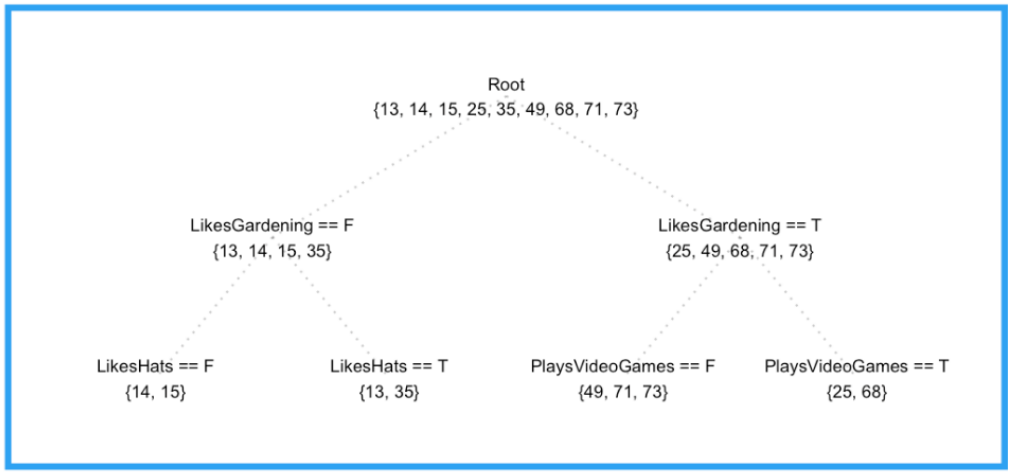
Predict a person's age example



This is nice, but it's missing valuable information from the feature LikesVideoGames. Let's try letting terminal nodes have 2 samples.

Example

Overfit Tree



Here we pick up some information from `PlaysVideoGames` but we also pick up information from `LikesHats` - a good indication that we're overfitting and our tree is splitting random noise.

Predict a person's age example

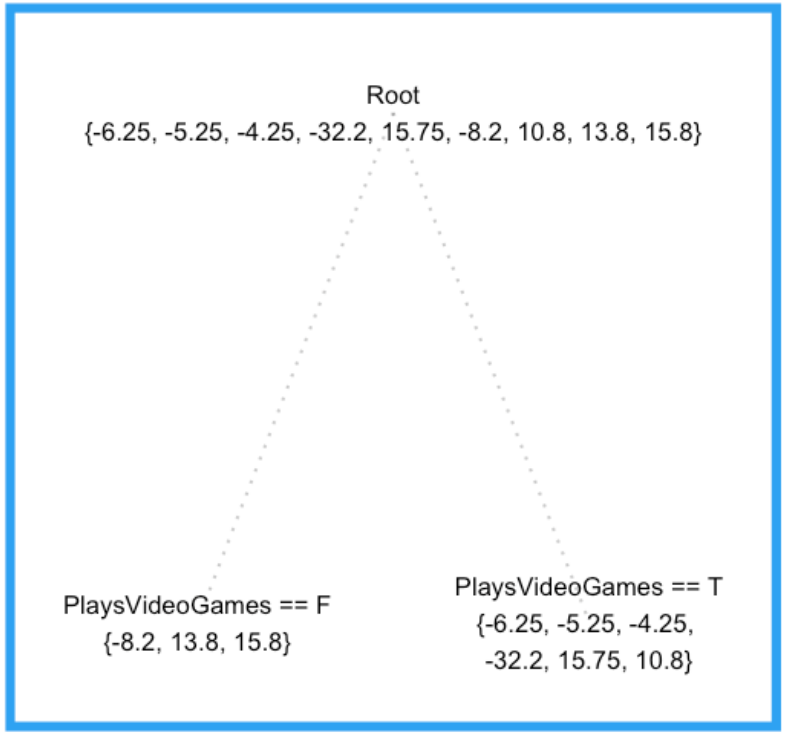
Suppose we measure the training errors from our first tree.

PersonID	Age	Tree1 Prediction	Tree1 Residual
1	13	19.25	-6.25
2	14	19.25	-5.25
3	15	19.25	-4.25
4	25	57.2	-32.2
5	35	19.25	15.75
6	49	57.2	-8.2
7	68	57.2	10.8
8	71	57.2	13.8
9	73	57.2	15.8

Predict a person's age example

Now we can fit a second regression tree to the residuals of the first tree.

Tree2



Predict a person's age example

Now we can improve the predictions from our first tree by adding the "error-correcting" predictions from this tree.

PersonID	Age	Tree1 Prediction	Tree1 Residual	Tree2 Prediction	Combined Prediction	Final Residual
1	13	19.25	-6.25	-3.567	15.68	2.683
2	14	19.25	-5.25	-3.567	15.68	1.683
3	15	19.25	-4.25	-3.567	15.68	0.6833
4	25	57.2	-32.2	-3.567	53.63	28.63
5	35	19.25	15.75	-3.567	15.68	-19.32
6	49	57.2	-8.2	7.133	64.33	15.33
7	68	57.2	10.8	-3.567	53.63	-14.37
8	71	57.2	13.8	7.133	64.33	-6.667
9	73	57.2	15.8	7.133	64.33	-8.667

Tree1 SSE	Combined SSE
1994	1765

Gradient Boosting - Theory 1

Inspired by the idea above, we create our first (naive) formalization of gradient boosting. In pseudocode

1. Fit a model to the data, $F_1(x) = y$

It's not hard to see how we can generalize this idea by inserting more models that correct the errors of the previous model.

Gradient Boosting - Theory 1

Inspired by the idea above, we create our first (naive) formalization of gradient boosting. In pseudocode

1. Fit a model to the data, $F_1(x) = y$
2. Fit a model to the residuals, $h_1(x) = y - F_1(x)$

It's not hard to see how we can generalize this idea by inserting more models that correct the errors of the previous model.

Gradient Boosting - Theory 1

Inspired by the idea above, we create our first (naive) formalization of gradient boosting. In pseudocode

1. Fit a model to the data, $F_1(x) = y$
2. Fit a model to the residuals, $h_1(x) = y - F_1(x)$
3. Create a new model,

It's not hard to see how we can generalize this idea by inserting more models that correct the errors of the previous model.

Gradient Boosting - Theory 1

Specifically,

$$F(x) = F_1(x) \mapsto F_2(x) = F_1(x) + h_1(x) \cdots \mapsto$$
$$F_M(x) = F_{M-1}(x) + h_{M-1}(x)$$

where $F_1(x)$ is an initial model fit to y

Since we initialize the procedure by fitting $F_1(x)$, our task at each step is to find

$$h_m(x) = y - F_m(x)$$

.

Gradient Boosting - Theory 2

Now we'll tweak our model to conform to most gradient boosting implementations - we'll initialize the model with a single prediction value. Since our task (for now) is to minimize squared error, we'll initialize F with the mean of the training target values.

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma) = \arg\min_{\gamma} \sum_{i=1}^{n} (\gamma - y_i)^2 = \frac{1}{n} \sum_{i=1}^{n} y_i$$

Then we can define each subsequent F_m*recursively*, just like before

$$F_{m+1}(x) = F_m(x) + h_m(x) = y, \textit{for } m \geq 0$$

where h_m comes from a class of base learners $\mathcal{H}$ (e.g. regression trees).

Gradient Boosting - Theory 3 Gradient Descent

Let's refresh this idea using the concept of gradient descent. Consider a differentiable function we want to minimize. For example,

$$L(x_1, x_2) = \frac{1}{2}(x_1 - 15)^2 + \frac{1}{2}(x_2 - 25)^2$$

The goal here is to find the pair (x_1, x_2) that minimizes L. Notice, you can interpret this function as calculating the squared error for two data points, 15 and 25 given two prediction values, x_1 and x_2 (but with a $\frac{1}{2}$ multiplier to make the math work out nicely). Although we can minimize this function directly, gradient descent will let us minimize more complicated loss functions that we can't minimize directly.

Gradient Boosting - Theory 3 Gradient Descent

Initialization Steps: Number of iteration steps M = 100 Starting point $s^0 = (0, 0)$ Step size $\gamma = 0.1$

For iteration $m = 1 \rightarrow M$:

1. Calculate the gradient of L at the point $s^{(m-1)}$

That is,

$$s^m = s^{(m-1)} - \gamma \nabla L(s^{(m-1)})$$

If γ is small and M is sufficiently large, s^M will be the location of L's minimum value.

Gradient Boosting - Theory 3 Gradient Descent

Initialization Steps: Number of iteration steps M = 100 Starting point $s^0 = (0, 0)$ Step size $\gamma = 0.1$

For iteration $m = 1 \rightarrow M$:

1. Calculate the gradient of L at the point $s^{(m-1)}$
2. "Step" in the direction of greatest descent (the negative gradient) with step size γ.

That is,

$$s^m = s^{(m-1)} - \gamma \nabla L(s^{(m-1)})$$

If γ is small and M is sufficiently large, s^M will be the location of L's minimum value.

Leveraging Gradient Descent - Theory 4

Now we can use gradient descent for our gradient boosting model.

The objective function we want to minimize is L. Our starting point is $F_0(x)$.

For iteration $m = 1$, we compute the gradient of L with respect to $F_0(x)$.

Then we fit a weak learner to the gradient components.

In the case of a regression tree, leaf nodes produce an average gradient among samples with similar features.

For each leaf, we step in the direction of the average gradient (using line search to determine the step magnitude). The result is F_1.

Then we can repeat the process until we have F_M.

Leveraging Gradient Descent - Theory 4

Initialize the model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

For m = 1 to M:

- Compute pseudo residuals,
  $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$

Leveraging Gradient Descent - Theory 4

Initialize the model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

For m = 1 to M:

- Compute pseudo residuals,
  $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \ldots, n$.
- Fit base learner, $h_m(x)$ to pseudo residuals

Leveraging Gradient Descent - Theory 4

Initialize the model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

For m = 1 to M:

- Compute pseudo residuals,
  $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$
- Fit base learner, $h_m(x)$ to pseudo residuals
- Compute step magnitude multiplier γ_m.

Leveraging Gradient Descent - Theory 4

Initialize the model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$$

For m = 1 to M:

- Compute pseudo residuals,
  $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \ldots, n$.
- Fit base learner, $h_m(x)$ to pseudo residuals
- Compute step magnitude multiplier γ_m.
- Update $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$

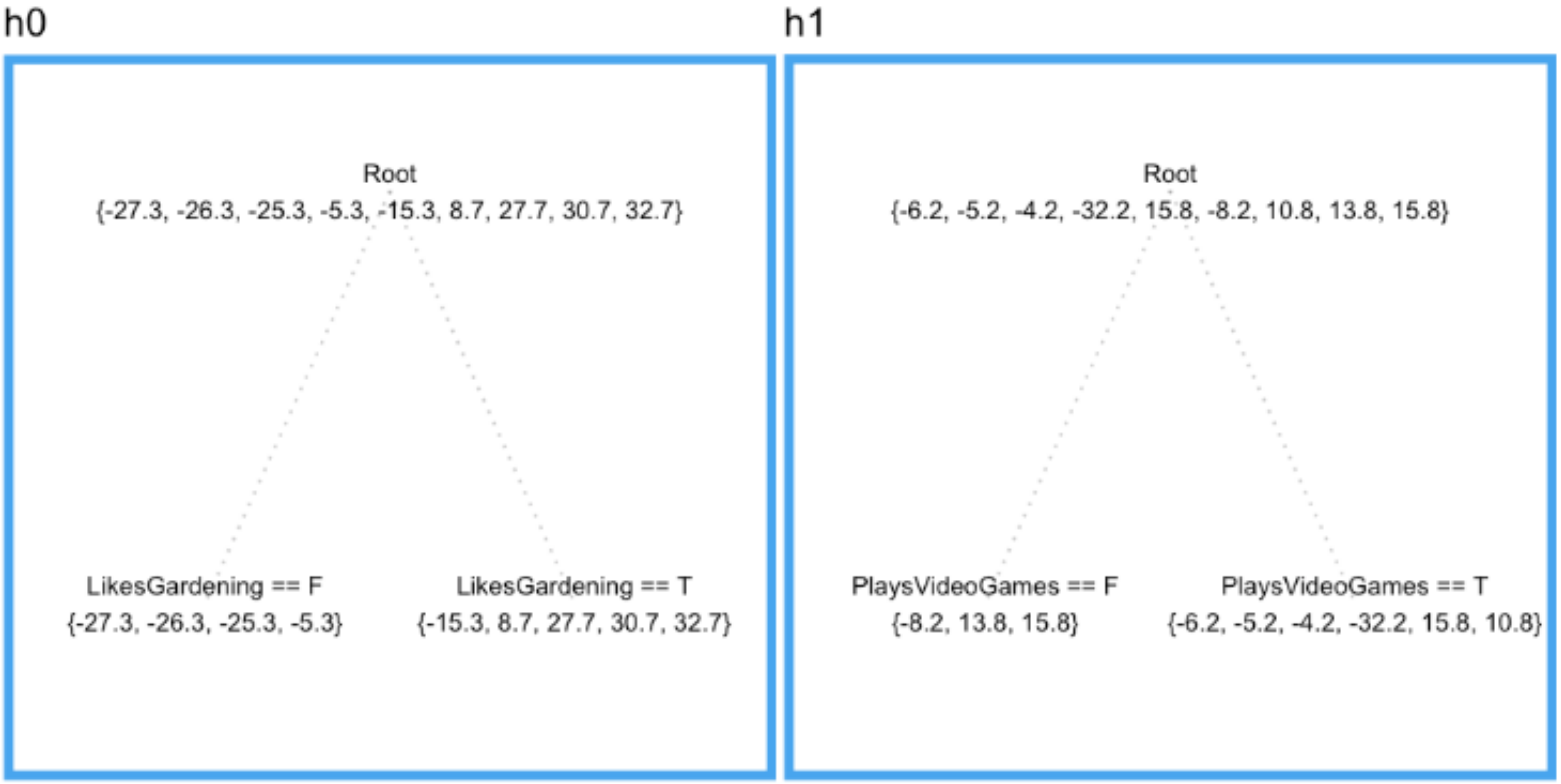
Leveraging Gradient Descent - Theory 4

In case you want to check your understanding so far, our current gradient boosting applied to our sample problem for squared error yields the following results.

Squared Error

Age	F0	PseudoResidual0	h0	gamma0	F1	PseudoResidual1	h1	gamma1	F2
13	40.33	-27.33	-21.08	1	19.25	-6.25	-3.567	1	15.68
14	40.33	-26.33	-21.08	1	19.25	-5.25	-3.567	1	15.68
15	40.33	-25.33	-21.08	1	19.25	-4.25	-3.567	1	15.68
25	40.33	-15.33	16.87	1	57.2	-32.2	-3.567	1	53.63
35	40.33	-5.333	-21.08	1	19.25	15.75	-3.567	1	15.68
49	40.33	8.667	16.87	1	57.2	-8.2	7.133	1	64.33
68	40.33	27.67	16.87	1	57.2	10.8	-3.567	1	53.63
71	40.33	30.67	16.87	1	57.2	13.8	7.133	1	64.33
73	40.33	32.67	16.87	1	57.2	15.8	7.133	1	64.33

Leveraging Gradient Descent - Theory 4



Gradient Boosting in Practice

Advantage

- Incredibly effective in practice.

Disadvantage

What else can it do? Although the example above presented gradient boosting as a regression model, it's also very effective as a classification problems

Gradient Boosting in Practice

Advantage

- Incredibly effective in practice.
- Often performs better than any other algorithm

Disadvantage

What else can it do? Although the example above presented gradient boosting as a regression model, it's also very effective as a classification problems

Gradient Boosting in Practice

Advantage

- Incredibly effective in practice.
- Often performs better than any other algorithm
- Directly optmizes cost function

Disadvantage

What else can it do? Although the example above presented gradient boosting as a regression model, it's also very effective as a classification problems

Gradient Boosting in Practice

Advantage

- Incredibly effective in practice.
- Often performs better than any other algorithm
- Directly optmizes cost function

Disadvantage

- Overfits (need to find a proper stopping point)

What else can it do? Although the example above presented gradient boosting as a regression model, it's also very effective as a classification problems

Gradient Boosting in Practice

Advantage

- Incredibly effective in practice.
- Often performs better than any other algorithm
- Directly optmizes cost function

Disadvantage

- Overfits (need to find a proper stopping point)
- Sensitive to extreme values and noises

What else can it do? Although the example above presented gradient boosting as a regression model, it's also very effective as a classification problems

Train a GBM Model in R

```
# Train a 5000-tree GBM model

> model <- gbm(formula = response ~ .,
               distribution = "bernoulli",
               data = train,
               n.trees = 5000)
```

Please complete "train_gbm_model.Rmd"

Understanding GBM model output

```
# Print  model
print(credit_model)
```

```
gbm(formula = default ~ ., distribution = "bernoulli",
    data = credit_train, n.trees = 100)
A gradient boosted model with bernoulli loss function.
100 iterations were performed.
There were 4 predictors of which 0 had non-zero influence.
```

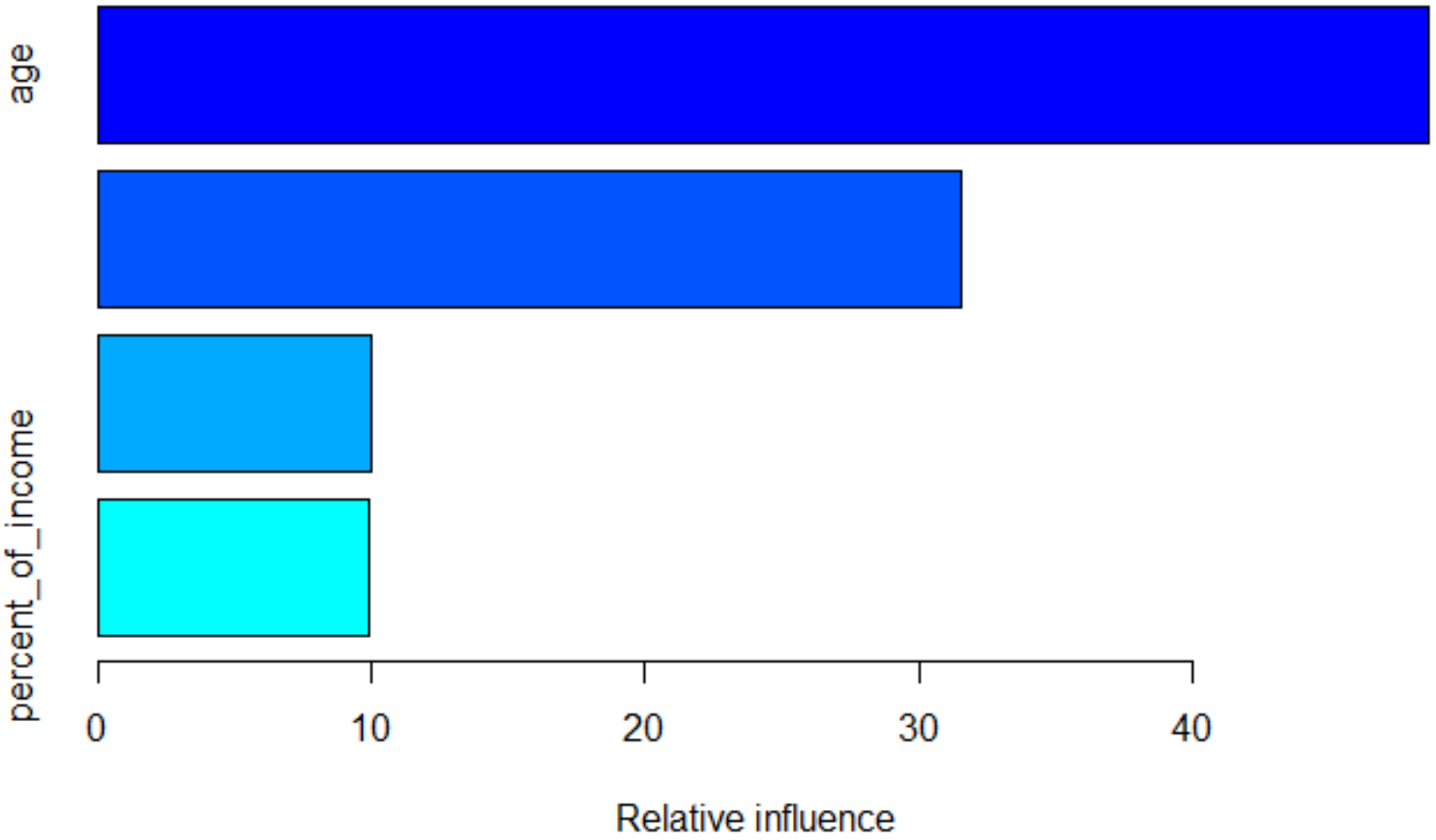
Variable importance

```r
# summary() prints variable importance
summary(credit_model)
```

	var <fctr>	rel.inf <dbl>
age	age	48.645680
months_loan_duration	months_loan_duration	31.505420
years_at_residence	years_at_residence	9.965874
percent_of_income	percent_of_income	9.883026

4 rows

Variable importance



Prediction using GBM

```
?predict.gbm
predict(model, type = "response", n.trees = 10000)
```

Tuning a GBM model

GBM Hyperparameters

- *n.trees:* number of trees

Tuning a GBM model

GBM Hyperparameters

- *n.trees:* number of trees
- *bag.fraction:* proportion of observations to be sampled in each tree

Tuning a GBM model

GBM Hyperparameters

- *n.trees:* number of trees
- *bag.fraction:* proportion of observations to be sampled in each tree
- *n.minobsinnode:* minimum number of observations in the trees terminal nodes

Tuning a GBM model

GBM Hyperparameters

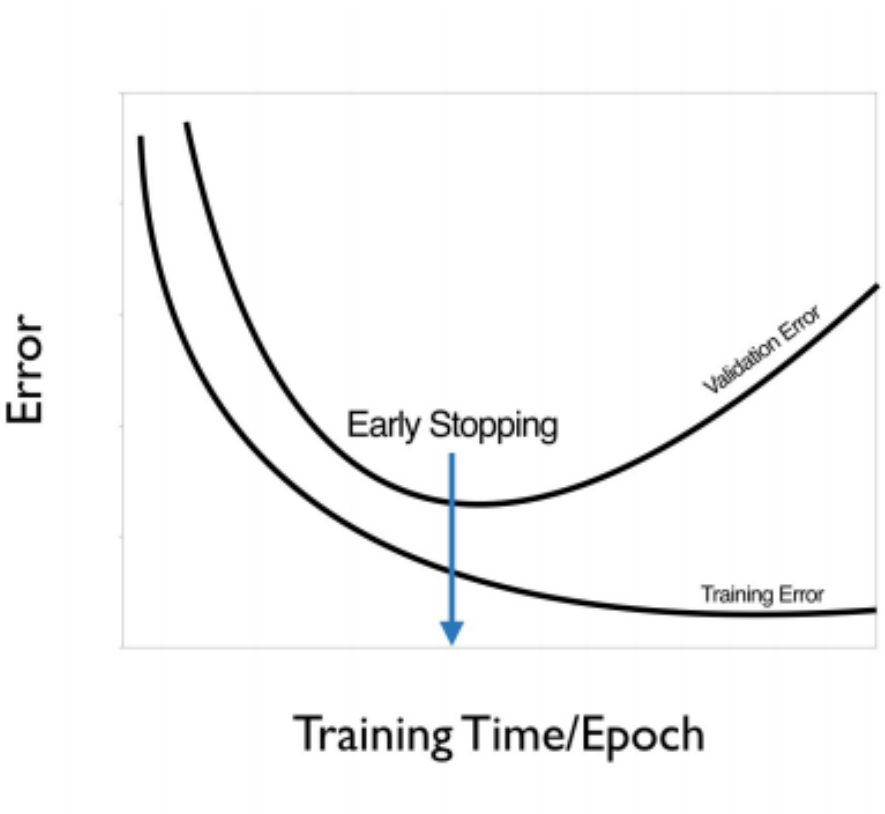
- *n.trees:* number of trees
- *bag.fraction:* proportion of observations to be sampled in each tree
- *n.minobsinnode:* minimum number of observations in the trees terminal nodes
- *interaction.depth:* maximum nodes per tree

Tuning a GBM model

GBM Hyperparameters

- *n.trees:* number of trees
- *bag.fraction:* proportion of observations to be sampled in each tree
- *n.minobsinnode:* minimum number of observations in the trees terminal nodes
- *interaction.depth:* maximum nodes per tree
- *shrinkage:* learning rate

Early Stopping



Early Stopping in GBMs

```
# train a GBM model

model <- gbm(formula = response ~ .,
             distribution = "bernoulli",
             data = train,
             n.trees = 5000,
             cv.folds = 3)
```

```
# get optimal ntree based on CV error

ntree_opt_cv <- gbm.perf(model, method = "cv")
```

```
# get optimal ntree based on OOB error

ntree_opt_oob <- gbm.perf(model, method = "OOB")
```