UCK358E – INTR. TO ARTIFICIAL INTELLIGENCE

SPRING ‘23

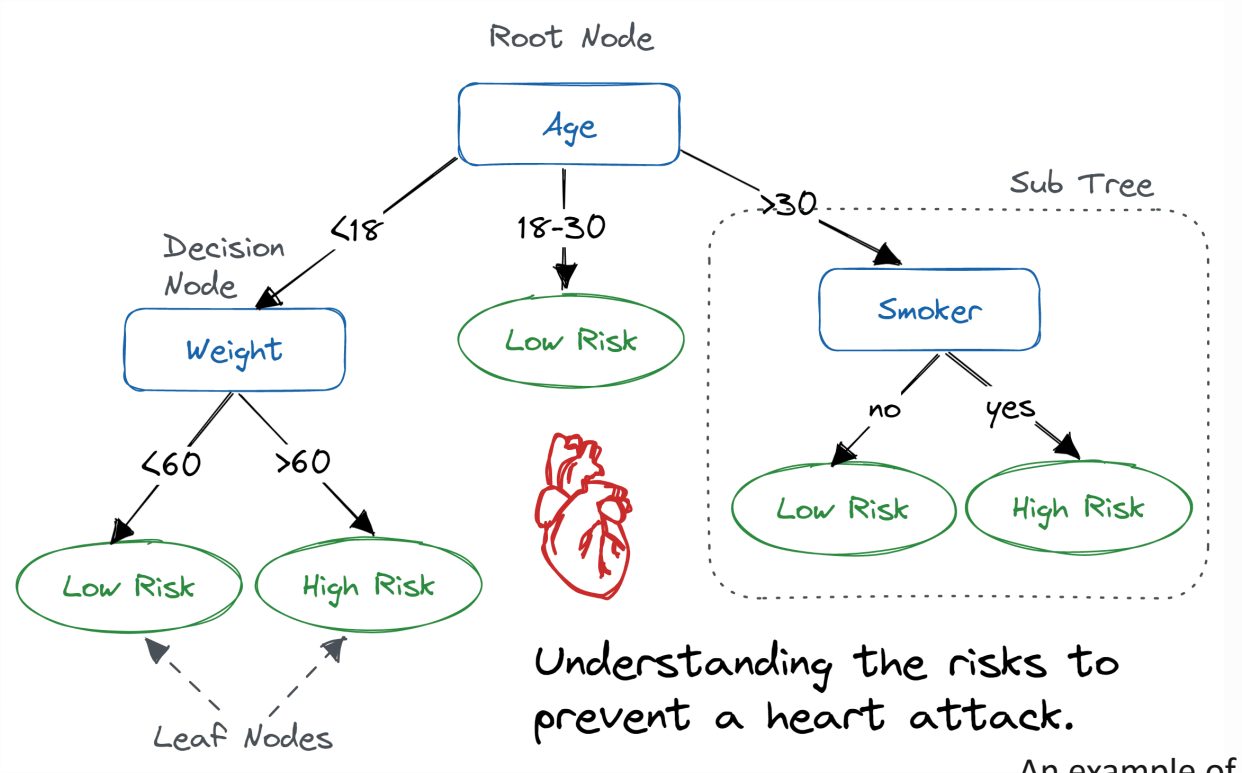
LECTURE 6

DECISION TREES

Instructor: Asst. Prof. Barış Başpınar

Decision Tree

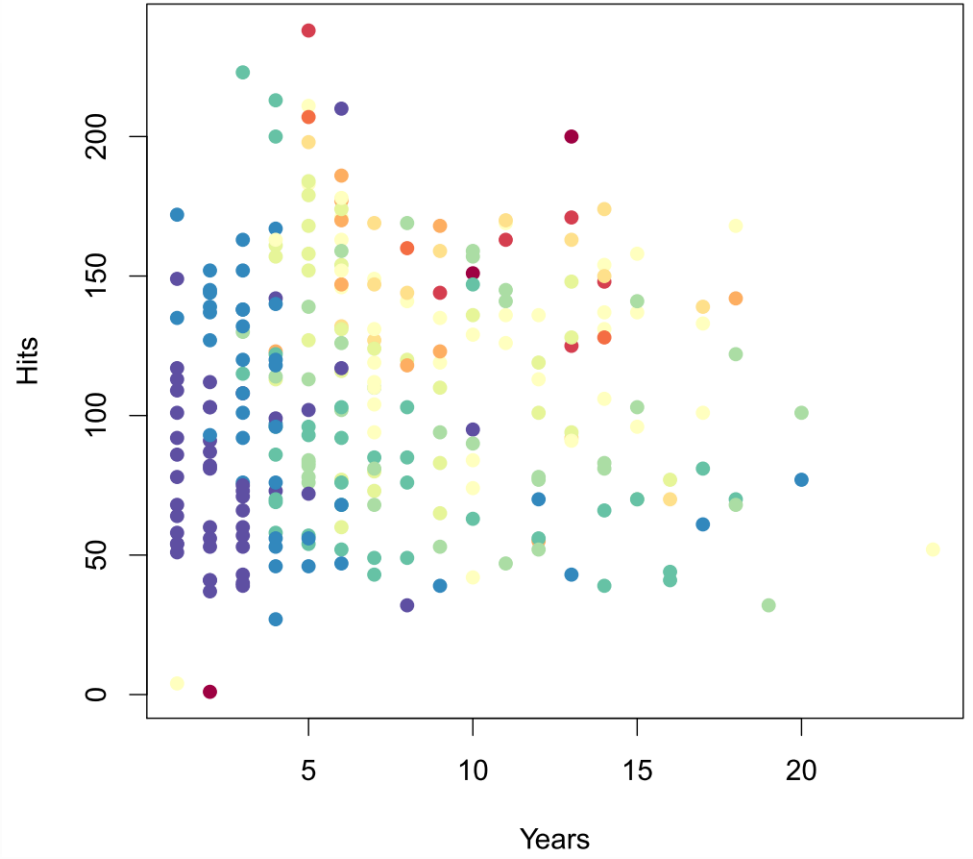
- Decision trees are non-parametric supervised learning method used for classification and regression
- These involve stratifying or segmenting the predictor space into a number of simple regions
- Essentially, they learn a hierarchy of if/else questions, leading to a decision

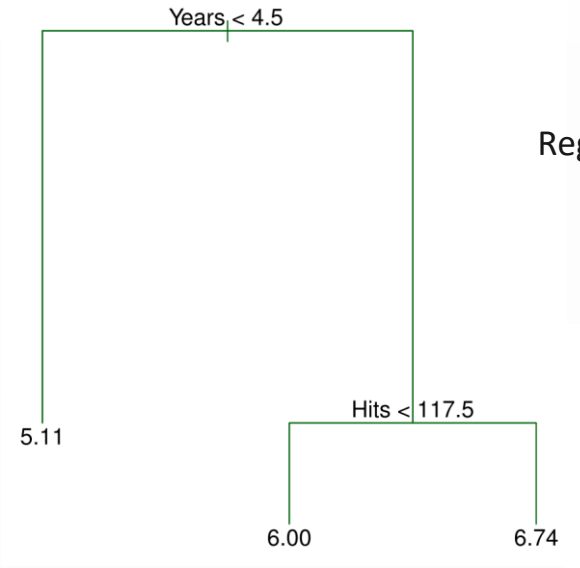


An example of classification tree

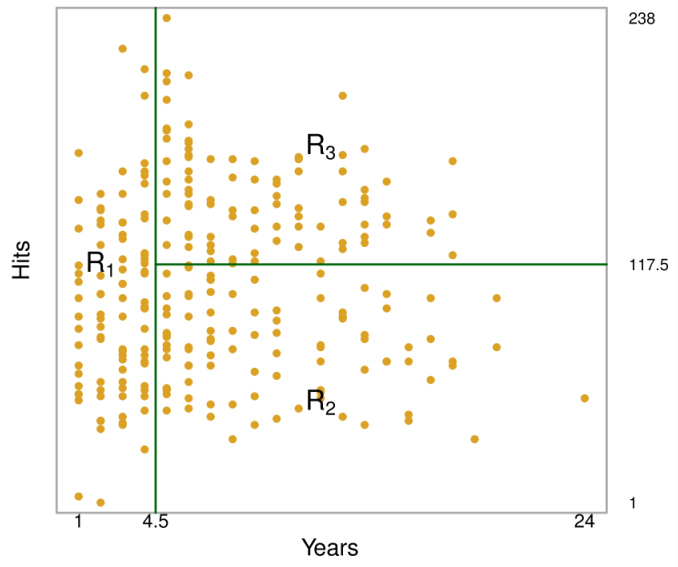
Baseball salary data - how would you stratify it?

- Salary is color-coded from low (blue, green) to high (yellow, red)



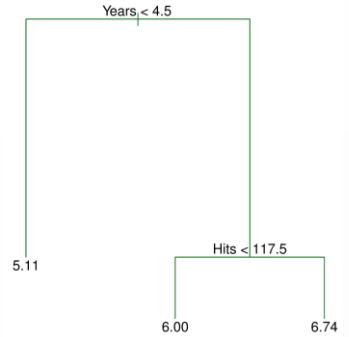


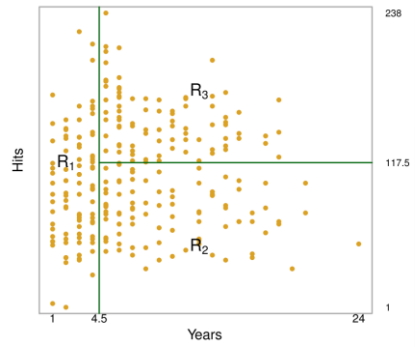
Regression tree for this data



Interpretation of Results

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

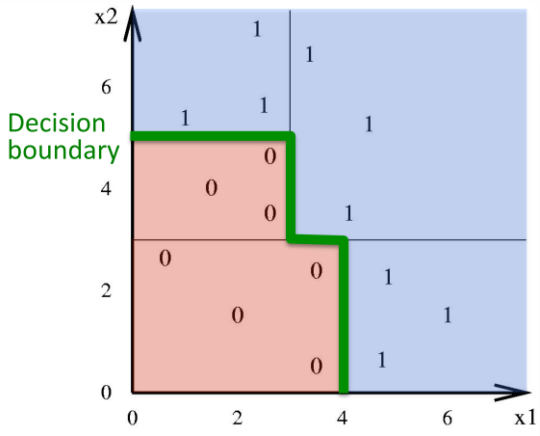


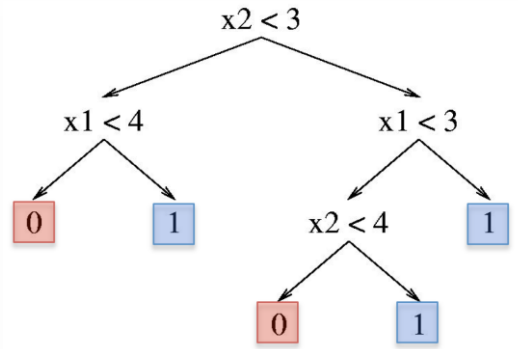


- Years is the most important factor in determining Salary, and players with less experience earn lower salaries than more experienced players
- Given that a player is less experienced, the number of Hits that he made in the previous year seems to play little role in his Salary
- But among players who have been in the major leagues for five or more years, players who made more Hits last year tend to have higher salaries
- It is easy to display, interpret and explain

Decision Trees – Decision Boundary

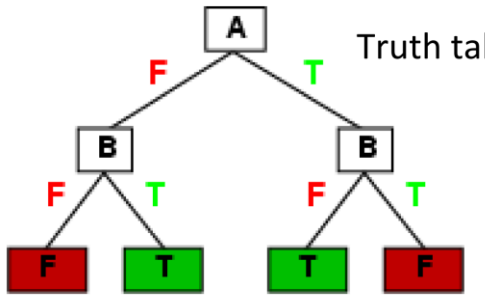
- Decision trees divide the feature space into axis-parallel (hyper-)rectangles
  - Non-linear decision boundaries can be generated





- Decision trees can represent any Boolean function of the input attributes

A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



Truth table row → path to leaf

Decision Trees

- Several decision tree algorithms:
  - ID3 (Iterative Dichotomiser 3)
    - Handling classification tasks
    - Using categorical attributes/features)
    - Using entropy ($H(s)$) or information gain (IG(s))

    for attribute selection
  - C4.5 (the successor of ID3)
    - Handling classification tasks
    - Using both continuous and categorical features
  - CART (classification and regression tree)
    - Handling both classification and regression tasks

- How to build a decision tree:
  - Start at the top of the tree
  - Grow it by "splitting" attributes one by one. To determine which attribute to split, look at "node impurity"
  - Assign leaf nodes the majority vote in the leaf
  - When we get to the bottom, prune the tree to prevent overfitting

- Key problems:
  - How to select best decision attribute?
  - How decide the threshold for the selected attribute? (for continuous features)
  - How to prune the tree?

Will the customer wait for a table?

- Let us firstly focus on a classification task with categorical variables (to illustrate the C4.5 algorithm)

Attributes/Features

1.	Alternate: whether there is a suitable alternative restaurant nearby.
2.	Bar: whether the restaurant has a comfortable bar area to wait in.
3.	Fri/Sat: true on Fridays and Saturdays.
4.	Hungry: whether we are hungry.
5.	Patrons: how many people are in the restaurant (values are None, Some, and Full).
6.	Price: the restaurant's price range ($, $$, $$$).
7.	Raining: whether it is raining outside.
8.	Reservation: whether we made a reservation.
9.	Type: the kind of restaurant (French, Italian, Thai or Burger).
10.	WaitEstimate: the wait estimated by the host (0-10 minutes, 10-30, 30-60, >60).

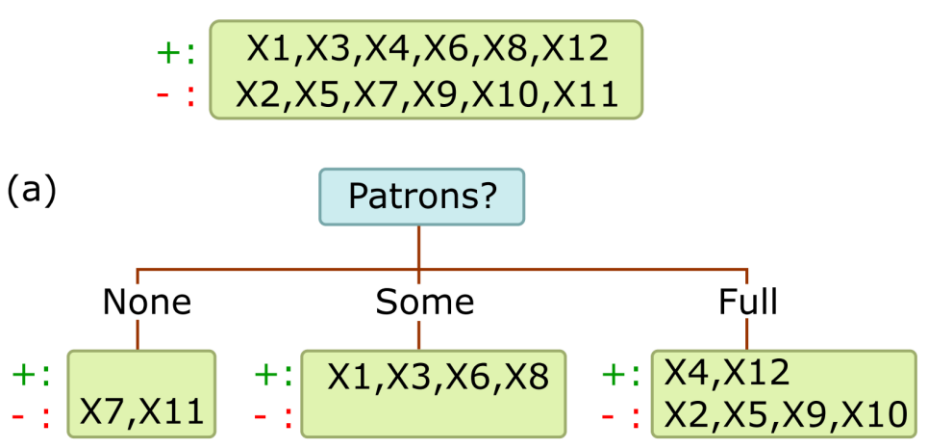
Continuous variables can be discretized to evaluate them as categorical variables

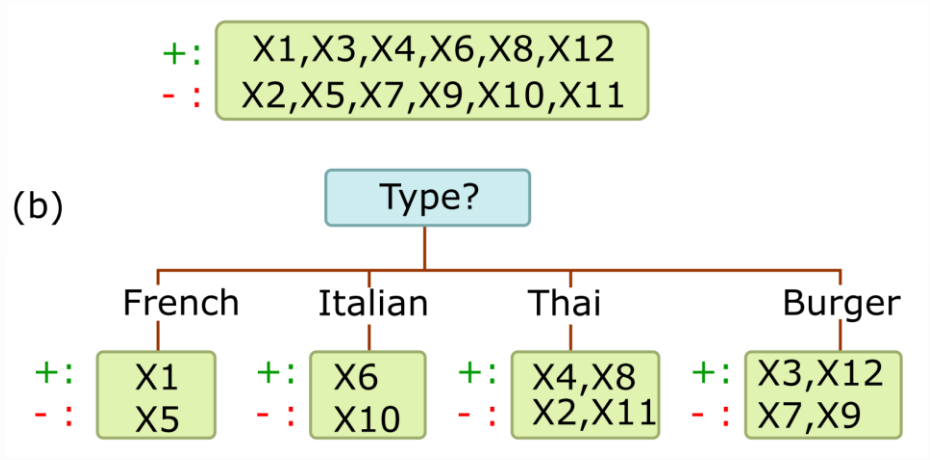
Examples/Samples

Example	Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
X_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0-10	Yes
X_2	Yes	No	No	Yes	Full	$	No	No	Thai	30-60	No
X_3	No	Yes	No	No	Some	$	No	No	Burger	0-10	Yes
X_4	Yes	No	Yes	Yes	Full	$	No	No	Thai	10-30	Yes
X_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	No
X_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0-10	Yes
X_7	No	Yes	No	No	None	$	Yes	No	Burger	0-10	No
X_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0-10	Yes
X_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	No
X_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10-30	No
X_{11}	No	No	No	No	None	$	No	No	Thai	0-10	No
X_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30-60	Yes

Will the customer wait for a table?

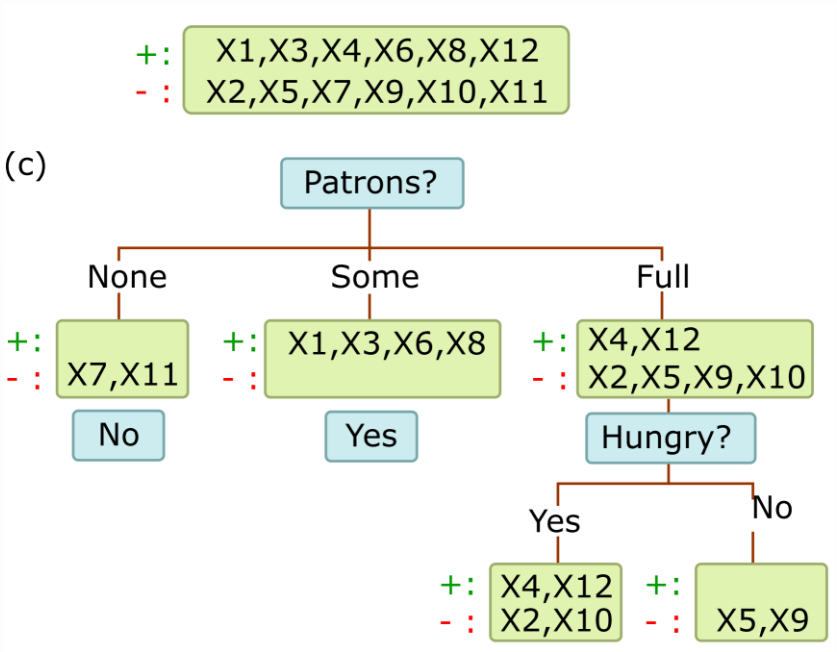
- Here are two options for the first feature to split at the top of the tree.
- Which one should we choose? Which one gives me the most information?





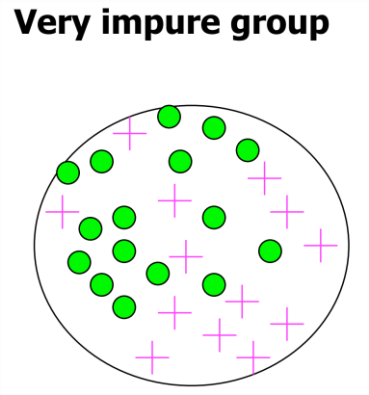
Will the customer wait for a table?

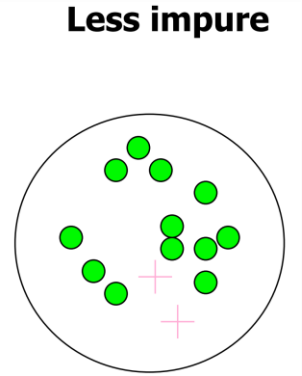
- What we need is a formula to compute “information” to chose the best attribute
- Before we do that, here’s another example. Let’s say we pick one of them (Patrons). Maybe then we’ll pick Hungry next, because it has a lot of “information”:

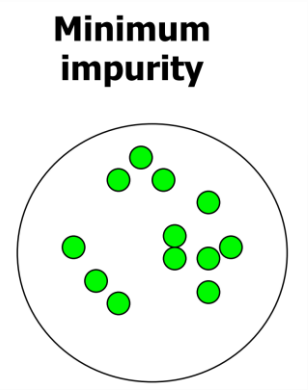


Choosing an Attribute

- **Idea**: a good attribute splits the examples into subsets that are (ideally) “all positive” or “all negative”
- **Metric**: Impurity/Entropy (informal)
  - Measures the level of impurity in a group of examples





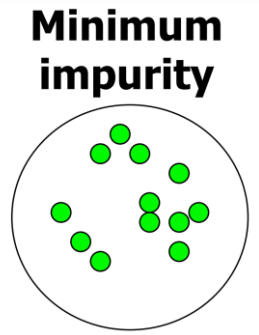


Entropy: a common way to measure impurity

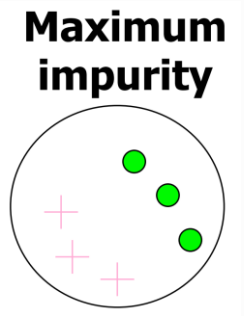
- Entropy $H(X)$ of a random variable X

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

- 2-Class Cases:
  - What is the entropy of a group in which all examples belong to the same class?
    - entropy = $-1\log_2 1 = 0$

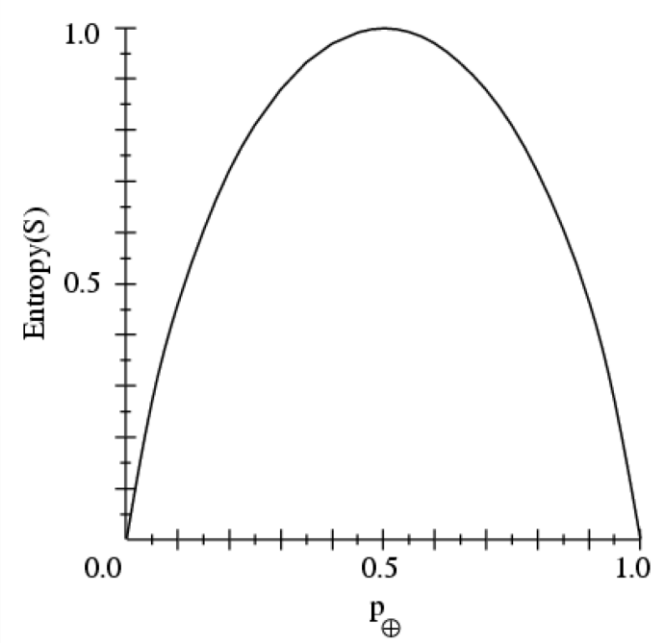


  - What is the entropy of a group with 50% in either class?
    - entropy = $-0.5\log_2 0.5 - 0.5\log_2 0.5 = 1$



Entropy: a common way to measure impurity

- 2-Class Cases:



- S is a sample of training examples
- $p_\oplus$ is the proportion of positive examples in S
- $p_\ominus$ is the proportion of negative examples in S
- Entropy measures the impurity of S

$$H(S) \equiv -p_\oplus \log_2 p_\oplus - p_\ominus \log_2 p_\ominus$$

Information Gain

- We want to determine which attribute in a given set of training feature vectors is most useful for discriminating between the classes to be learned
- Information gain tells us how important a given attribute of the feature vectors is
- We can use it to decide the ordering of attributes in the nodes of a decision tree (as in C4.5)

Entropy $H(X)$ of a random variable X:

$$H(X) = -\sum_{i=1}^{n} P(X=i)\log_2 P(X=i)$$

Specific conditional entropy $H(X|Y=v)$ of X given $Y=v$:

$$H(X|Y=v) = -\sum_{i=1}^{n} P(X=i|Y=v)\log_2 P(X=i|Y=v)$$

Conditional entropy $H(X|Y)$ of X given Y:

$$H(X|Y) = \sum_{v \in values(Y)} P(Y=v)H(X|Y=v)$$

Mutual information (or Information Gain) of X and Y :

$$I(X,Y) = H(X) - H(X|Y) = H(Y) - H(Y|X)$$

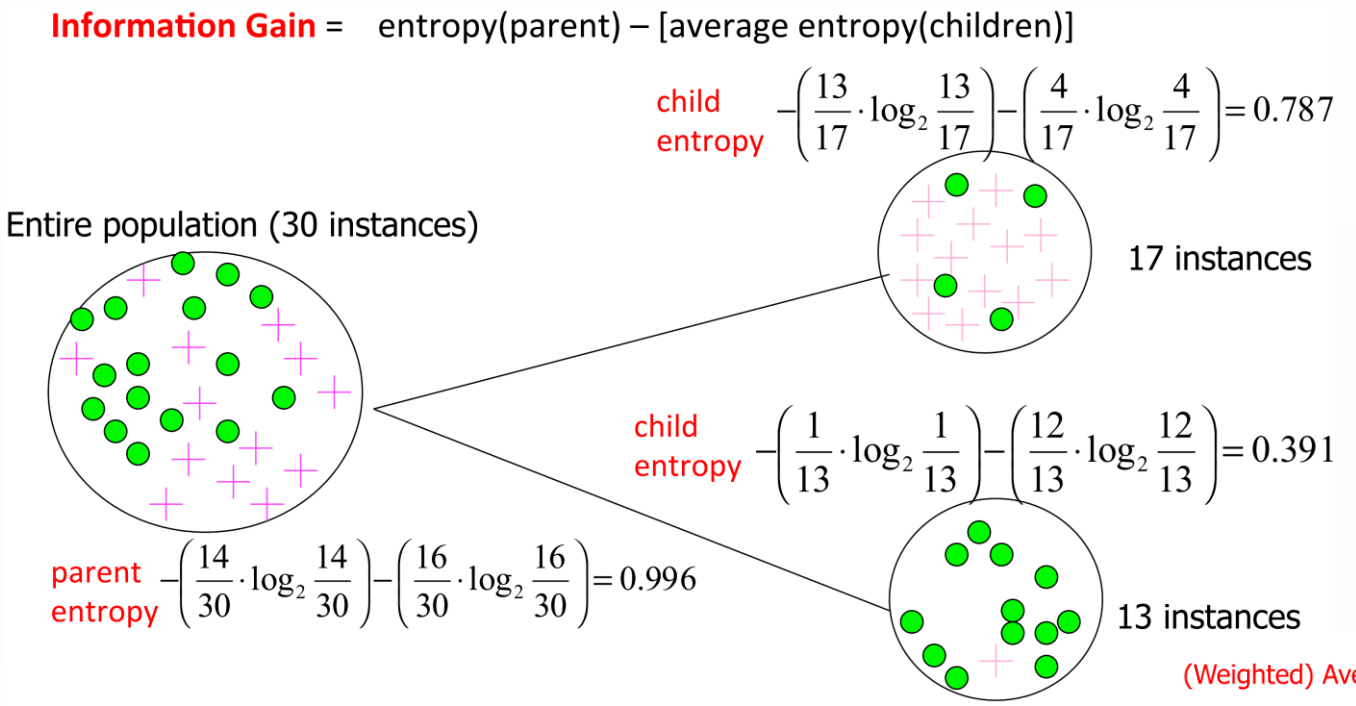
Information Gain

- Information Gain is the mutual information between input attribute A and target variable Y
  - Information Gain is the expected reduction in entropy of target variable Y for data sample S, due to sorting on variable A

$$Gain(S,A) = I_S(A,Y) = H_S(Y) - H_S(Y|A)$$

- An example for calculating information gain:

Information Gain = entropy(parent) – [average entropy(children)]



(Weighted) Average Entropy of Children = $\left(\frac{17}{30}\cdot 0.787\right)+\left(\frac{13}{30}\cdot 0.391\right)=0.615$

Information Gain= 0.996 - 0.615 = 0.38

Will the customer wait for a table?

- Back to the example with the restaurants:

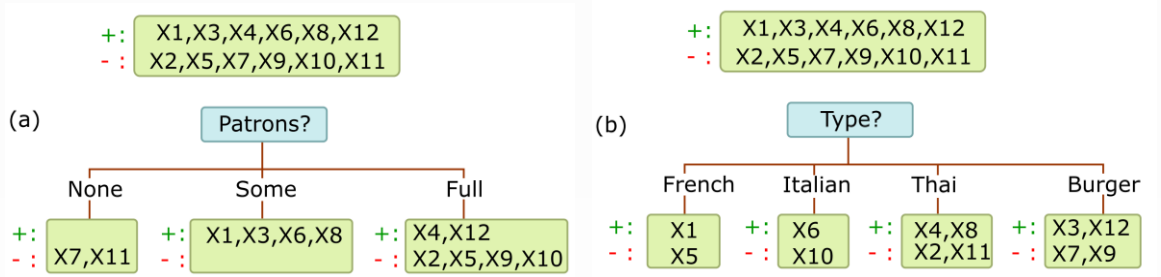
The Information Gain is calculated like this:

$$\begin{aligned}\text{Gain}(S, A) &= \text{expected reduction in entropy due to branching on attribute } A \\ &= \text{original entropy} - \text{entropy after branching}\end{aligned}$$

$$\text{Gain}(S, \text{Patrons}) = H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) - \left[\frac{2}{12}H([0,1]) + \frac{4}{12}H([1,0]) + \frac{6}{12}H\left(\left[\frac{2}{6}, \frac{4}{6}\right]\right)\right] \approx 0.541$$

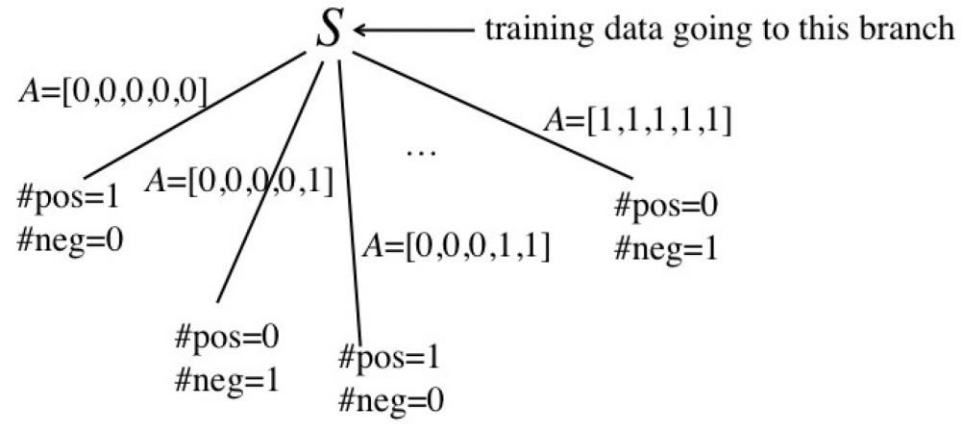
$$\text{Gain}(S, \text{Type}) = 1 - \left[\frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) + \frac{2}{12}H\left(\left[\frac{1}{2}, \frac{1}{2}\right]\right) + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right) + \frac{4}{12}H\left(\left[\frac{2}{4}, \frac{2}{4}\right]\right)\right] \approx 0$$

- Actually Patrons has the highest gain among the attributes, and is chosen to be the root of the tree
- In general, we want to choose the feature A that maximizes Gain(S, A)



Information Gain: A Comment

- One problem with Gain is that it likes to partition too much, and favors numerous splits: e.g., if each branch contains 1 example:



Then,

$$H\left[\frac{\#\text{pos}_j}{\#\text{pos}_j + \#\text{neg}_j}, \frac{\#\text{neg}_j}{\#\text{pos}_j + \#\text{neg}_j}\right] = 0 \text{ for all } j,$$

so all those negative terms would be zero and we'd choose that attribute over all the others.

- An alternative to Gain is the Gain Ratio

Gain Ratio

- We want to have a large Gain, but also we want small partitions. We'll choose our attribute according to:

$$\frac{\text{Gain}(S, A)}{\text{SplitInfo}(S, A)} \begin{array}{l} \leftarrow \text{ want large} \\ \leftarrow \text{ want small} \end{array}$$

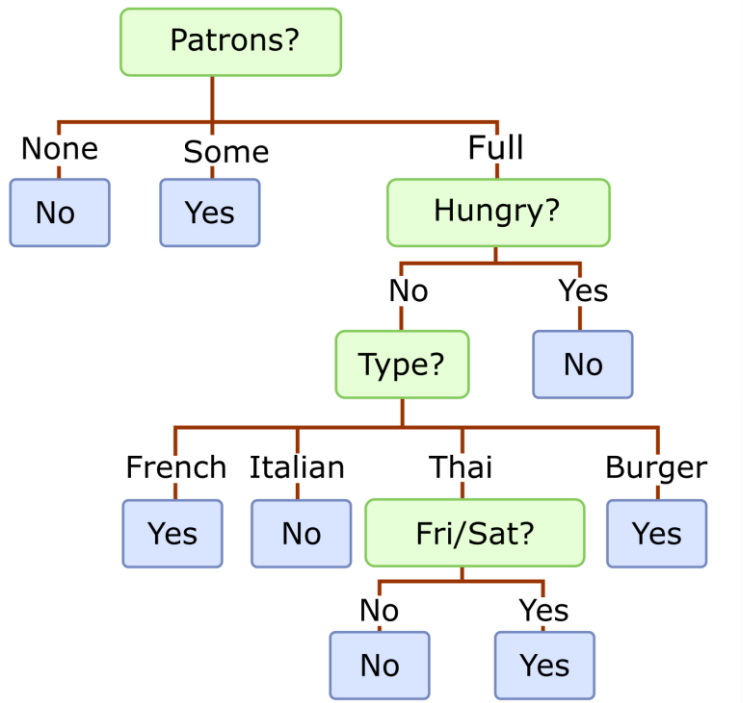
where SplitInfo(S, A) comes from the partition:

$$\text{SplitInfo}(S, A) = -\sum_{j=1}^{J} \frac{|S_j|}{|S|} \log\left(\frac{|S_j|}{|S|}\right)$$

where $|S_j|$ is the number of examples in branch j. We want each term in the S sum to be large. That means we want $\frac{|S_j|}{|S|}$ to be large, meaning that we want lots of examples in each branch.

Will the customer wait for a table?

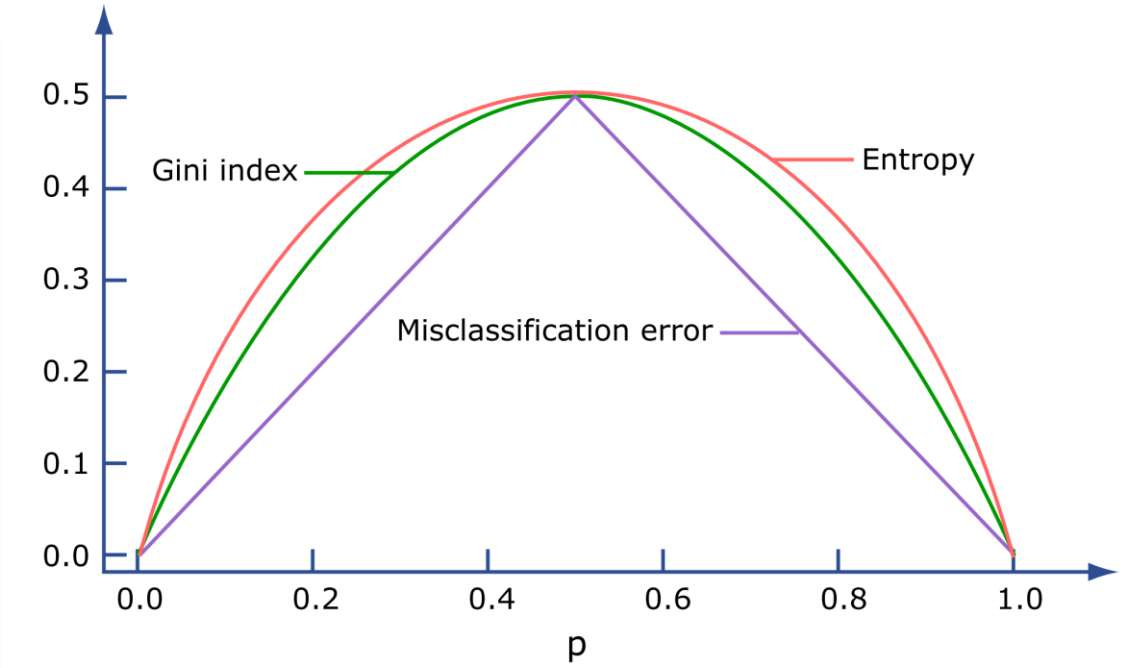
- Keep splitting until:
  - no more examples left (no point trying to split)
  - all examples have the same class
  - no more attributes to split
- For the restaurant example, we get tree in Figure
- There are possibilities to replace H([p, 1 − p]), it is not the only thing we can use!
  - One example is the Gini index 2p(1 − p) used by CART.
  - Another example is the value 1 − max(p, 1 − p)



The decision tree induced from the 12-example training set.

Alternative Metrics for Attribute Selection in Classification Trees

- C4.5 uses information gain for splitting, and CART uses the Gini index. (CART only has binary splits.)



Node impurity measures for two-class classification, as a function of the proportion p in class 2. Cross-entropy has been scaled to pass through (0.5, 0.5).

- All three are similar, but cross-entropy and the Gini index are differentiable, and hence more amenable to numerical optimization.

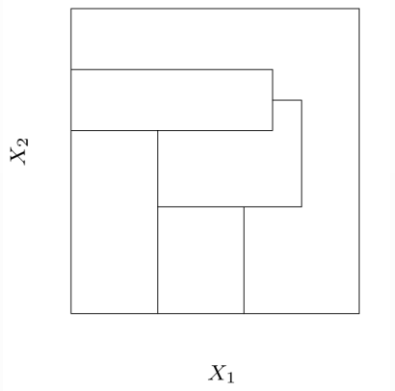
Misclassification error: $\frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \max_k(\hat{p}_{mk})$

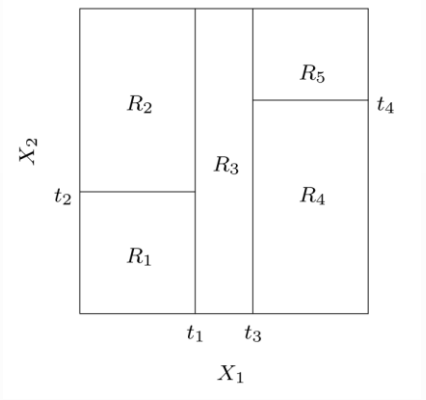
Gini index: $\sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).$

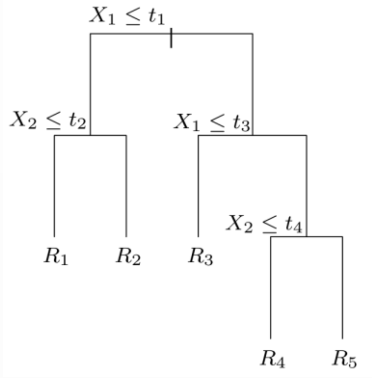
Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.$

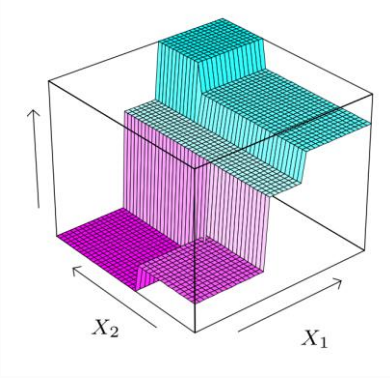
CART (Classification and Regression Tree)

- It uses recursive binary partitions like that in the second figure below
  - First split the space into two regions, and model the response by the mean of Y in each region,
  - Choose the variable and split-point to achieve the best fit,
  - Then one or both of these regions are split into two more regions,
  - And this process is continued, until some stopping rule is applied.









- Partitions and CART.
  - Second figure shows a partition of a two-dimensional feature space by recursive binary splitting, as used in CART, applied to some fake data
  - First figure shows a general partition that cannot be obtained from recursive binary splitting
  - Third figure shows the tree corresponding to the partition in the second figure, and
  - A perspective plot of the prediction surface appears in the last figure

- Note that multiway splits can be achieved by a series of binary splits

CART – Regression Task

- The algorithm needs to automatically decide on the splitting variables and split points, and also what topology (shape) the tree should have
- Suppose first that we have a partition into M regions $R_1, R_2, \ldots, R_M$, and we model the response as a constant c_m in each region:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m)$$

- If we adopt as our criterion minimization of the sum of squares $\sum(y_i - f(x_i))^2$, it is easy to see that the best $\hat{c}_m$ is just the average of y_i in region R_m

$$\hat{c}_m = \text{ave}(y_i | x_i \in R_m)$$

CART – Regression Task

- Now finding the best binary partition in terms of minimum sum of squares is generally computationally infeasible. Hence we proceed with a greedy algorithm.
- Starting with all of the data, consider a splitting variable j and split point s, and define the pair of half-planes

$$R_1(j,s) = \{X|X_j \leq s\} \text{ and } R_2(j,s) = \{X|X_j > s\}$$

- Then we seek the splitting variable j and split point s that solve

$$\min_{j,\,s}\left[\min_{c_1}\sum_{x_i\in R_1(j,s)}(y_i-c_1)^2+\min_{c_2}\sum_{x_i\in R_2(j,s)}(y_i-c_2)^2\right]$$

- For any choice j and s, the inner minimization is solved by

$$\hat{c}_1 = \text{ave}(y_i|x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = \text{ave}(y_i|x_i \in R_2(j,s))$$

  - The split point s can be determined very quickly. And hence by scanning through all of the inputs, determination of the best pair (j,s) is feasible

- Then this process is repeated on all of the resulting regions until completing the tree

CART – Regression Task: Pruning

- How large should we grow the tree?
  - Clearly a very large tree might overfit the data, while a small tree might not capture the important structure
  - Tree size is a tuning parameter governing the model's complexity
- One approach would be to split tree nodes only if the decrease in sum-of-squares due to the split exceeds some threshold
  - This strategy is too short-sighted, however, since a seemingly worthless split might lead to a very good split below it
- The preferred strategy is to grow a large tree T_0, stopping the splitting process only when some minimum node size (say 5) is reached. Then this large tree is pruned using *cost-complexity pruning*, which we now describe

CART – Regression Task: Pruning

- We define a subtree $T \subseteq T_0$ to be any tree that can be obtained by pruning T_0, that is, collapsing any number of its internal (non-terminal) nodes
- We index terminal nodes by m, with node m representing region R_m. Let $|T|$ denote the number of terminal nodes in T. Letting

$$N_m = \#\{x_i \in R_m\},$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i,$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2,$$

We define the cost complexity criterion:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

CART – Regression Task: Pruning

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q_m(T) + \alpha|T|$$

- The idea is to find, for each α, the subtree $T \subseteq T_0$ to minimize $C_\alpha(T)$
- The tuning parameter $\alpha \geq 0$ governs the tradeoff between tree size and its goodness of fit to the data
  - Large values of α result in smaller trees T_α , and conversely for smaller values of α
  - As the notation suggests, with $\alpha = 0$ the solution is the full tree T_0
- Estimation of α can be achieved by five- or tenfold cross-validation:
  - We select an optimal value $\hat{\alpha}$ to minimize the cross-validated sum of squares
  - We then return to the full data set and obtain the subtree corresponding to $\hat{\alpha}$. Our final tree is $T_{\hat{\alpha}}$

CART – Classification Task

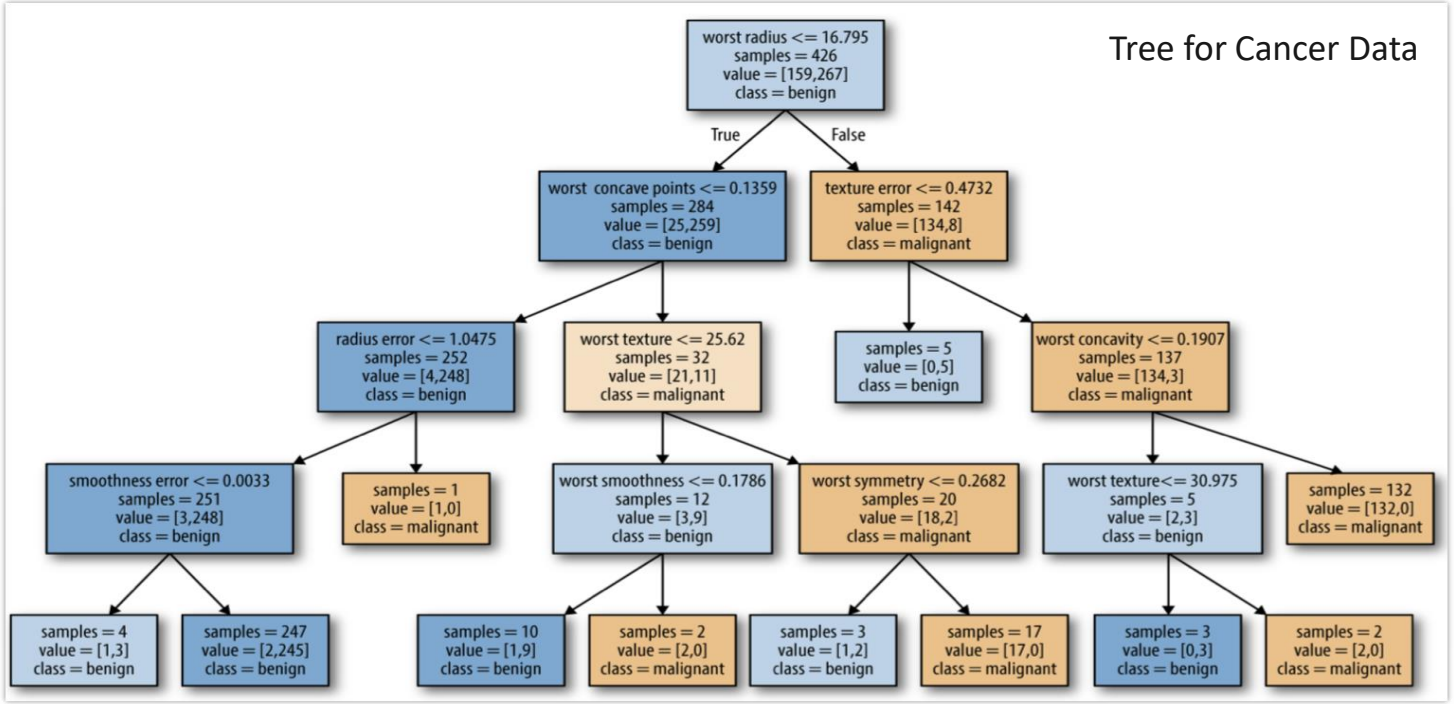
- If the target is a classification outcome taking values $1, 2, \ldots, K$, the only changes needed in the tree algorithm pertain to the criteria for splitting nodes and pruning the tree.
- As presented before, an impurity measure should be used

$$
\begin{aligned}
&\text{Misclassification error:} && \frac{1}{N_m}\sum_{i\in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{mk(m)}.\\
&\text{Gini index:} && \sum_{k\neq k'} \hat{p}_{mk}\hat{p}_{mk'} = \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}).\\
&\text{Cross-entropy or deviance:} && -\sum_{k=1}^{K} \hat{p}_{mk}\log\hat{p}_{mk}.
\end{aligned}
$$

- The cross-entropy and the Gini index are more sensitive to changes in the node probabilities than the misclassification rate. For this reason, either the Gini index or cross-entropy should be used when growing the tree.
- To guide cost-complexity pruning, any of the three measures can be used, but typically it is the misclassification rate.

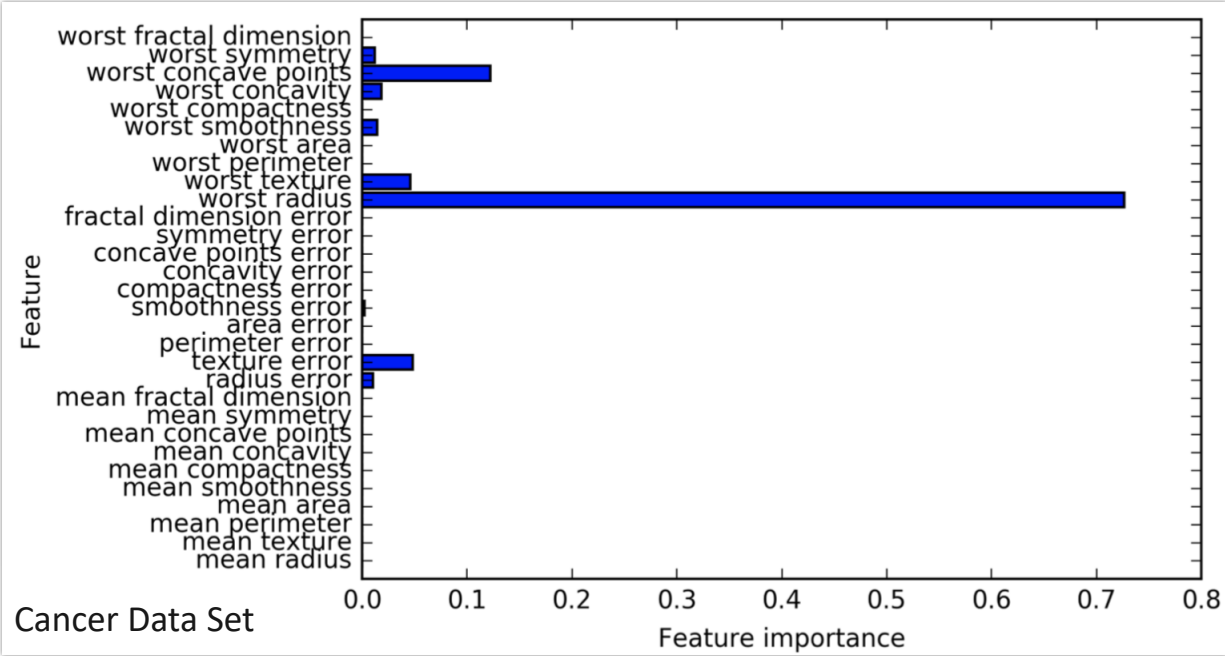
Visualizing the Decision Tree

- The visualization of the tree provides a great in-depth view of how the algorithm makes predictions, and is a good example of a machine learning algorithm that is easily explained to nonexperts.
- However, even with a tree of depth four, as seen here, the tree can become a bit overwhelming.



Feature Importance

- Instead of looking at the whole tree, which can be taxing, there are some useful properties that we can derive to summarize the workings of the tree.
- The most commonly used summary is *feature importance*, which rates how important each feature is for the decision a tree makes.
- It is a number between 0 and 1 for each feature, where 0 means “not used at all” and 1 means “perfectly predicts the target.” The feature importances always sum to 1:



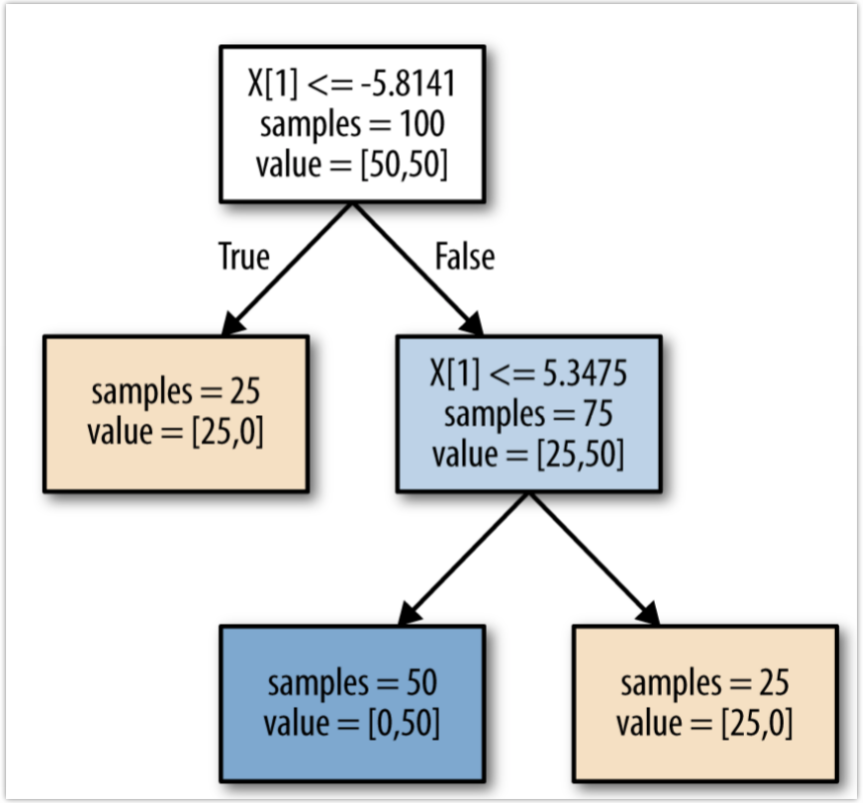
Cancer Data Set

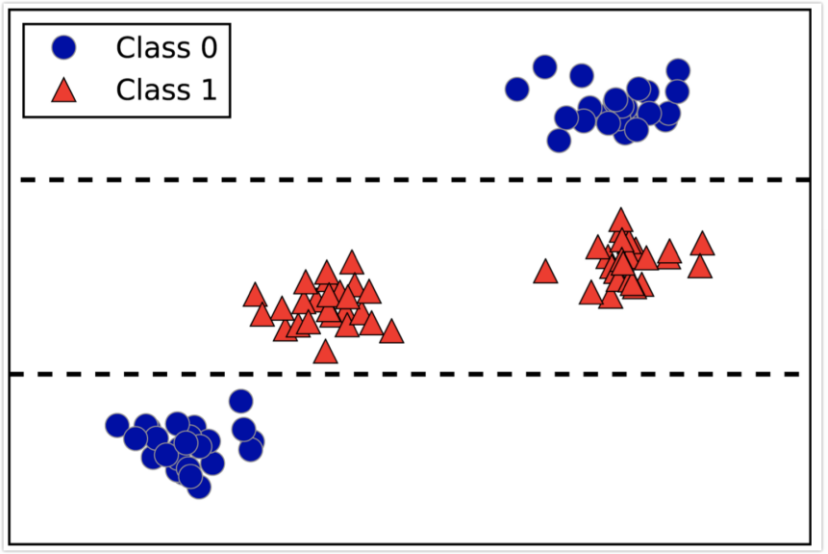
Feature Importance

- Here we see that the feature used in the top split (“worst radius”) is by far the most important feature. This confirms our observation in analyzing the tree that the first level already separates the two classes fairly well.
- However, if a feature has a low “feature_importance”, it doesn’t mean that this feature is uninformative.
  - It only means that the feature was not picked by the tree, likely because another feature encodes the same information.
- In contrast to the coefficients in linear models, feature importances are always positive, and don’t encode which class a feature is indicative of.
  - The feature importances tell us that “worst radius” is important, but not whether a high radius is indicative of a sample being benign or malignant.
  - In fact, there might not be such a simple relationship between features and class

Feature Importance

- The plot shows a dataset with two features and two classes. Here, all the information is contained in X[1], and X[0] is not used at all.





- X[0] is not used in the model, but we cannot conclude that it has no information or it is not important. A high accuracy model can also be obtained by including X[0].

- The relation between X[1] and the output class is not monotonous, meaning we cannot say “a high value of X[1] means class 0, and a low value means class 1” (or vice versa).

Advantages of Decision Trees

- The resulting model can easily be visualized and understood by nonexperts (at least for smaller trees).
- The algorithms are completely invariant to scaling of the data.
  - As each feature is processed separately, and the possible splits of the data don’t depend on scaling,
  - No preprocessing like normalization or standardization of features is needed for decision tree algorithms.
  - In particular, decision trees work well when you have features that are on completely different scales, or a mix of binary and continuous features.
  - Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

Disadvantages of Decision Trees

- The main downside of decision trees is that even with the use of pre-pruning, they tend to overfit and provide poor generalization performance.
- Therefore, in most applications, the ensemble methods are usually used in place of a single decision tree.
- Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.
- The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts.
  - Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node.
  - Such algorithms cannot guarantee to return the globally optimal decision tree.
  - This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.
- Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

Ensemble of Decision Trees

- Ensembles are methods that combine multiple machine learning models to create more powerful models
- There are many models in the machine learning literature that belong to this category,
- But there are two ensemble models that have proven to be effective on a wide range of datasets for classification and regression, both of which use decision trees as their building blocks:
  - Random forests
  - Gradient boosted decision trees

Random Forest

- As we just observed, a main drawback of decision trees is that they tend to overfit the training data.
- Random forests are one way to address this problem. **A random forest is essentially a collection of decision trees, where each tree is slightly different from the others.**
- The idea behind random forests is that each tree might do a relatively good job of predicting, but will likely overfit on part of the data.
- If we build many trees, all of which work well and overfit in different ways, we can reduce the amount of overfitting by averaging their results. This reduction in overfitting, while retaining the predictive power of the trees, can be shown using rigorous mathematics.
- To implement this strategy, we need to build many decision trees. Each tree should do an acceptable job of predicting the target, and should also be different from the other trees.
- Random forests get their name from injecting randomness into the tree building to ensure each tree is different. There are two ways in which the trees in a random forest are randomized:
  - by selecting the data points used to build a tree
  - by selecting the features in each split test

Building Random Forests

- To build a random forest model, you need to decide on the number of trees to build
  - Let's say we want to build 10 trees
  - These trees will be built completely independently from each other, and
  - The algorithm will make different random choices for each tree to make sure the trees are distinct
- To build a tree, we first take what is called a bootstrap sample of our data. That is, from our n data points, we repeatedly draw an example randomly with replacement (meaning the same sample can be picked multiple times), n times.
- This will create a data-set that is as big as the original dataset, but some data points will be missing from it (approximately one third), and some will be repeated.

To illustrate, let's say we want to create a bootstrap sample of the list ['a', 'b', 'c', 'd'].

- A possible bootstrap sample would be ['b', 'd', 'd', 'c'].
- Another possible sample would be ['d', 'a', 'd', 'a'].

Building Random Forests

- Next, a decision tree is built based on this newly created dataset. However, the algorithm is described for the decision tree is slightly modified.
- Instead of looking for the best test for each node, in each node the algorithm randomly selects a subset of the features, and it looks for the best possible test involving one of these features.
- The number of features that are selected is controlled by the "*max_features*" parameter.
- This selection of a subset of features is repeated separately in each node, so that each node in a tree can make a decision using a different subset of the features.

  - The bootstrap sampling leads to each decision tree in the random forest being built on a slightly different dataset.
  - Because of the selection of features in each node, each split in each tree operates on a different subset of features.
  - Together, these two mechanisms ensure that all the trees in the random forest are different.

Building Random Forests

- A critical parameter in this process is "*max_features*":
  - If we set "*max_features*" to "*n_features*", that means that each split can look at all features in the dataset, and no randomness will be injected in the feature selection (the randomness due to the bootstrapping remains, though).
  - If we set "*max_features*" to 1, that means that the splits have no choice at all on which feature to test, and can only search over different thresholds for the feature that was selected randomly.
    - Therefore, a high "*max_features*" means that the trees in the random forest will be quite similar, and they will be able to fit the data easily, using the most distinctive features.
    - A low "*max_features*" means that the trees in the random forest will be quite different, and that each tree might need to be very deep in order to fit the data well.
- To make a prediction using the random forest, the algorithm first makes a prediction for every tree in the forest.
- For regression, we can average these results to get our final prediction.
- For classification, a "soft voting" strategy is used. This means each tree makes a "soft" prediction, providing a probability for each possible output label. The probabilities predicted by all the trees are averaged, and the class with the highest probability is predicted.

Random Forest Pseudocode

Random Forest for Regression or Classification.

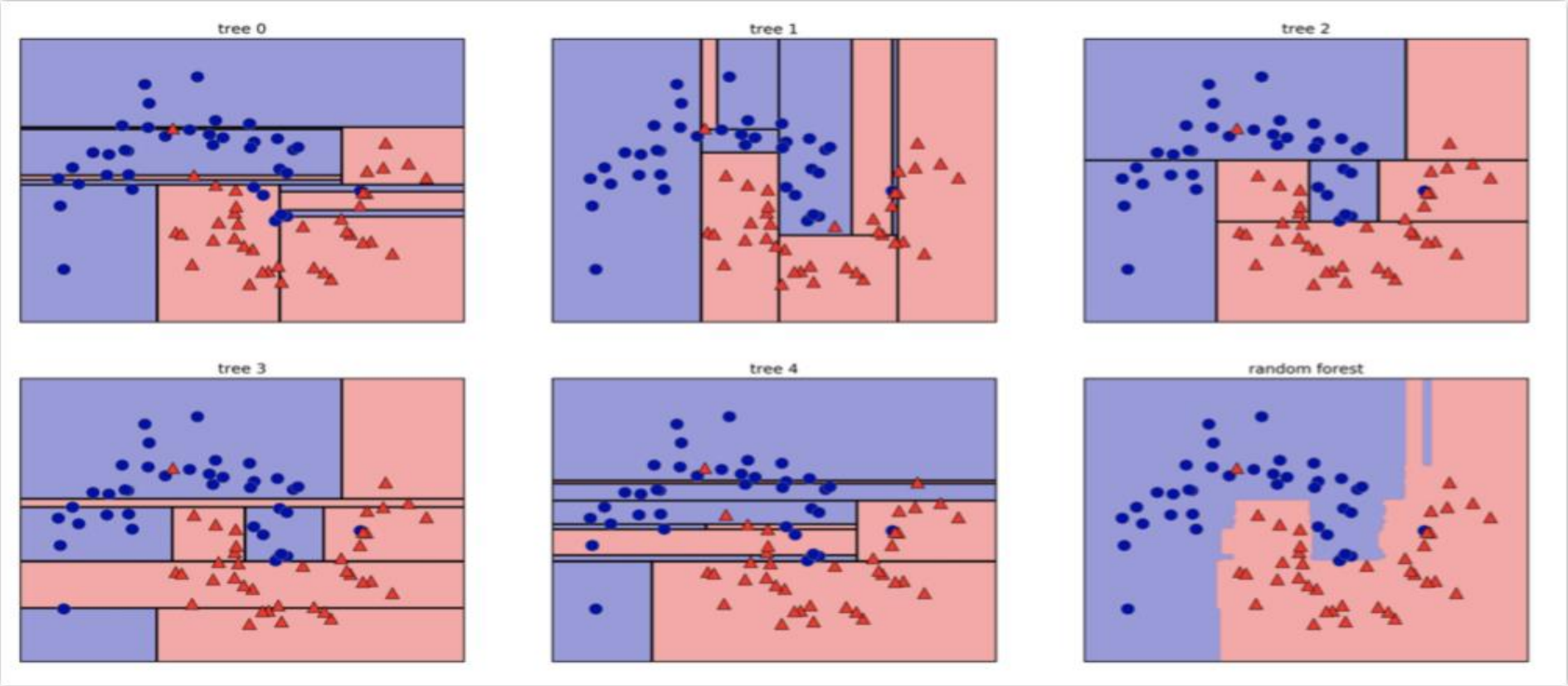
1. For $b = 1$ to B:
   (a) Draw a bootstrap sample $\mathbf{Z}^*$ of size N from the training data.
   (b) Grow a random-forest tree T_b to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size n_{min} is reached.
      i. Select m variables at random from the p variables.
      ii. Pick the best variable/split-point among the m.
      iii. Split the node into two daughter nodes.
2. Output the ensemble of trees $\{T_b\}_1^B$.

To make a prediction at a new point x:

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

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

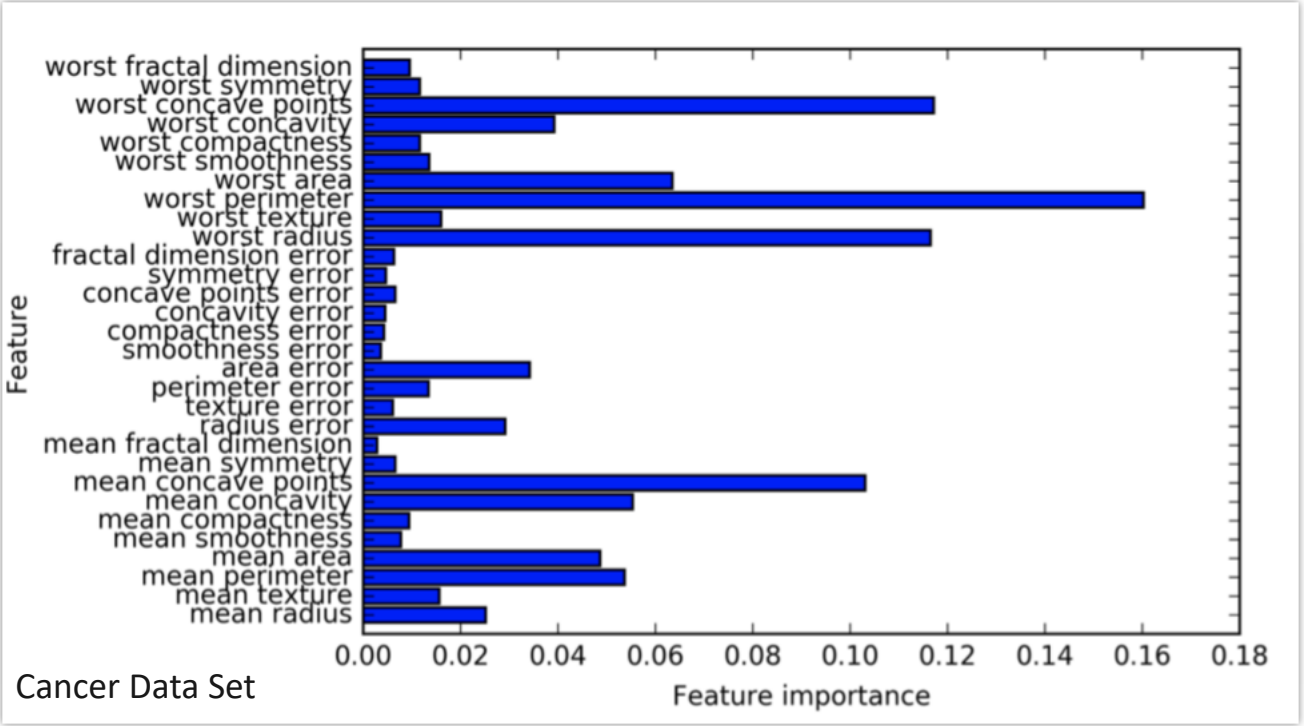
Two Moons Data



- The random forest overfits less than any of the trees individually, and provides a much more intuitive decision boundary.
- In any real application, it is used many more trees (often hundreds or thousands), leading to even smoother boundaries.

Feature Importance

- Similarly to the decision tree, the random forest provides feature importances, which are computed by aggregating the feature importances over the trees in the forest.
- Typically, the feature importances provided by the random forest are more reliable than the ones provided by a single tree.



Cancer Data Set

- As you can see, the random forest gives nonzero importance to many more features than the single tree.
- Similarly to the single decision tree, the random forest also gives a lot of importance to the “worst radius” feature, but it actually chooses “worst perimeter” to be the most informative feature overall.
- The randomness in building the random forest forces the algorithm to consider many possible explanations, the result being that the random forest captures a much broader picture of the data than a single tree.

Strengths, Weaknesses, and Parameters

- Random forests for regression and classification are currently among the most widely used machine learning methods. They are very powerful, often work well without heavy tuning of the parameters, and don’t require scaling of the data.
- Essentially, random forests share all of the benefits of decision trees, while making up for some of their deficiencies. One reason to still use decision trees is if you need a compact representation of the decision-making process.
- It is basically impossible to interpret tens or hundreds of trees in detail, and trees in random forests tend to be deeper than decision trees (because of the use of feature subsets). Therefore, if you need to summarize the prediction making in a visual way to nonexperts, a single decision tree might be a better choice.
- While building random forests on large data- sets might be somewhat time consuming, it can be parallelized across multiple CPU cores within a computer easily.
- You should keep in mind that random forests, by their nature, are random, and setting different random states (or not setting the random_state at all) can drastically change the model that is built.
- The more trees there are in the forest, the more robust it will be against the choice of random state. If you want to have reproducible results, it is important to fix the random_state.

Strengths, Weaknesses, and Parameters

- Random forests don't tend to perform well on very high dimensional, sparse data, such as text data. For this kind of data, linear models might be more appropriate.
- Random forests require more memory and are slower to train and to predict than linear models. If time and memory are important in an application, it might make sense to use a linear model instead
- The important parameters to adjust are "*n_estimators*", "*max_features*", and possibly pre-pruning options like "*max_depth*".
- For "*n_estimators*", larger is always better. Averaging more trees will yield a more robust ensemble by reducing overfitting.
  - However, there are diminishing returns, and more trees need more memory and more time to train.
- "*max_features*" determines how random each tree is, and a smaller "*max_features*" reduces overfitting.
- In general, it's a good rule of thumb to use the default values:
  - *max_features*=$\sqrt{n\_features}$ for classification and *max_features*=$\log_2(n\_features)$ for regression.
  - Adding "*max_features*" or "*max_leaf_nodes*" might sometimes improve performance. It can also drastically reduce space and time requirements for training and prediction.

References

- C. Rudin, Prediction: Machine Learning And Statistics, Lecture Notes.
- T. Hastie and R. Tibshirani, Statistical Learning, Lecture Note.
- E. Eaton, Decision Trees, Lecture Notes.
- J. Friedman, T. Hastie, and R. Tibshirani, “The Elements of Statistical Learning”, 2008.