Trees

CS 189

Alexei Efros

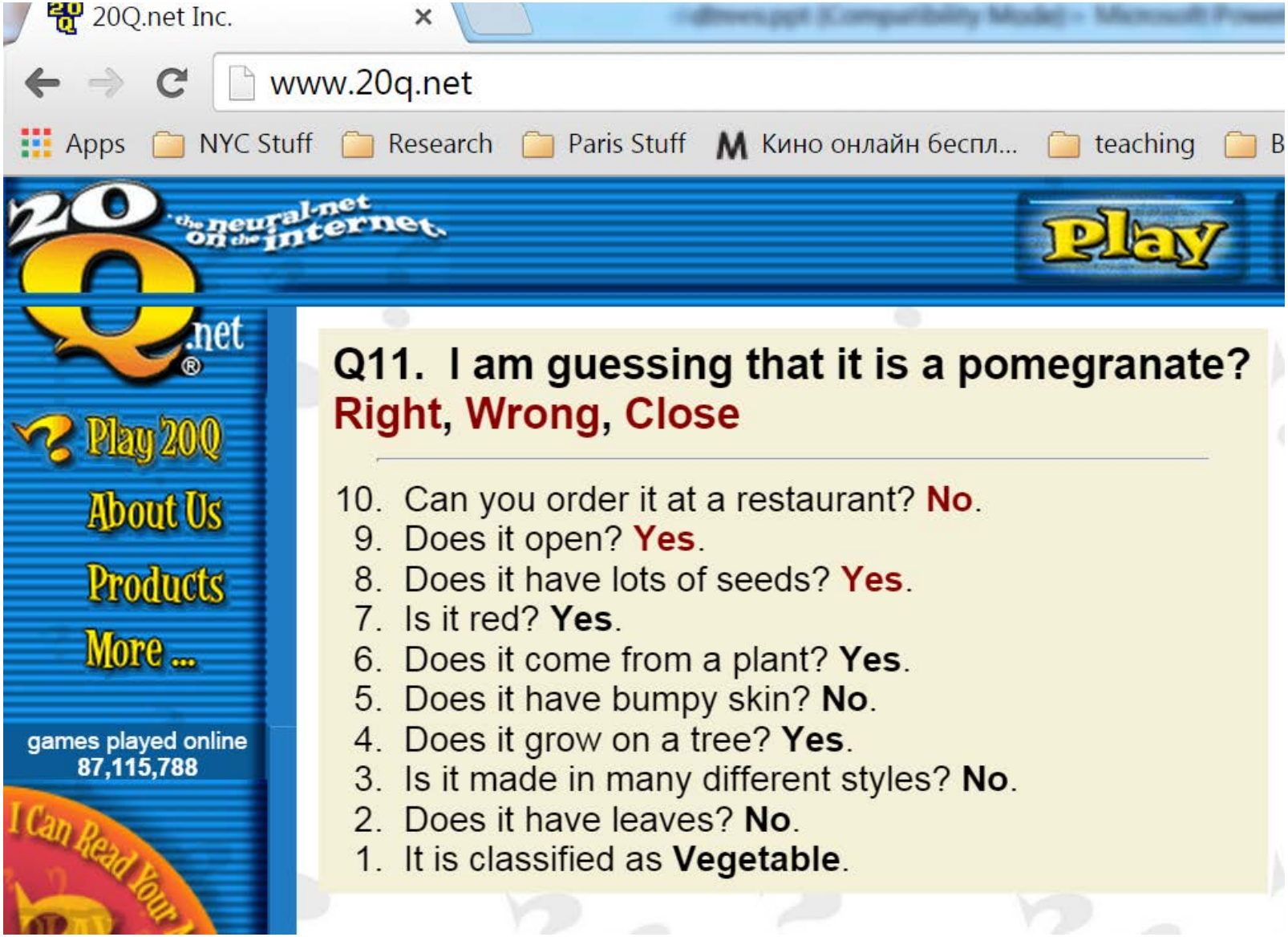
Fall 2015

Nearest Neighbors: Reducing Computational Cost

- Nearest-neighbors has O(N) complexity
  - Infeasible for large datasets

- Can we speed it up?
  - Think of guessing a number between 1 and 10…

20 Questions Game



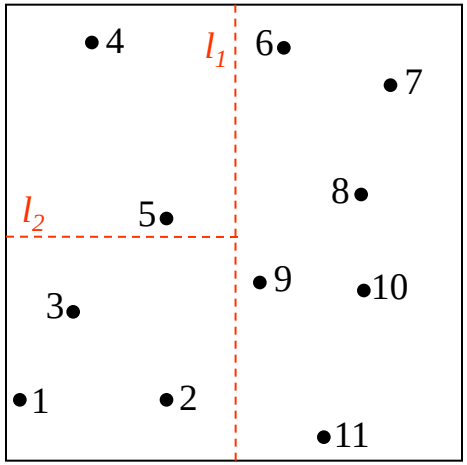
K-d tree

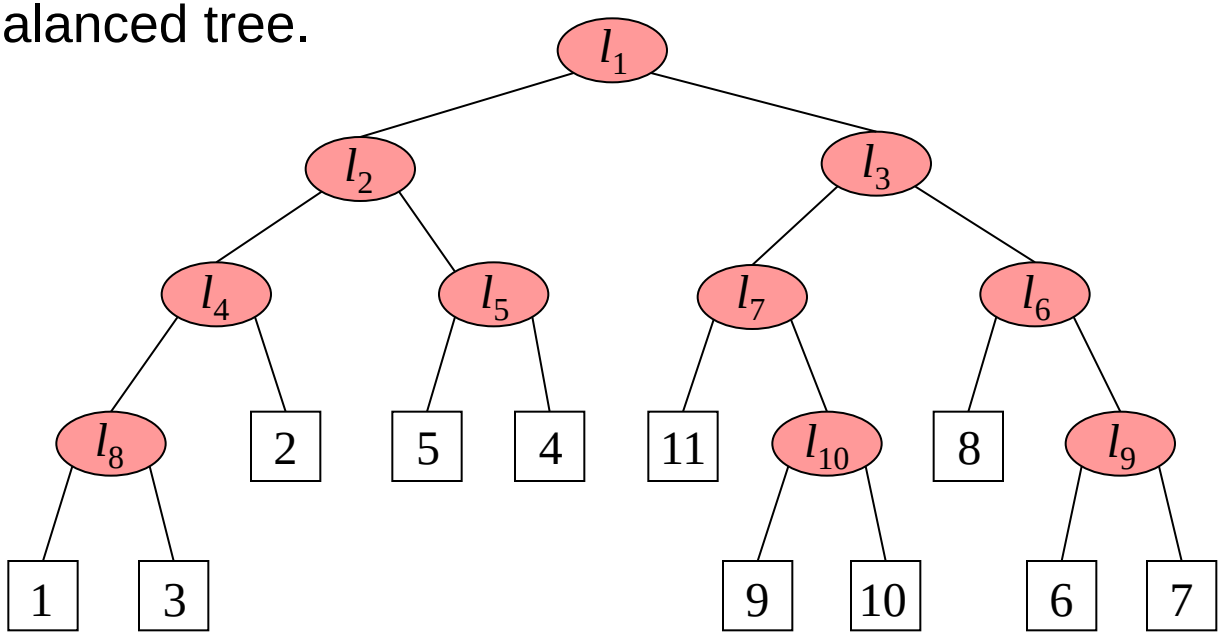
• K-d tree is a binary tree data structure for organizing a set of points in a K-dimensional space.

• Each internal node is associated with an axis aligned hyper-plane splitting its associated points into two sub-trees.

• Dimensions with high variance are chosen first.

• Position of the splitting hyper-plane is chosen as the mean/median of the projected points – balanced tree.

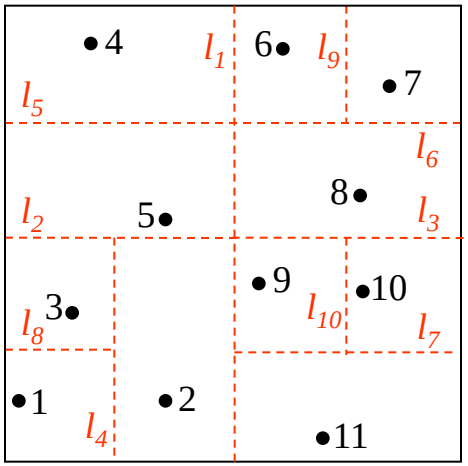


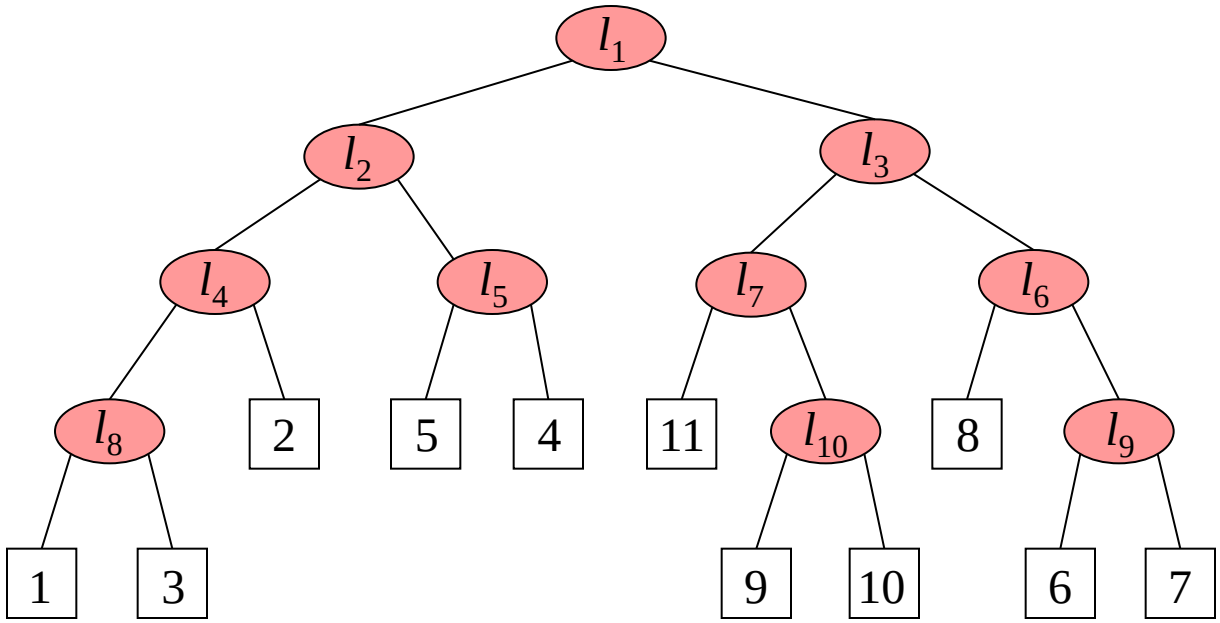


Images: Anna Atramentov

K-d tree construction

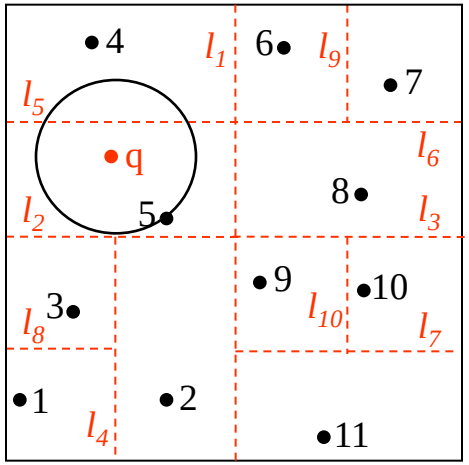
Simple 2D example

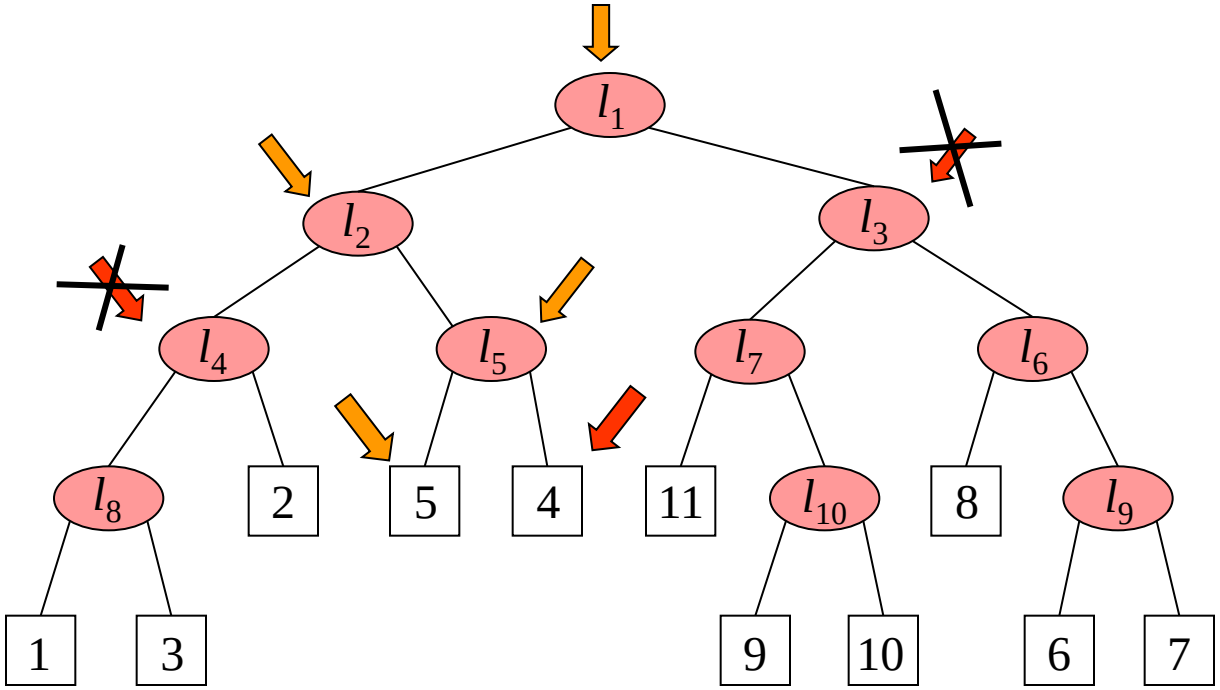




Slide credit: Anna Atramentov

K-d tree query





Slide credit: Anna Atramentov

K-d tree: Backtracking

Backtracking is necessary as the true nearest neighbor may not lie in the query cell.

But in some cases, almost all cells need to be inspected.

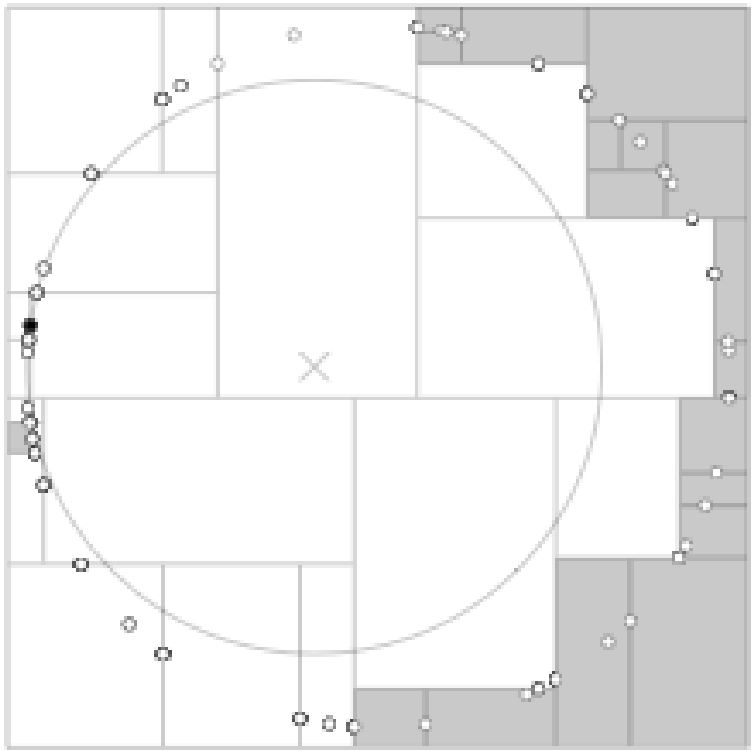


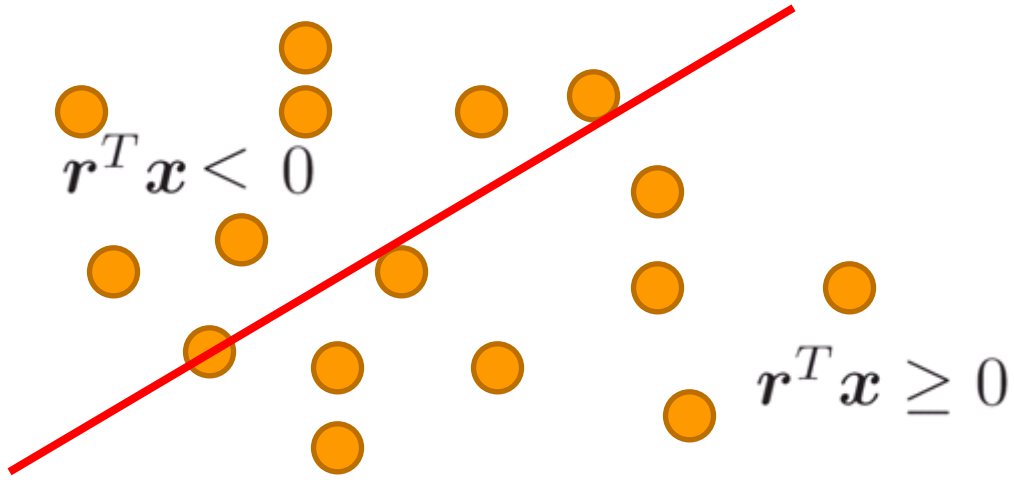
Figure 6.6

A bad distribution which forces almost all nodes to be inspected.

Figure: A. Moore

Do we need axis-aligned hyperplanes?

Normal unit vector *r* defines is a hyperplane separating the space

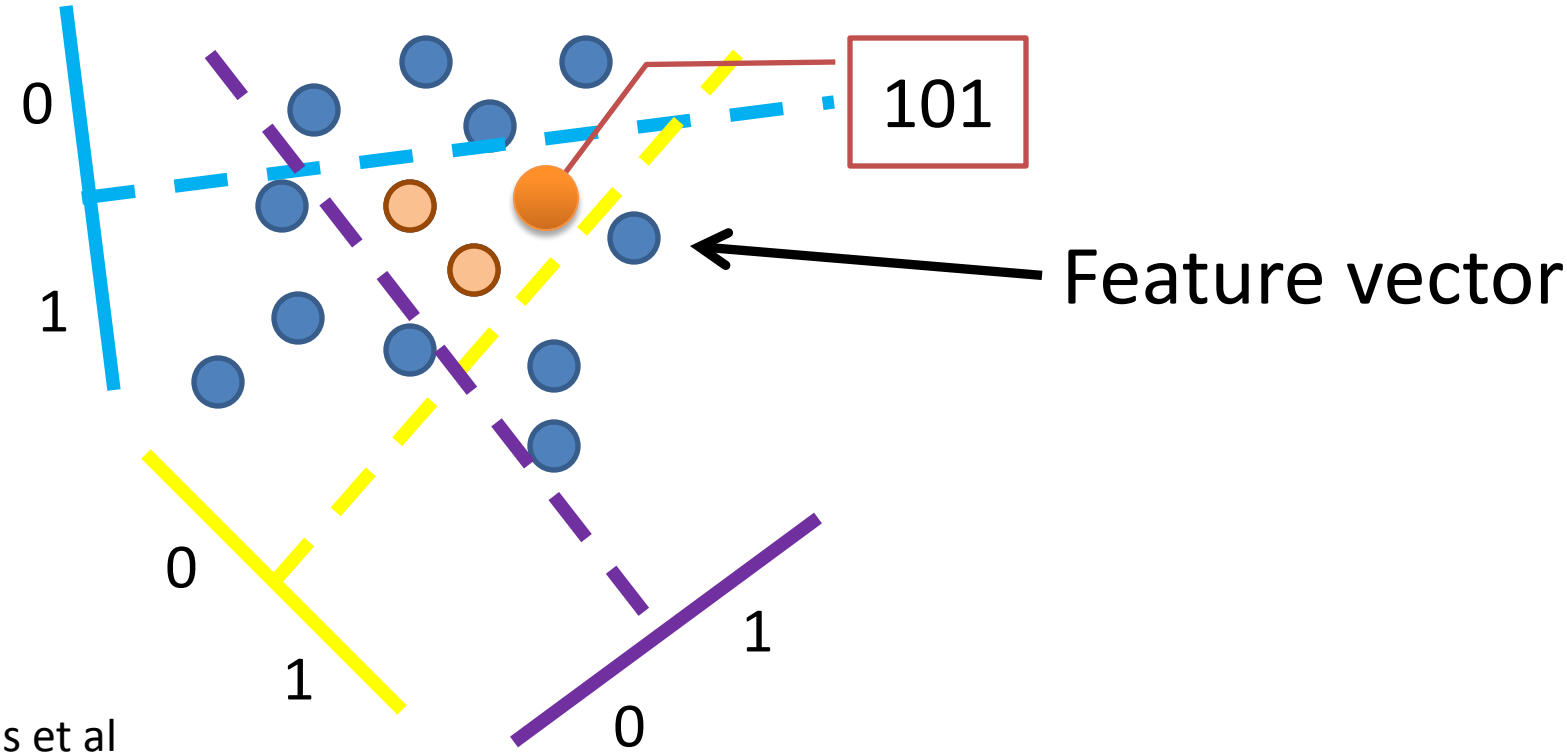


For any point x, define:

$$h_{\boldsymbol{r}}(\boldsymbol{x}) = \begin{cases} 1, & \text{if } \boldsymbol{r}^T\boldsymbol{x} \geq 0 \\ 0, & \text{otherwise} \end{cases}$$

Hashing by Random Projections

- Take random projections of data $r^T x$
- Quantize each projection with few bits



Fergus et al

Locality Sensitive Hashing

- The basic idea behind LSH is to project the data into a low-dimensional binary (Hamming) space; that is, each data point is mapped to a b-bit vector, called the *hash key*.
- Unlike normal hashing, here we want our hashes to cluster – create collisions
- Each hash function h must satisfy the locality sensitive hashing property:

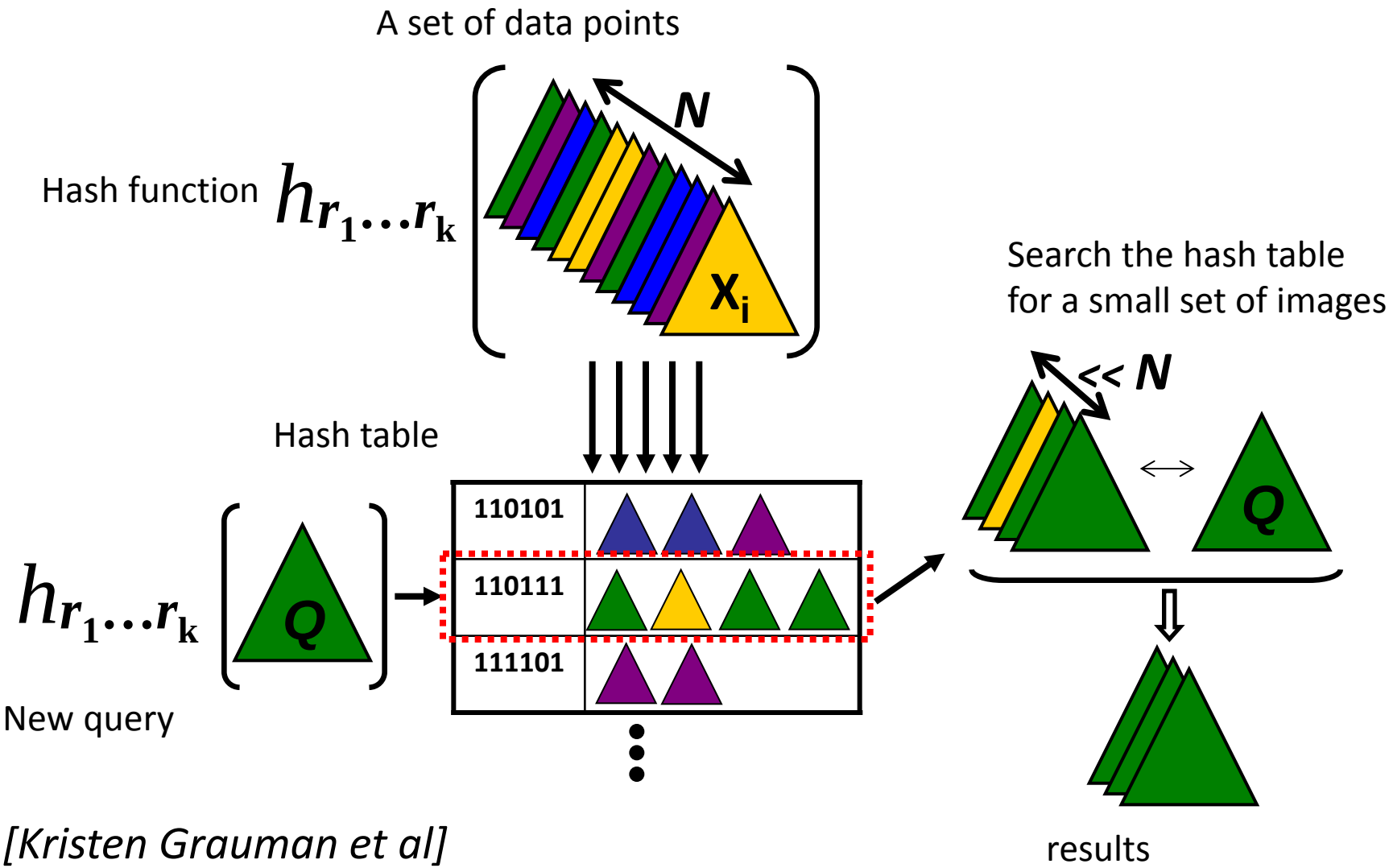
$$\Pr[h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)] = \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

  - Where $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in [0, 1]$ is the similarity function. In our case:

$$Pr[h(u) = h(v)] = 1 - \frac{\theta(u, v)}{\pi}$$

Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality-Sensitive Hashing Scheme Based on p-Stable Distributions. In *SOCG, 2004.*

Approximate Nearest-Neighbor Search



[Kristen Grauman et al]

Decision Trees

Nearest Neighbors

+ Nonlinear: arbitrary complex decision boundary possible

+ Non-parametric: can absorb unlimited amounts of data

+ / - Can decide on classes/labels at run-time

- This makes it dead slow!
- Even in kd-tree / LSH, we looked at the data *x*, but not the labels *y*

– Easily confused by irrelevant features

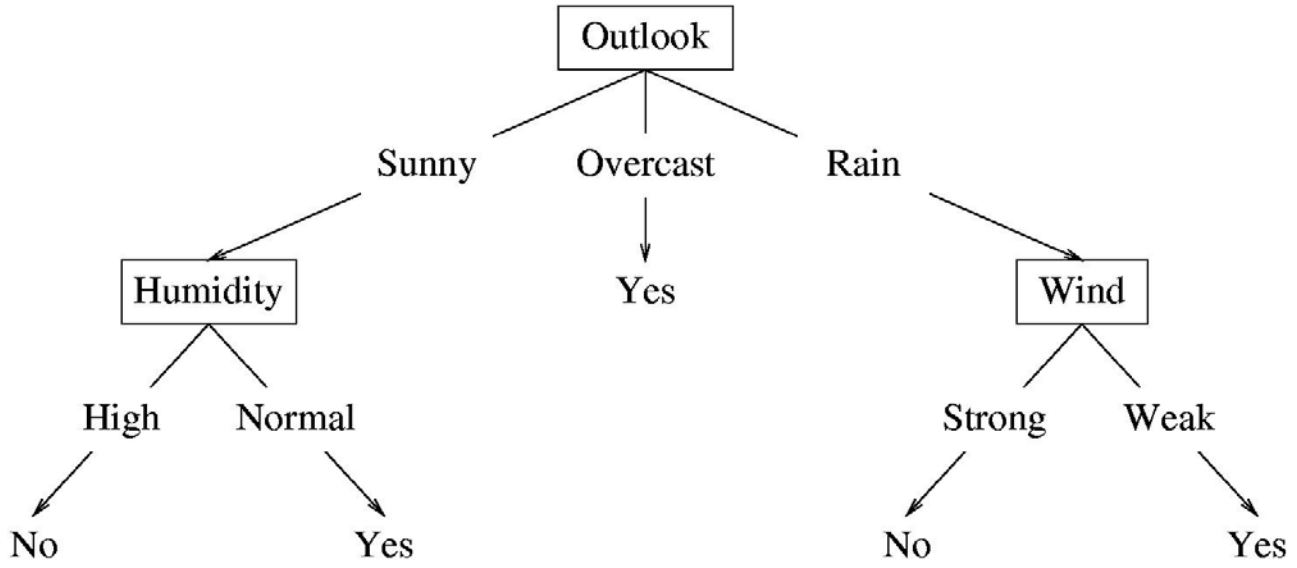
Decision Trees

- Let’s bring back labels!
  - Keep most benefits
  - but much more efficient

Let’s make a decision tree!

Decision Tree Hypothesis Space

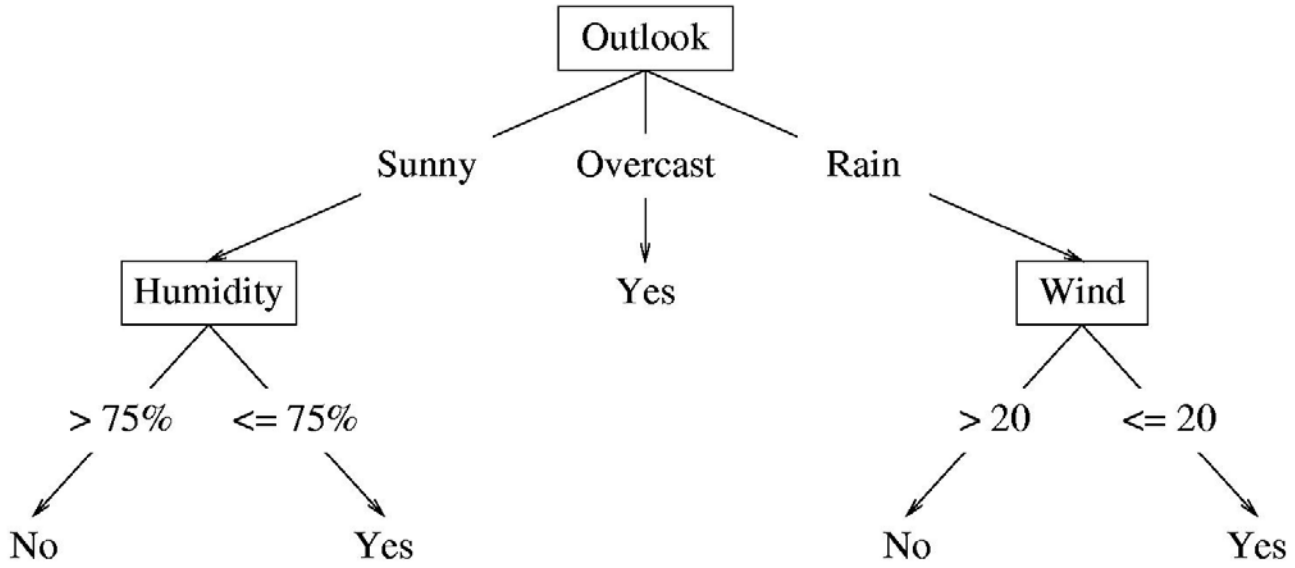
- **Internal nodes** test the value of particular features x_j and branch according to the results of the test.
- **Leaf nodes** specify the class $h(\mathbf{x})$.



Suppose the features are **Outlook** (x_1), **Temperature** (x_2), **Humidity** (x_3), and **Wind** (x_4). Then the feature vector $\mathbf{x} = (Sunny, Hot, High, Strong)$ will be classified as **No**. The **Temperature** feature is irrelevant.

Decision Tree Hypothesis Space

If the features are continuous, internal nodes may test the value of a feature against a threshold.



Benefits of Decision Trees

- works well with categorical features
- interpretable result
- Performs feature selection

- What would the decision regions look like?

Building a Decision Tree

1. Decide on the best attribute on which to split the data
2. Long live recursion!!!

Learning Algorithm for Decision Trees

The same basic learning algorithm has been discovered by many people independently:

GrowTree(S)
if ($y = 0$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(0)
else if ($y = 1$ for all $\langle \mathbf{x}, y \rangle \in S$) **return** new leaf(1)
else
    choose best attribute x_j
    $S_0 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;
    $S_1 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;
    return new node(x_j, GrowTree(S_0), GrowTree(S_1))

Decision trees for Classification

- Training time = find good set of “questions”
  - Construct the tree, i.e. pick the questions at each node of the tree. Typically done so as to make each of the child nodes “purer” (lower entropy). Each leaf node will be associated with a set of training examples
- Test time
  - Evaluate the tree by sequentially evaluating questions, starting from the root node. Once a particular leaf node is reached, we predict the class to be the one with the most examples (from training set) at this node.

Choosing the Best Attribute

One way to choose the best attribute is to perform a 1-step lookahead search and choose the attribute that gives the lowest error rate on the training data.

ChooseBestAttribute(S)

choose j to minimize J_j, computed as follows:

$S_0 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 0$;

$S_1 =$ all $\langle \mathbf{x}, y \rangle \in S$ with $x_j = 1$;

$y_0 =$ the most common value of y in S_0

$y_1 =$ the most common value of y in S_1

$J_0 =$ number of examples $\langle \mathbf{x}, y \rangle \in S_0$ with $y \neq y_0$

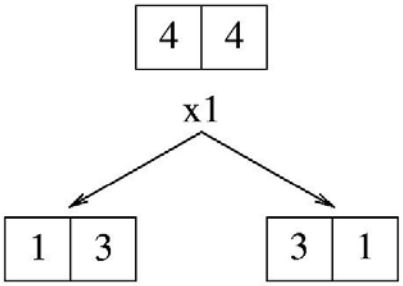
$J_1 =$ number of examples $\langle \mathbf{x}, y \rangle \in S_1$ with $y \neq y_1$

$J_j = J_0 + J_1$ (total errors if we split on this feature)

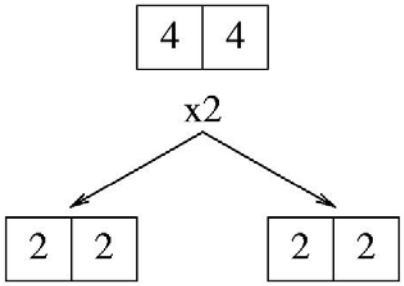
return j

Choosing the Best Attribute—An Example

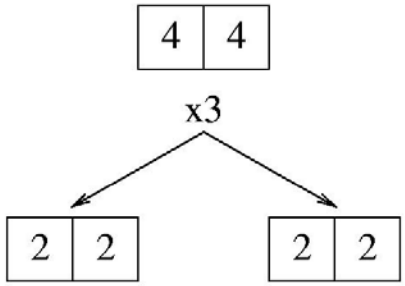
x_1	x_2	x_3	y
0	0	0	1
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	0



J=2



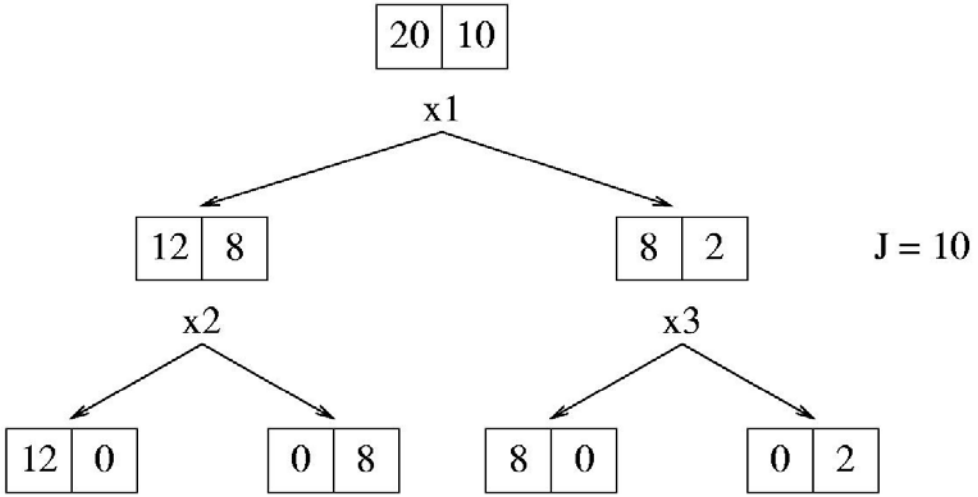
J=4



J=4

Choosing the Best Attribute (3)

Unfortunately, this measure does not always work well, because it does not detect cases where we are making "progress" toward a good tree.



A Better Heuristic From Information Theory

Let V be a random variable with the following probability distribution:

$P(V = 0)$	$P(V = 1)$
0.2	0.8

The *surprise*, $S(V = v)$ of each value of V is defined to be

$$S(V = v) = -\lg P(V = v).$$

An event with probability 1 gives us zero surprise.

An event with probability 0 gives us infinite surprise!

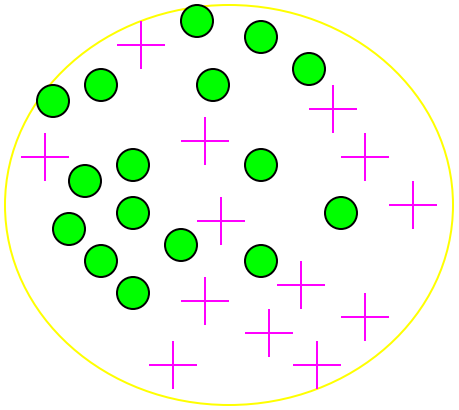
It turns out that the surprise is equal to the number of bits of information that need to be transmitted to a recipient who knows the probabilities of the results.

This is also called the *description length* of $V = v$.

Fractional bits only make sense if they are part of a longer message (e.g., describe a whole sequence of coin tosses).

Entropy: Average Surprise

- Entropy H = $\sum_i - p_i \log_2 p_i$



p_i is the probability of class i

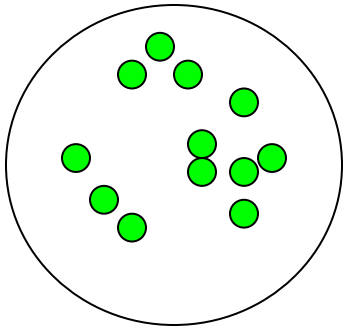
Compute it as the proportion of class i in the set.

in 2-Class case:

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

  We are done!

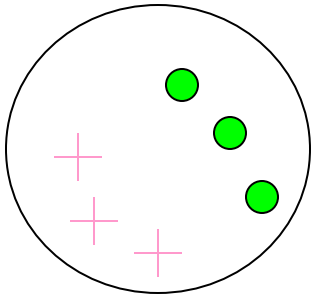
Minimum entropy



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

  Better start splitting

Maximum entropy

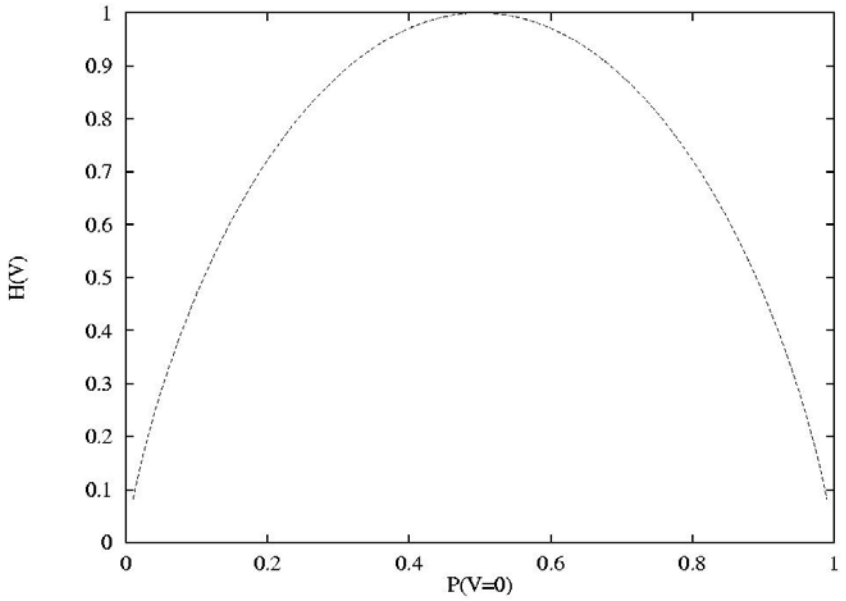


Entropy

The *entropy* of V, denoted $H(V)$ is defined as follows:

$$H(V) = \sum_{v=0}^{1} -P(H = v) \lg P(H = v).$$

This is the average surprise of describing the result of one "trial" of V (one coin toss).



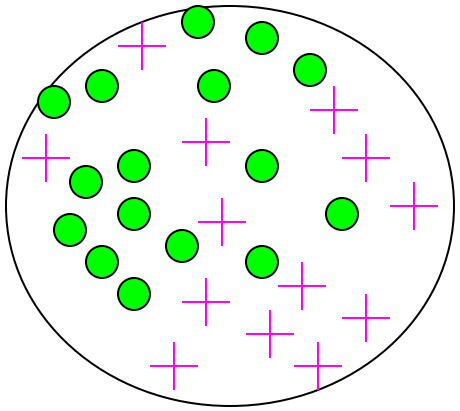
Entropy can be viewed as a measure of uncertainty.

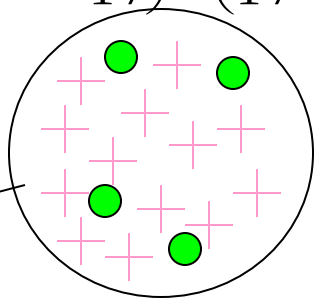
How much can we gain from splitting?

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

child entropy $-\left(\frac{13}{17}\cdot\log_2\frac{13}{17}\right)-\left(\frac{4}{17}\cdot\log_2\frac{4}{17}\right)=0.787$

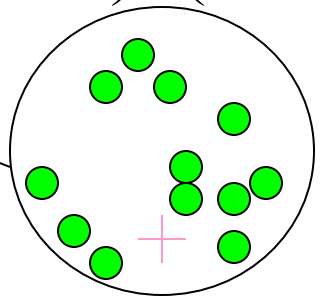
Entire population (30 instances)





17 instances

child entropy $-\left(\frac{1}{13}\cdot\log_2\frac{1}{13}\right)-\left(\frac{12}{13}\cdot\log_2\frac{12}{13}\right)=0.391$



13 instances

parent entropy $-\left(\frac{14}{30}\cdot\log_2\frac{14}{30}\right)-\left(\frac{16}{30}\cdot\log_2\frac{16}{30}\right)=0.996$

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

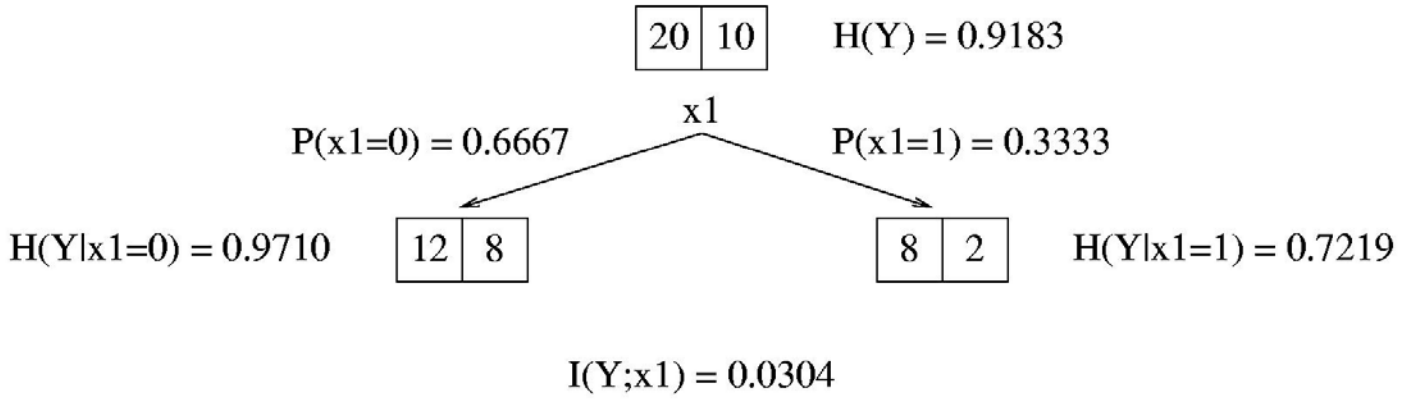
Information Gain = 0.996 - 0.615 = 0.38

Information Gain

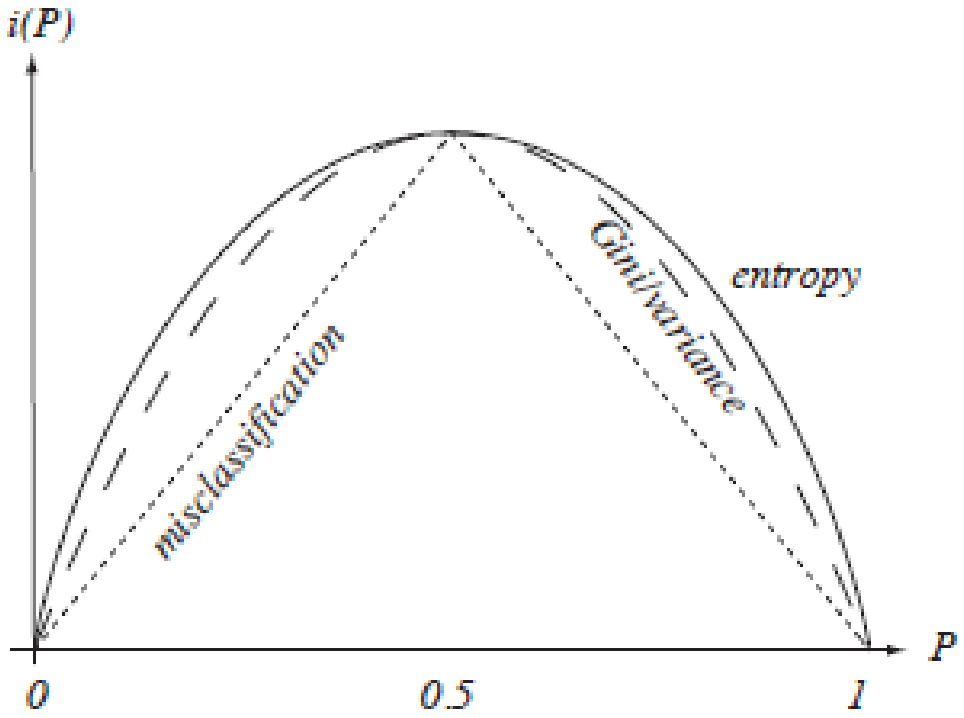
Now consider two random variables A and B that are not necessarily independent. The *mutual information* between A and B is the amount of information we learn about B by knowning the value of A (and vice versa—it is symmetric). It is computed as follows:

$$I(A;B) = \text{H(A)} \quad - \sum_b P(B=b) \cdot H(A|B=b)$$

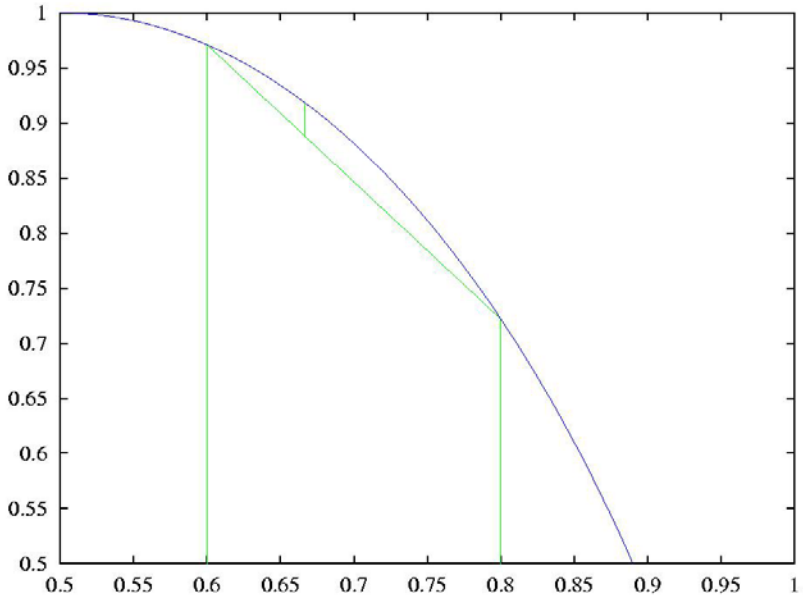
In particular, consider the class Y of each training example and the value of feature x_1 to be random variables. Then the mutual information quantifies how much x_1 tells us about the value of the class Y.



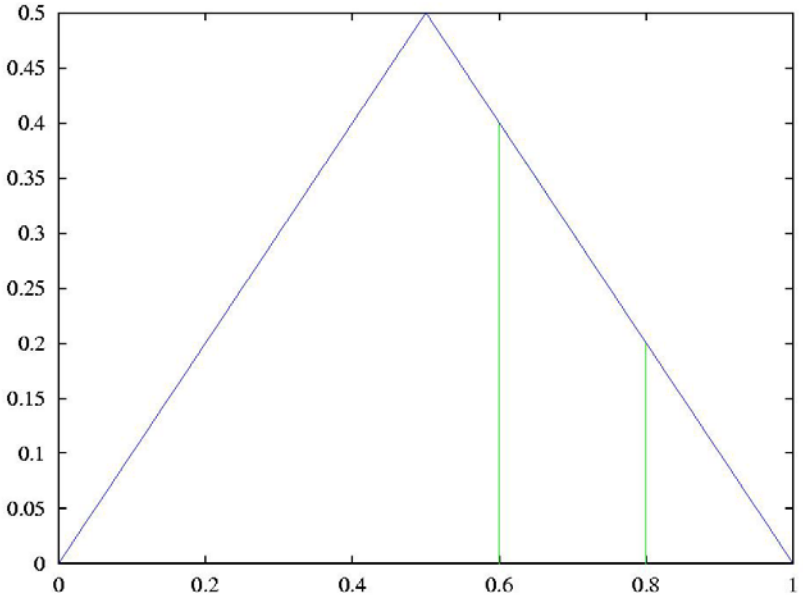
Misclassification Error vs. Entropy



Visualizing Heuristics



Entropy



Absolute Error

Mutual information works because it is a convex measure.

Non-binary Features

- Features with multiple discrete values
  - Multiway split
    - Need to normalize Information Gain, else it will always prefer multiway splits
  - One-vs-all split
    - Makes a cascade of binary splits
  - Group values into two disjoint subsets
- Real-valued Features
  - Consider a threshold split at each observed value of the feature
    - Reduces to previous problem

Multivariate Splits

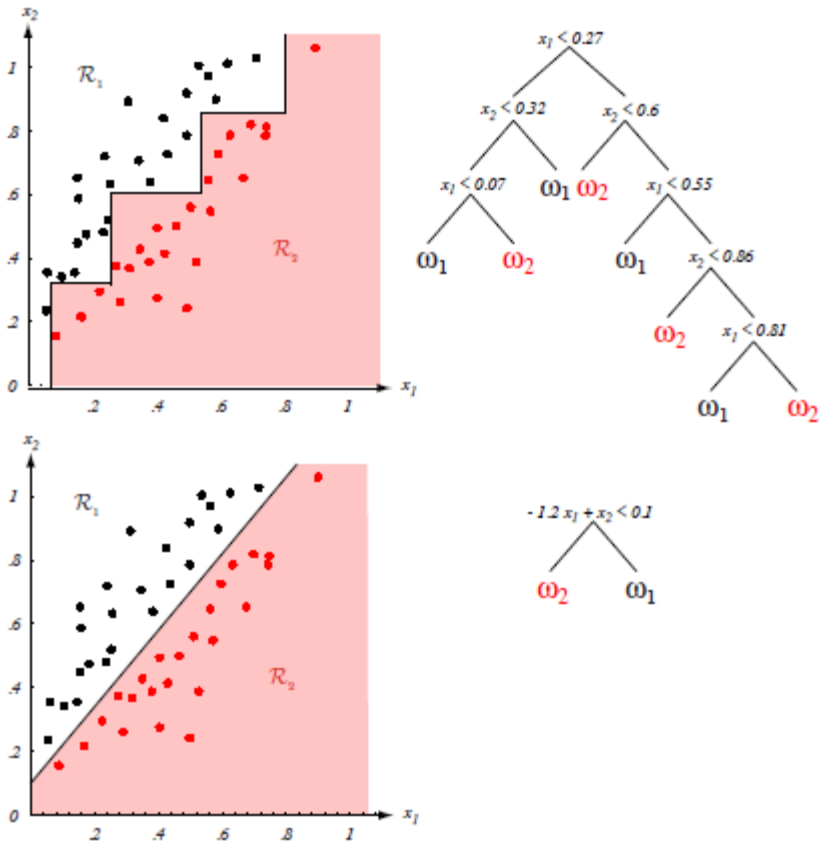


Figure 8.5: If the class of node decisions does not match the form of the training data, a very complicated decision tree will result, as shown at the top. Here decisions are parallel to the axes while in fact the data is better split by boundaries along another direction. If however "proper" decision forms are used (here, linear combinations of the features), the tree can be quite simple, as shown at the bottom.

Unknown Attribute Values

What if some examples are missing values of A?

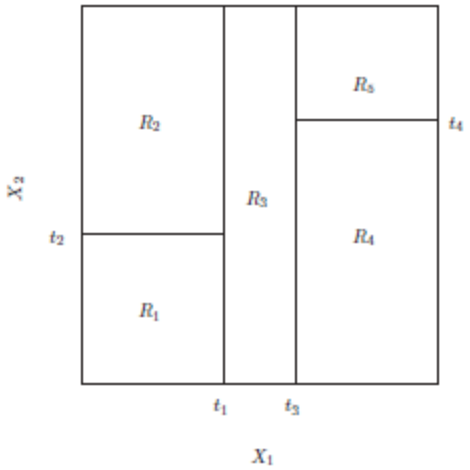
Use training example anyway, sort through tree

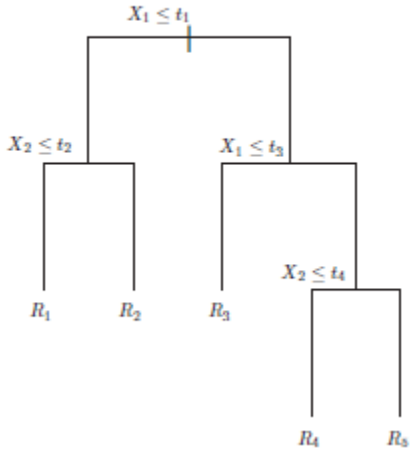
- If node n tests A, assign most common value of A among other examples sorted to node n
- Assign most common value of A among other examples with same target value
- Assign probability p_i to each possible value v_i of A
  Assign fraction p_i of example to each descendant in tree

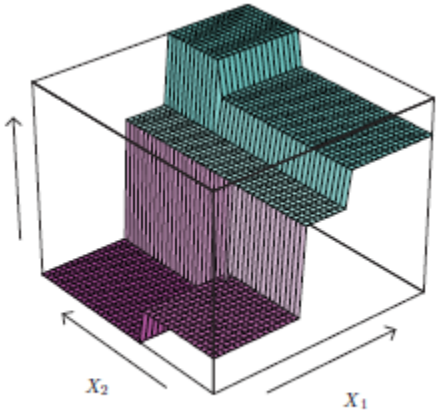
Classify new examples in same fashion

Classification vs. Regression Trees

- Classification tree: $x \rightarrow \{0,1\}$
- Regression tree: $x \rightarrow y \in R$







  - Instead of minimizing entropy, we minimize sum of variances after the split:

$$\sum_{i:\, x_i \in R_1(j,s)} (y_i - \hat{y}_{R_1})^2 + \sum_{i:\, x_i \in R_2(j,s)} (y_i - \hat{y}_{R_2})^2$$

Overfitting in Decision Tree Learning