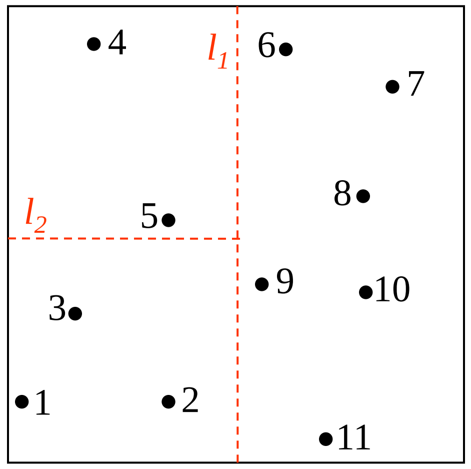
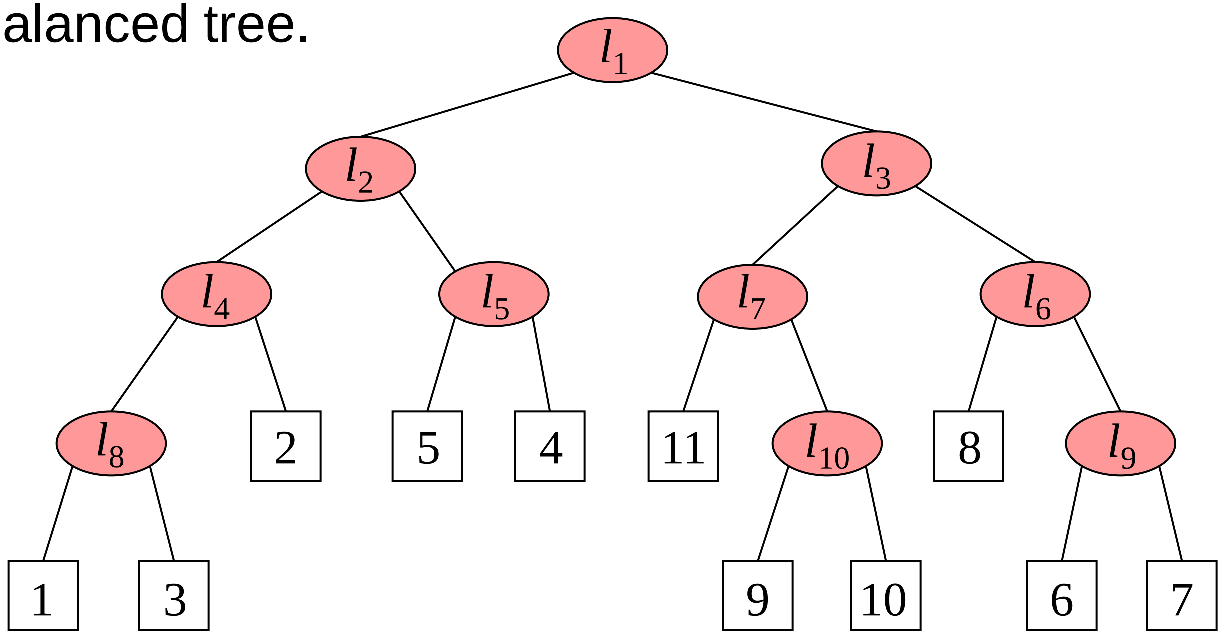
Reducing Computational Cost

- Nearest-neighbors has O(N) complexity
  - Infeasible for large datasets
- Can we speed it up?
  - Think of guessing a number between 1 and 10…

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.

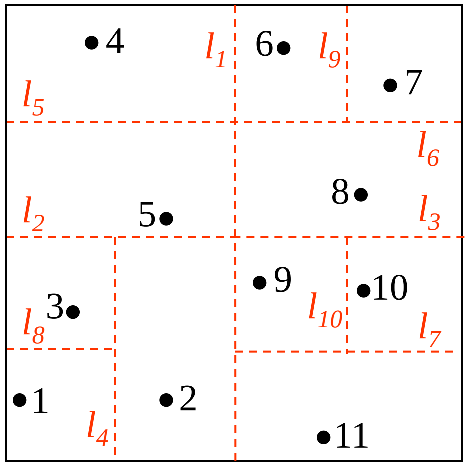


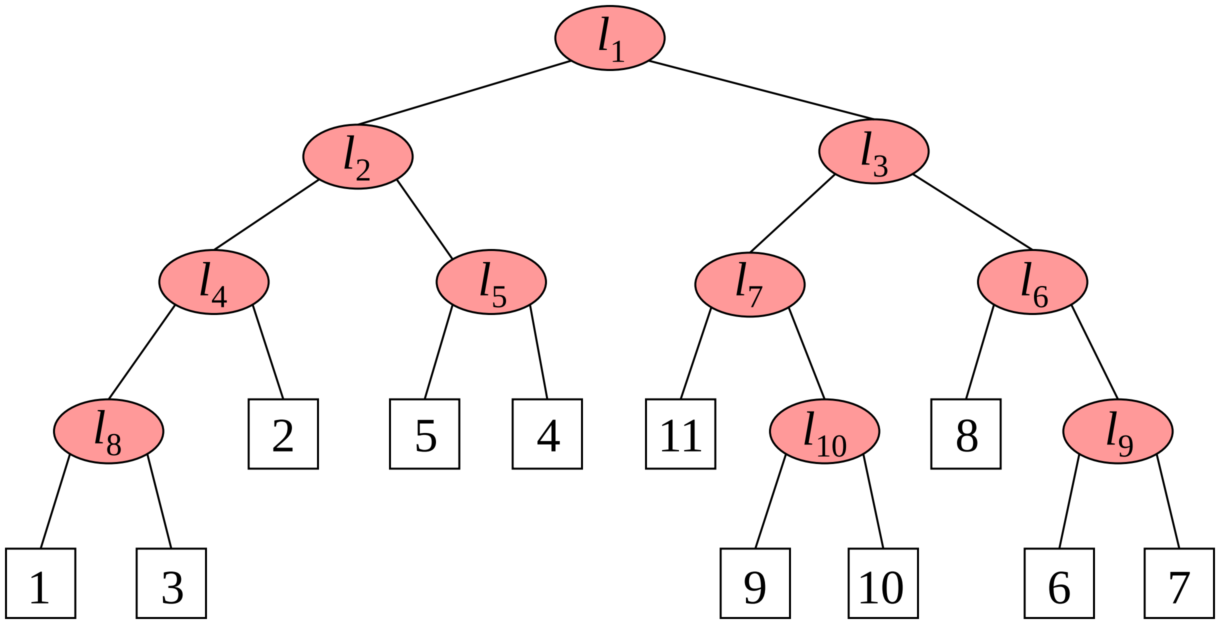


Images: Anna Atramentov

K-d tree construction

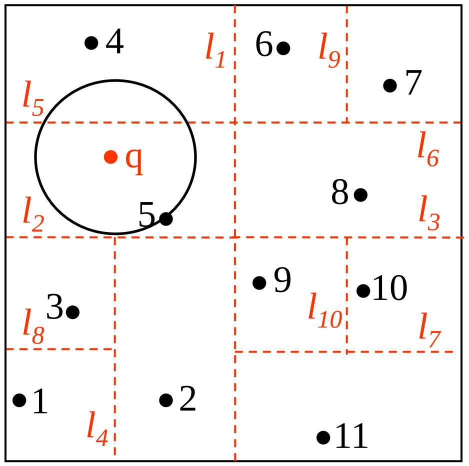
Simple 2D example

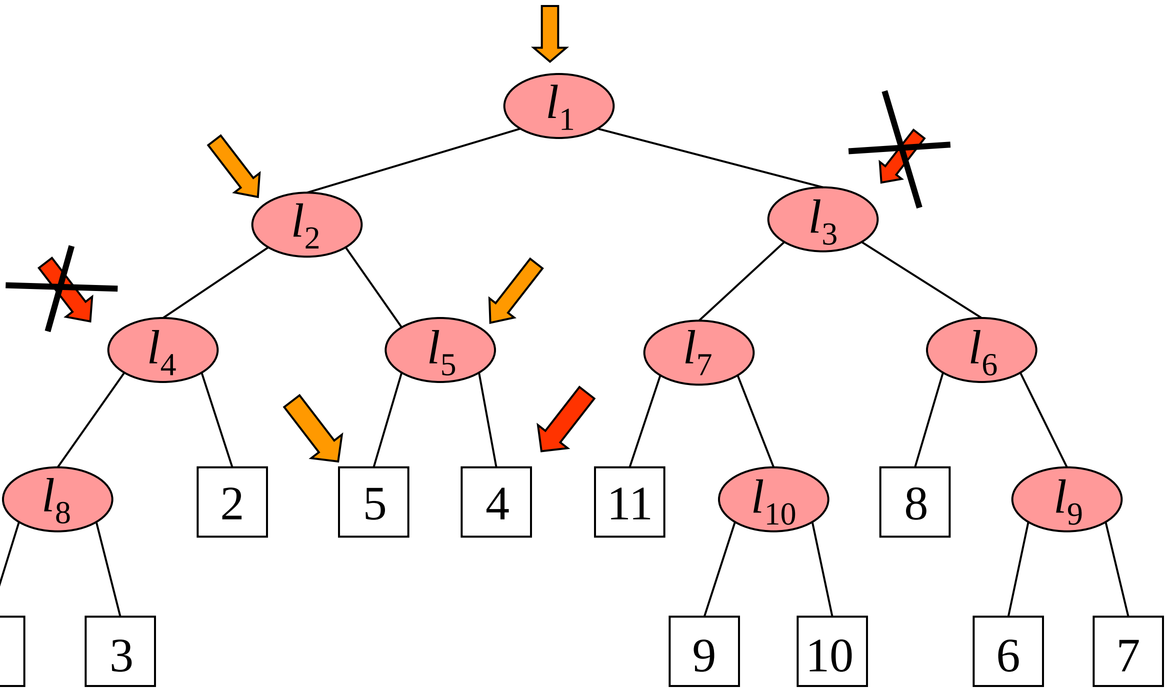




Slide credit: Anna Atramentov

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.

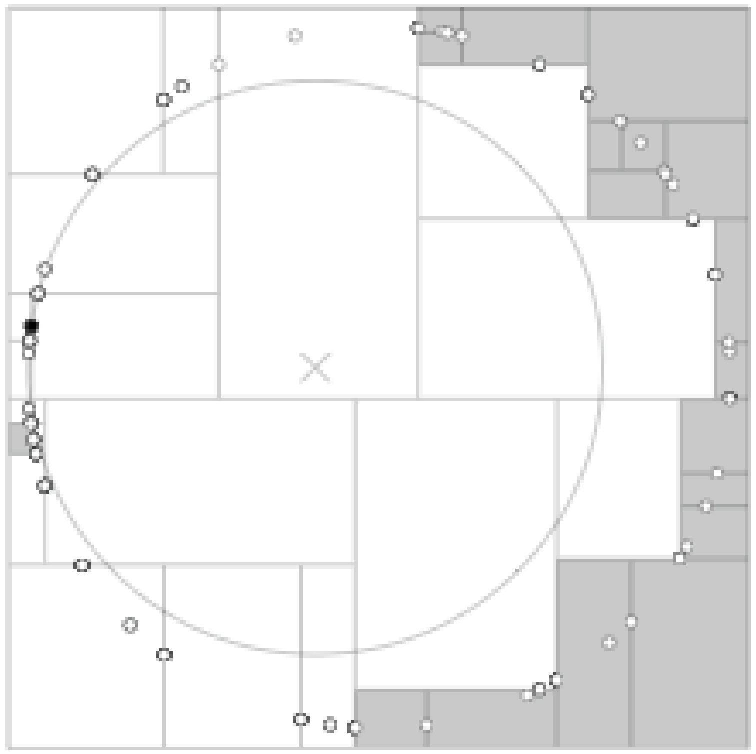


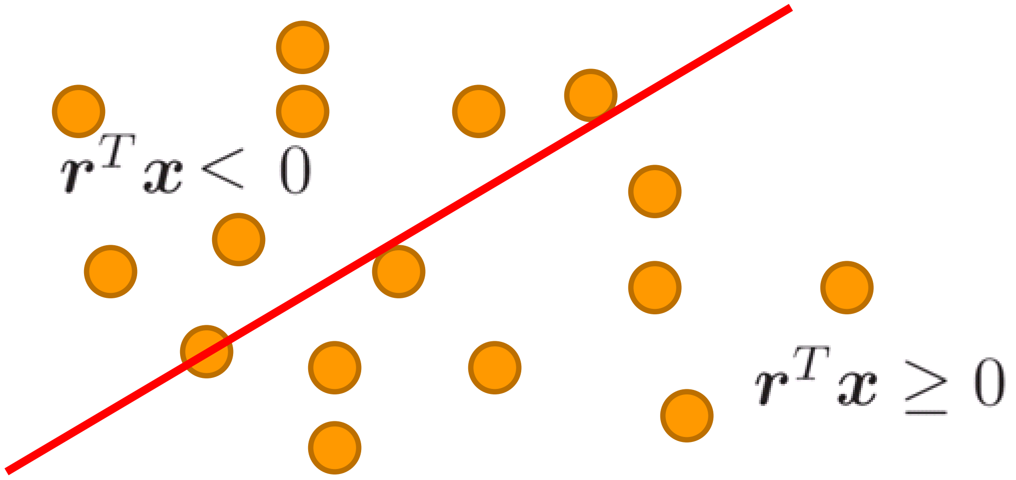
Figure 6.6

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

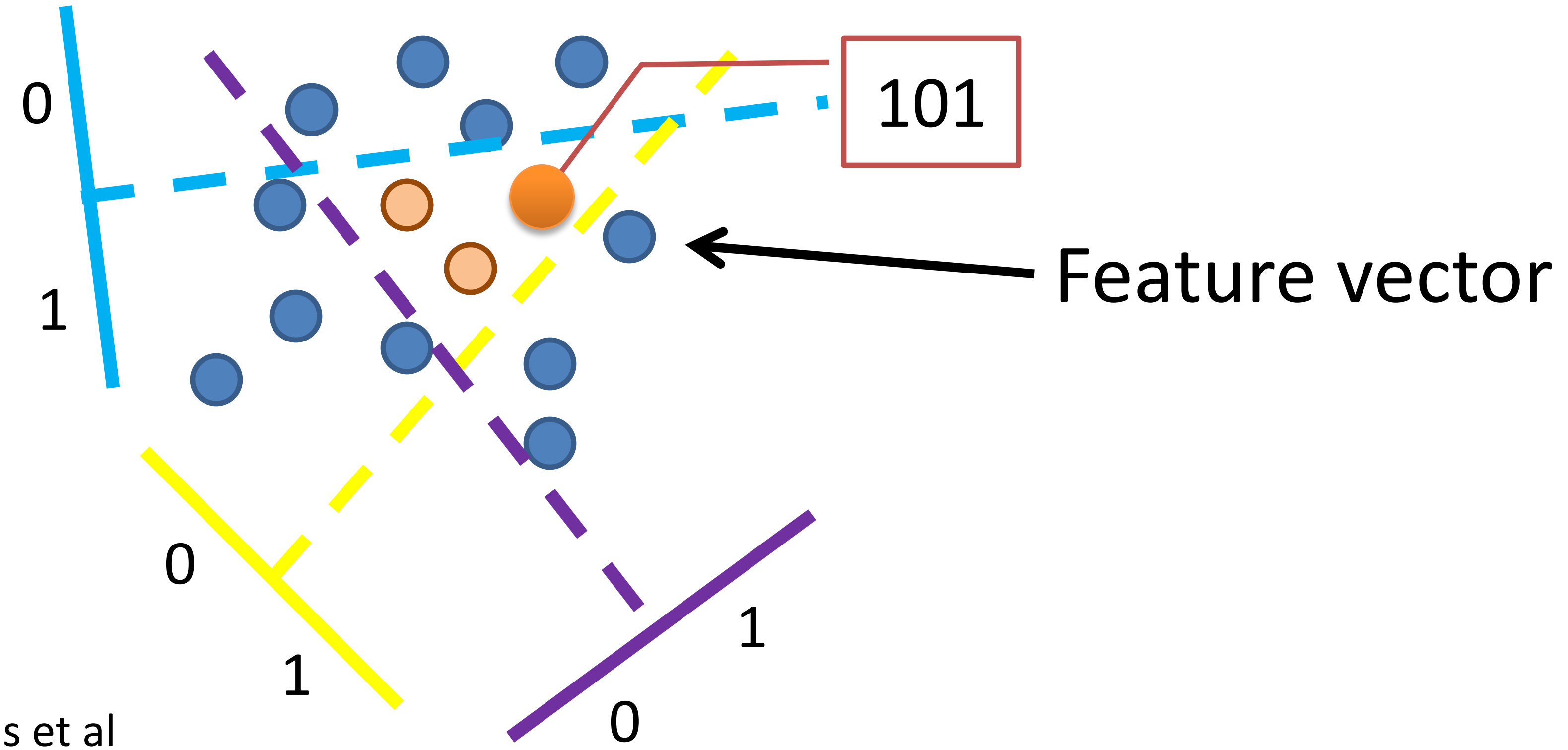

$\boldsymbol{r}^T\boldsymbol{x} < 0$
$\boldsymbol{r}^T\boldsymbol{x} \geq 0$

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:

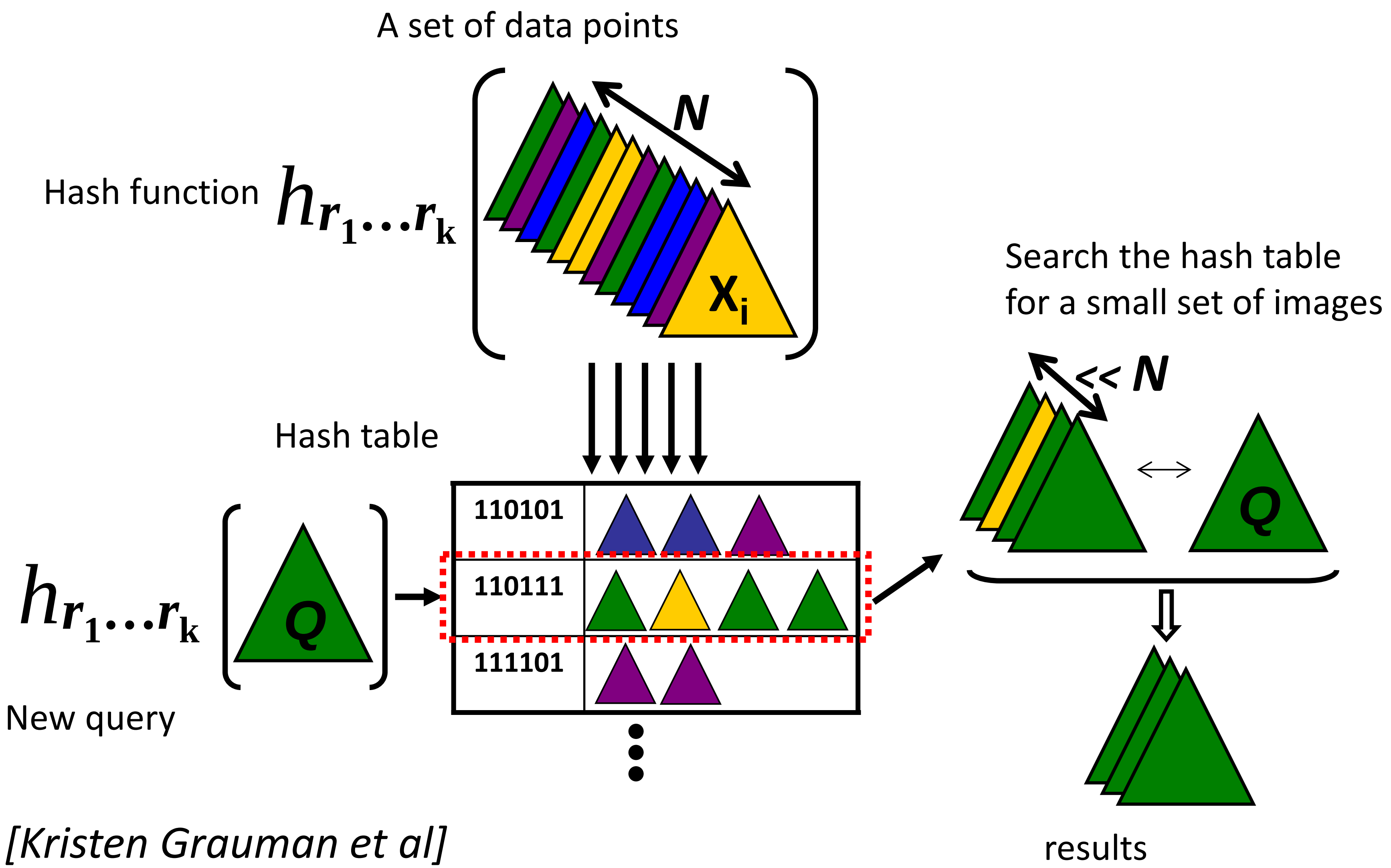
$$\Pr[h(\boldsymbol{x}_i) = h(\boldsymbol{x}_j)] = \mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

  - Where $\mathrm{sim}(\boldsymbol{x}_i, \boldsymbol{x}_j) \in [0, 1]$ is the similarity function. In our case: $Pr[h(u) = h(v)] = 1 - \frac{\theta(u,v)}{\pi}$

Kristen Grauman et al

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

Approximate Nearest-Neighbor Search



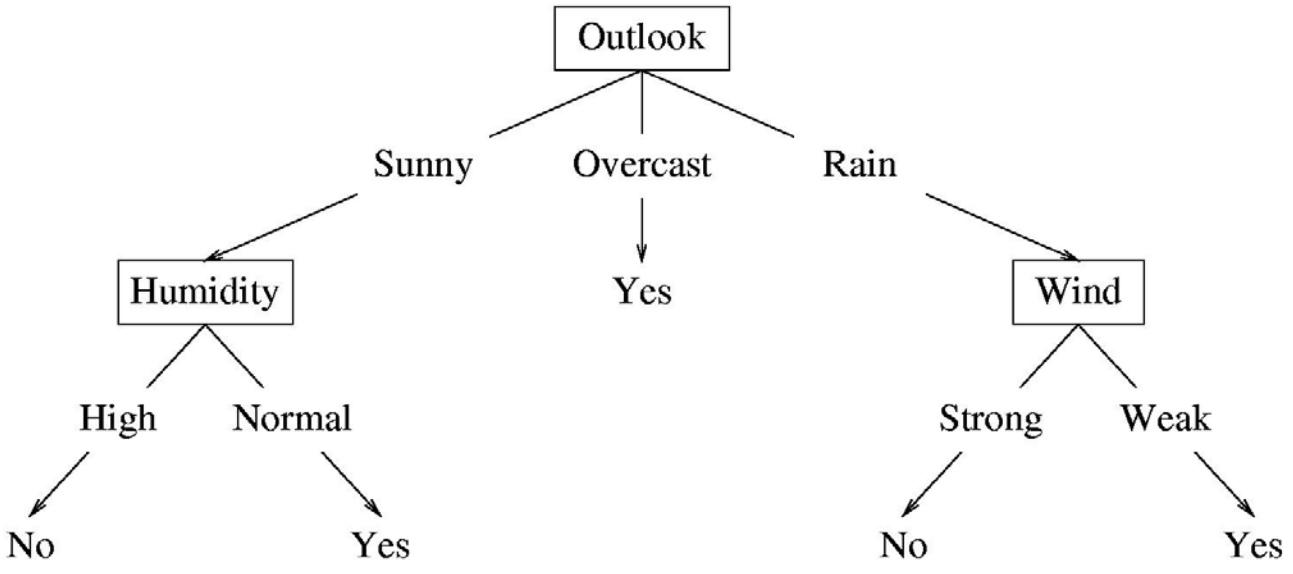
[Kristen Grauman et al]

Decision Trees

- In kd-tree, we looked at the data *x*, but not the labels *y*.
  - How can we use the labels?

Decision Tree Hypothesis Space

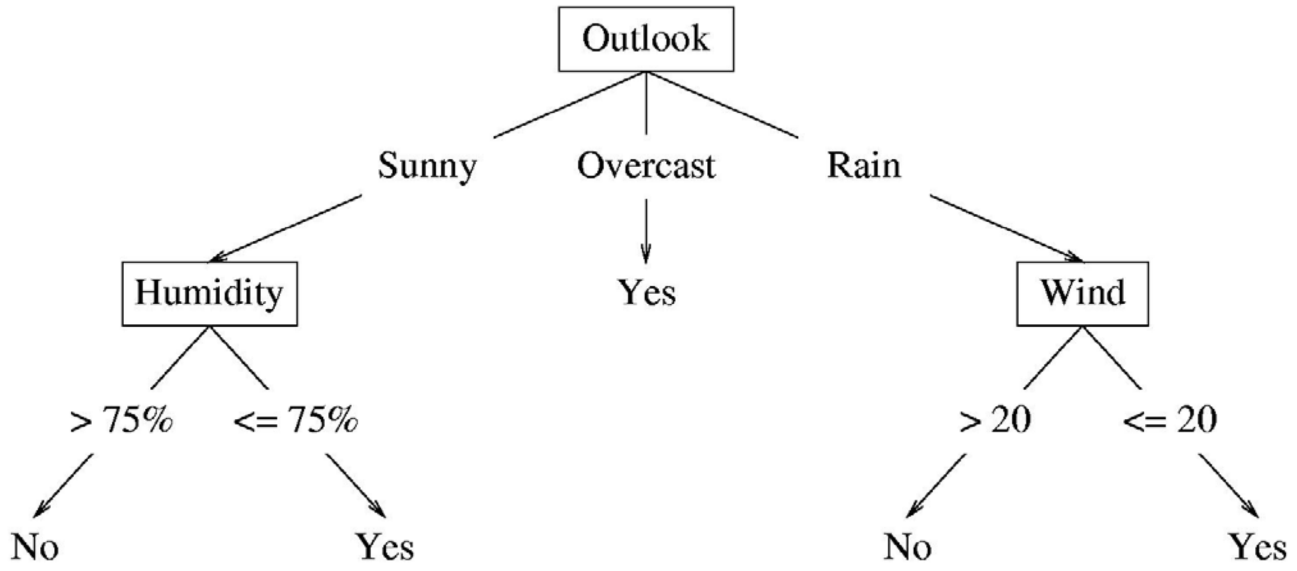
- **Internal nodes** test the value of particular features x_j and branch according to the results of the test.
- **Leaf nodes** specify the class $h(\mathbf{x})$.



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

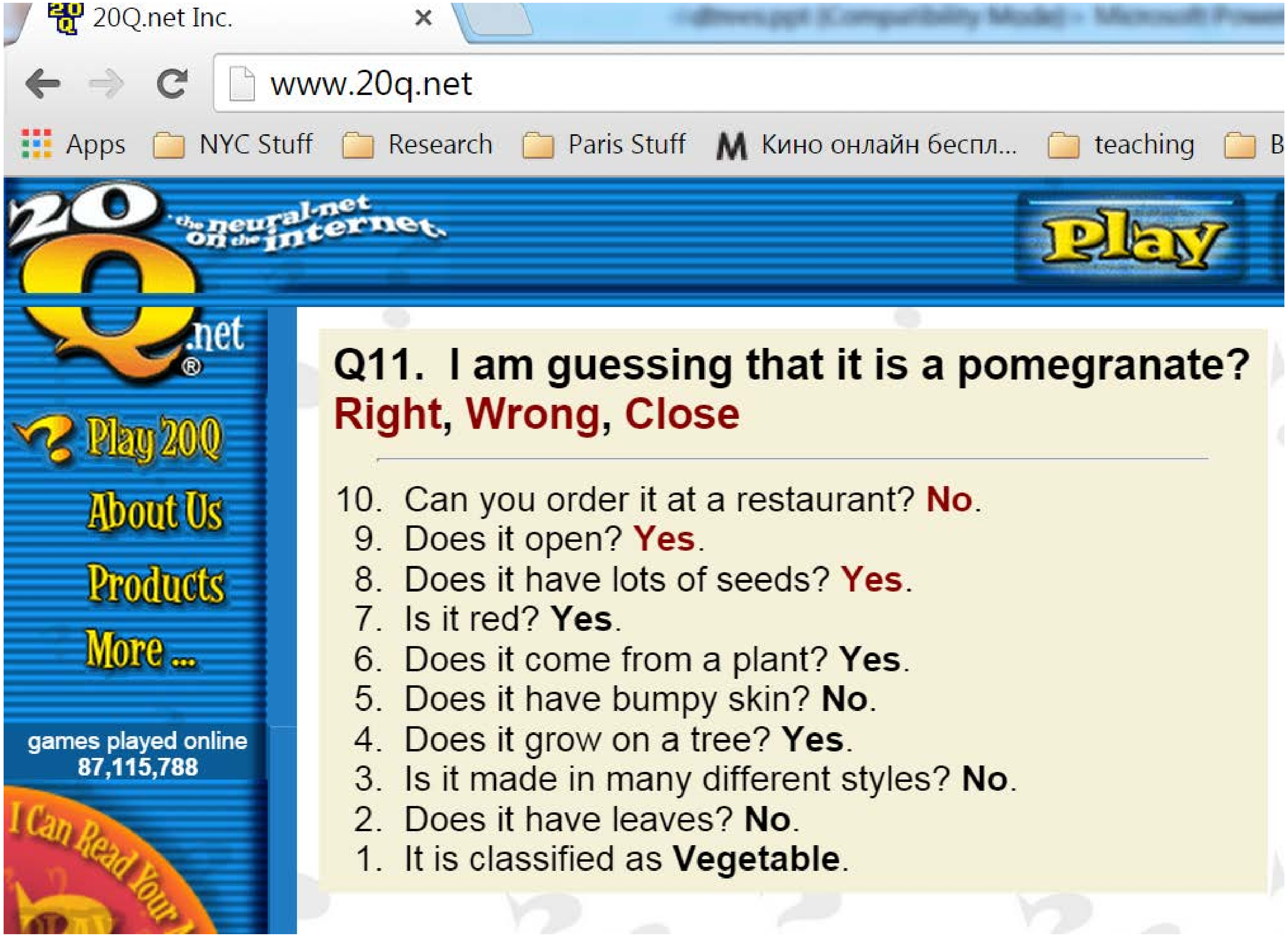
If the features are continuous, internal nodes may test the value of a feature against a threshold.



Benefits of Decision Trees

- Feature selection
- interpretable result
- What would the decision regions look like?
  - Somewhere between kNN and linear classifiers

20 Questions Game



games played online
87,115,788

Q11. I am guessing that it is a pomegranate?
Right, Wrong, Close

10. Can you order it at a restaurant? **No**.
9. Does it open? **Yes**.
8. Does it have lots of seeds? **Yes**.
7. Is it red? **Yes**.
6. Does it come from a plant? **Yes**.
5. Does it have bumpy skin? **No**.
4. Does it grow on a tree? **Yes**.
3. Is it made in many different styles? **No**.
2. Does it have leaves? **No**.
1. It is classified as **Vegetable**.

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.

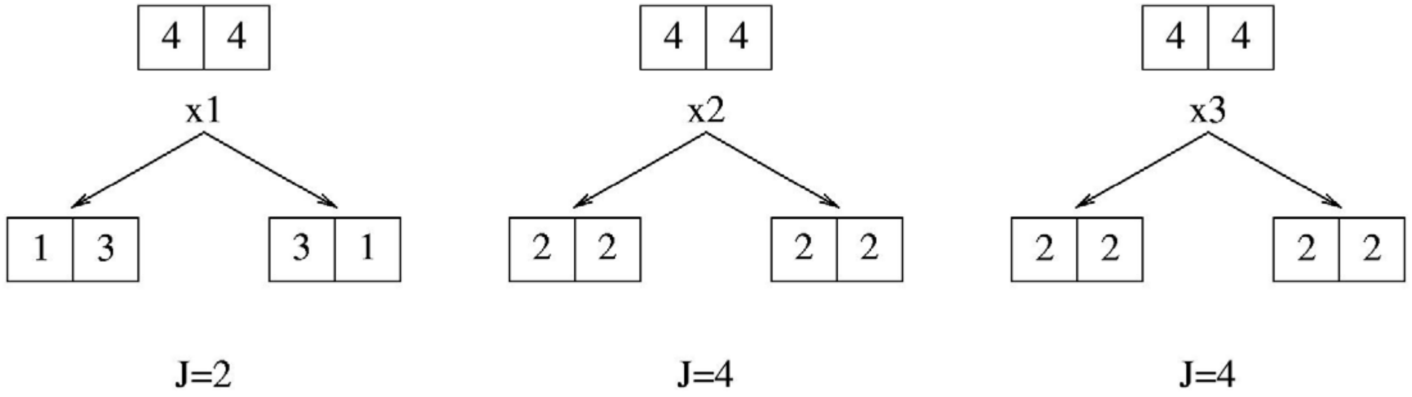
Choosing the Best Attribute

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


4 | 4
x1
1 | 3
3 | 1
J=2
4 | 4
x2
2 | 2
2 | 2
J=4
4 | 4
x3
2 | 2
2 | 2
J=4