# CS60020: Foundations of Algorithm Design and Machine Learning

Sourangshu Bhattacharya

# NON-PARAMETRIC MODELS

# Instance-Based Classifiers

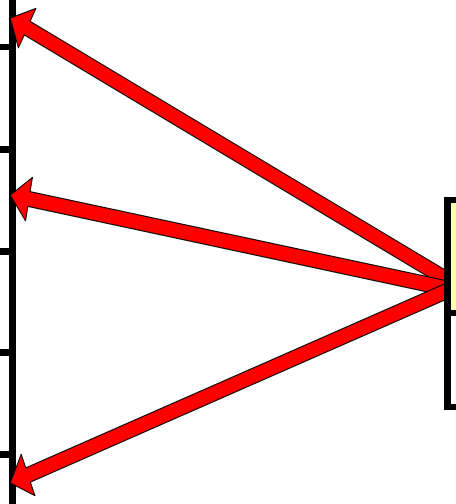
Set of Stored Cases

| Atr1 | …… | AtrN | Class |
|---|---|---|---|
| | | | A |
| | | | B |
| | | | B |
| | | | C |
| | | | A |
| | | | C |
| | | | B |

- Store the training records
- Use training records to predict the class label of unseen cases

Unseen Case

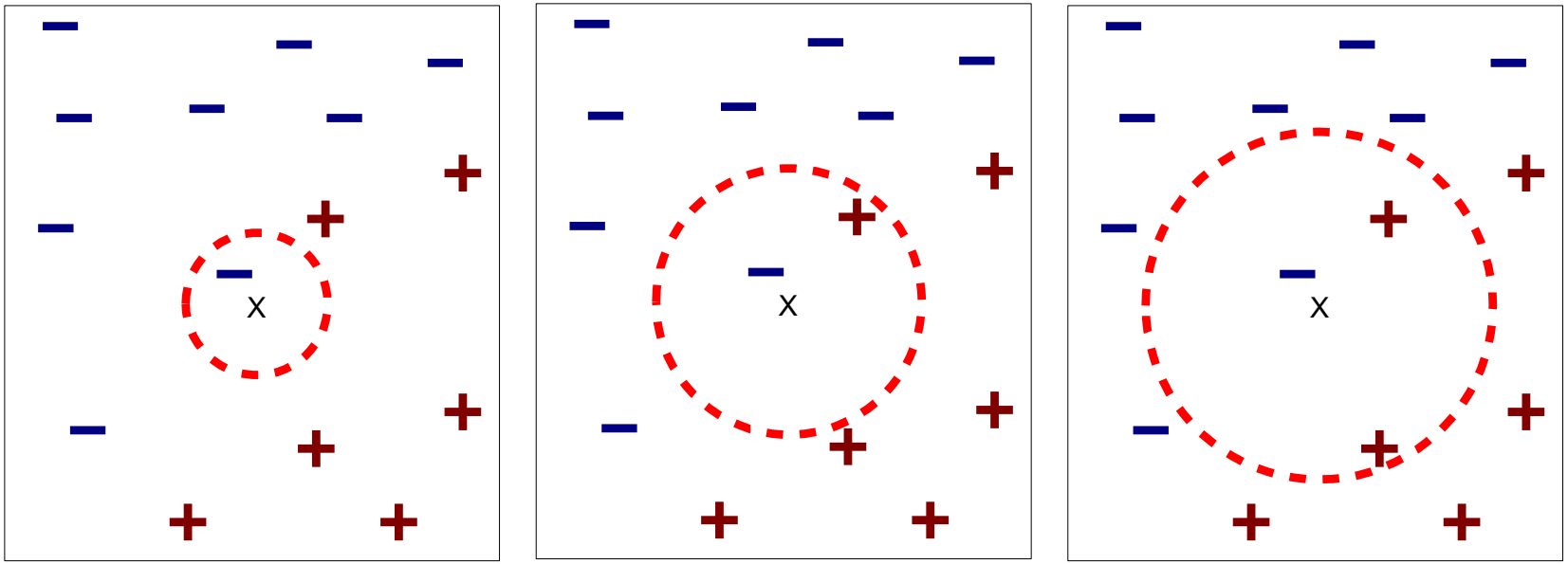
| Atr1 | …… | AtrN |
|---|---|---|
| | | |



# Instance Based Classifiers

- Examples:
  - Rote-learner
    - Memorizes entire training data and performs classification only if attributes of record match one of the training examples exactly
  - Nearest neighbor
    - Uses k “closest” points (nearest neighbors) for performing classification

# Definition of Nearest Neighbor




(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

# Nearest Neighbor Classification

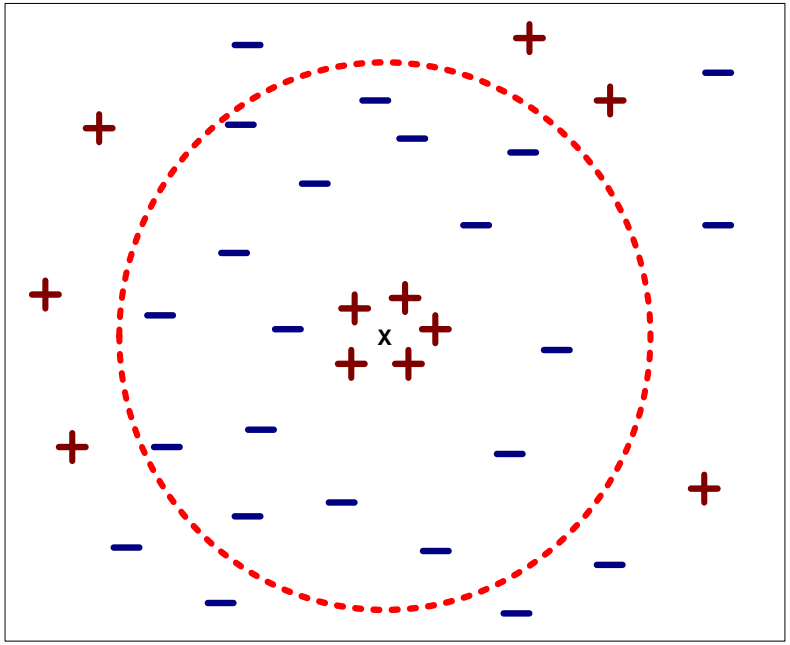
- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

- Determine the class from nearest neighbor list
  - take the majority vote of class labels among the k-nearest neighbors
  - Weigh the vote according to distance
    - weight factor, $w = 1/d^2$

# Nearest Neighbor Classification…

- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



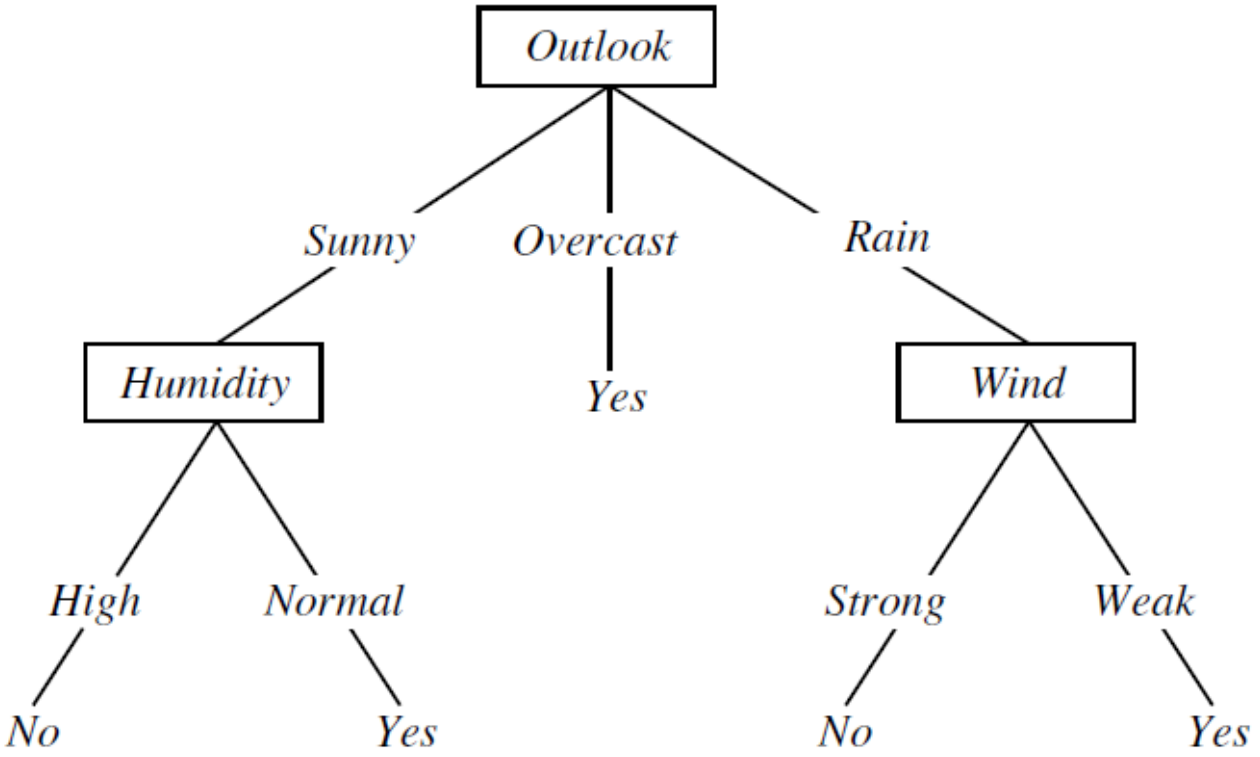
# Nearest Neighbor Classification…

- Scaling issues
  - Attributes may have to be scaled to prevent distance measures from being dominated by one of the attributes
  - Example of an attribute dominating distance computation:
    - height of a person may vary from 1.5m to 1.8m
    - weight of a person may vary from 90lb to 300lb
    - income of a person may vary from $10K to $1M

# Decision Tree

- Instance has values of attributes.
  - Same as feature values
- Target value is discrete.
- Each internal node tests an attribute.
- Each branch corresponds to a value of an attribute.
- Each leaf node assigns a classification.

# Decision tree - Example



# Decision tree learning

- For a test datapoint:
  - Determine the branches to take at each level based on attribute value.
  - At a leaf level, return the label of current node.
- For training, at each level:
  - Select an attribute split the training dataset on.
  - Examine the purity of data at each splitted node and determine whether it is a leaf node.
  - Determine the label at each node

# What attribute to select ?

- Entropy:
  - $S$ 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 $S$

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

# What attribute to select ?

$Entropy(S)$ = expected number of bits needed to encode class ($\oplus$ or $\ominus$) of randomly drawn member of $S$ (under the optimal, shortest-length code)

Why?

Information theory: optimal length code assigns $-\log_2 p$ bits to message having probability $p$.

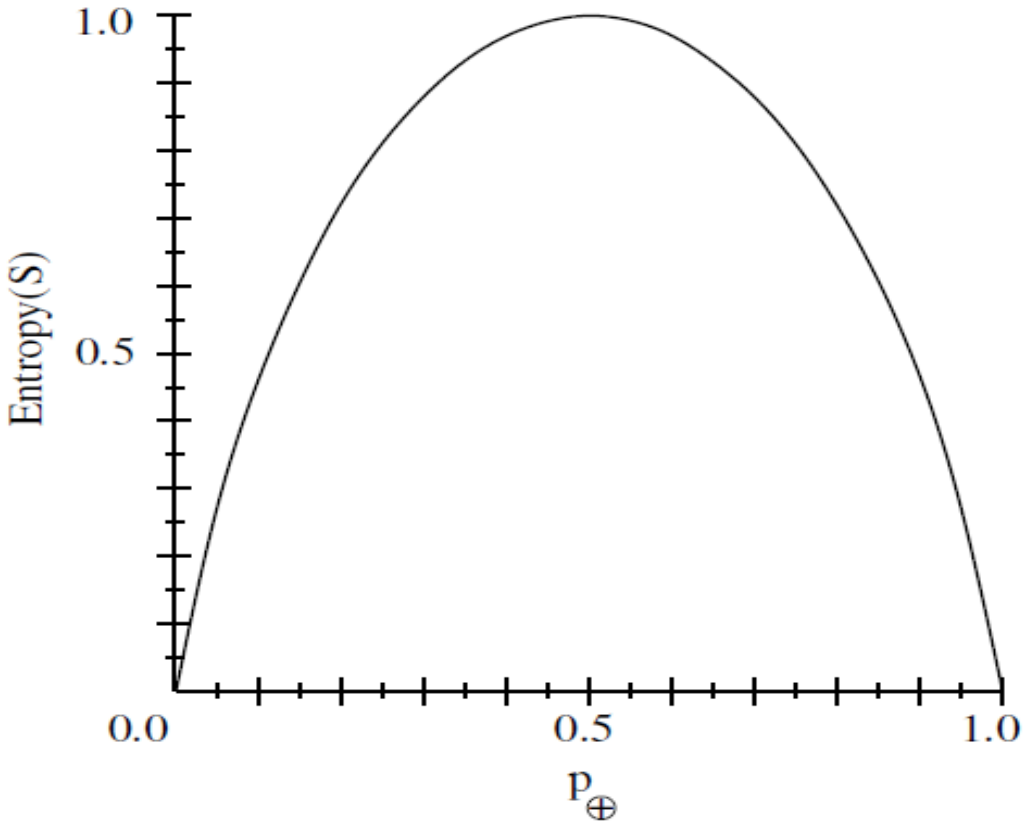
So, expected number of bits to encode $\oplus$ or $\ominus$ of random member of $S$:

$$p_{\oplus}(-\log_2 p_{\oplus}) + p_{\ominus}(-\log_2 p_{\ominus})$$

$$Entropy(S) \equiv -p_{\oplus}\log_2 p_{\oplus} - p_{\ominus}\log_2 p_{\ominus}$$

# What attribute to select ?

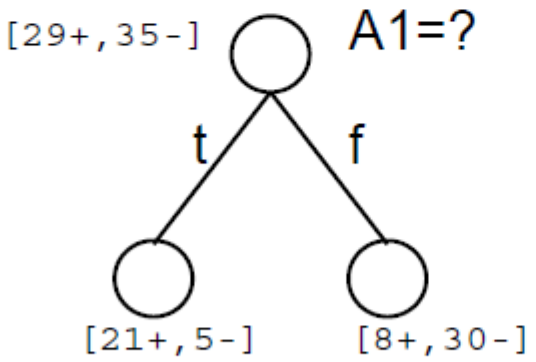
- Entropy:

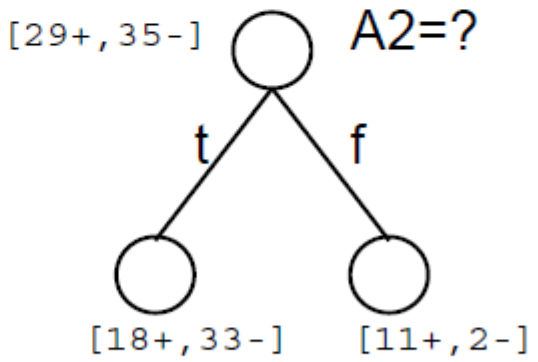


# Information Gain

$Gain(S, A)$ = expected reduction in entropy due to sorting on $A$

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$



# ID3 Algorithm

**function** ID3 (**R:** attributes, **S:** a training set, *default_value*) returns a **decision tree**;

**begin**

**if** S=∅, **then return** *default_value*;

**else if** S consists of records all with the value *v* **then return** value *v*;

**else f** R=∅, **then return** the most frequent value *v* for records of S;

**else**

**let** *A* be the attribute with largest Gain(*A*,*S*) among attributes in R;

**let** $\{a_j | j=1,2, .., m\}$ be the values of attribute *A*;

**let** $\{S_j | j=1,2, .., m\}$ be the subsets of *S* consisting respectively of records with value $a_j$ for A;

**return** a tree with root labeled *A* and arcs labeled $a_1, a_2, .., a_m$ going respectively to the trees (ID3(R-{A}, $S_1$), ID3(R-{A}, $S_2$), .....,ID3(R-{A}, $S_m$);

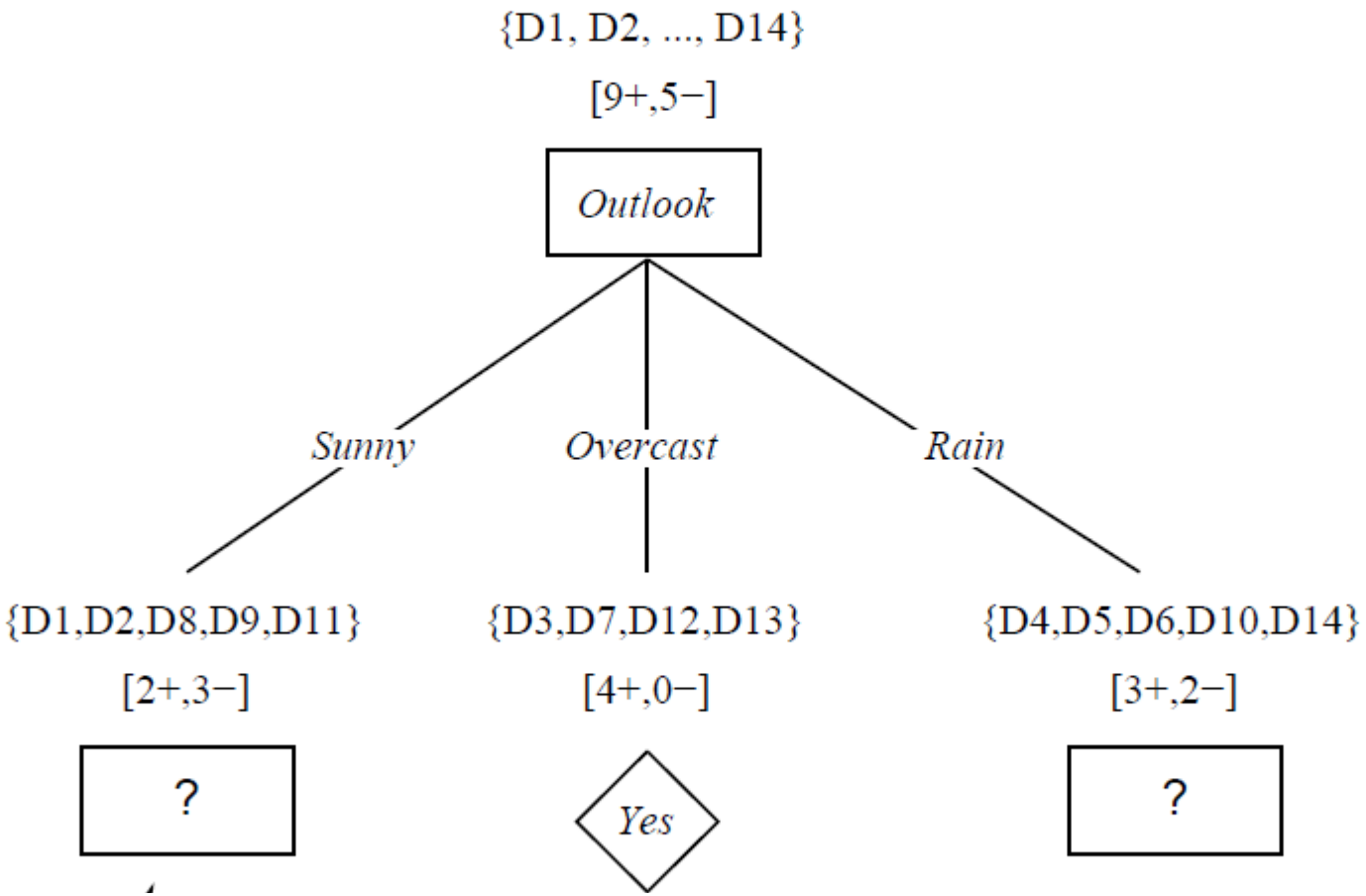
# ID3 example

| Day | Outlook | Temperature | Humidity | Wind | PlayTennis |
|---|---|---|---|---|---|
| D1 | Sunny | Hot | High | Weak | No |
| D2 | Sunny | Hot | High | Strong | No |
| D3 | Overcast | Hot | High | Weak | Yes |
| D4 | Rain | Mild | High | Weak | Yes |
| D5 | Rain | Cool | Normal | Weak | Yes |
| D6 | Rain | Cool | Normal | Strong | No |
| D7 | Overcast | Cool | Normal | Strong | Yes |
| D8 | Sunny | Mild | High | Weak | No |
| D9 | Sunny | Cool | Normal | Weak | Yes |
| D10 | Rain | Mild | Normal | Weak | Yes |
| D11 | Sunny | Mild | Normal | Strong | Yes |
| D12 | Overcast | Mild | High | Strong | Yes |
| D13 | Overcast | Hot | Normal | Weak | Yes |
| D14 | Rain | Mild | High | Strong | No |

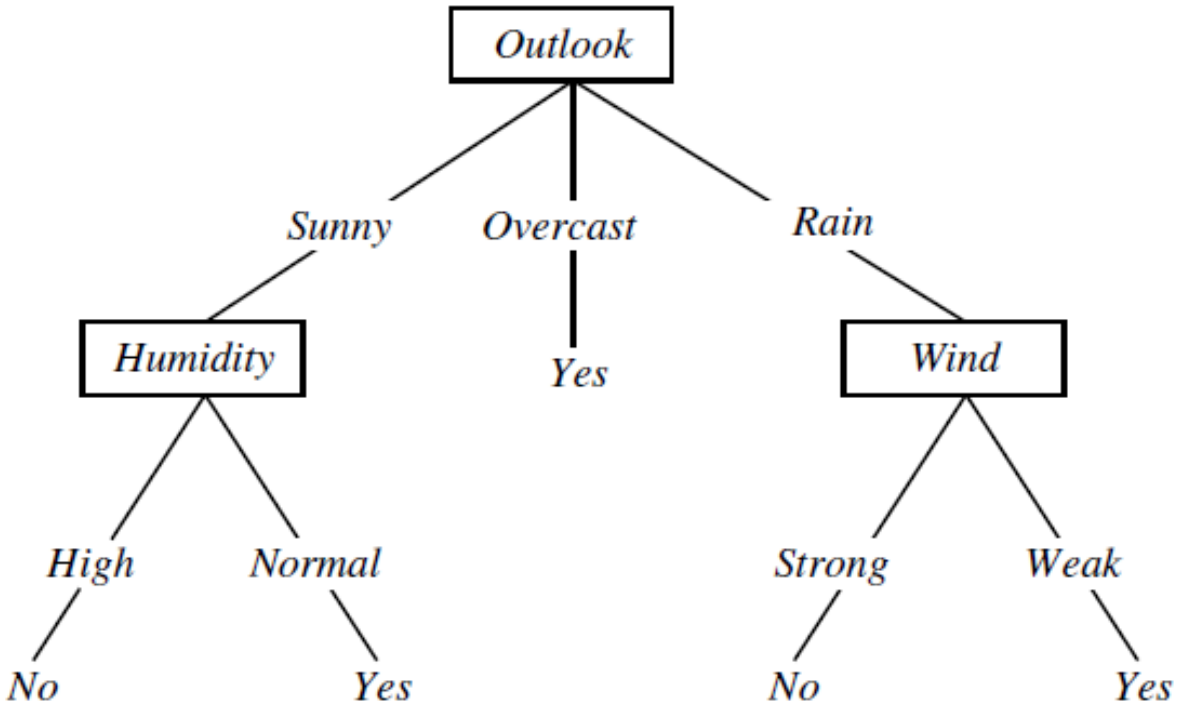
# Root level choice

- Gain(S,outlook) = 0.246
- Gain(S, Humidity) = 0.151
- Gain(S, Wind) = 0.048
- Gain(S, Temperature) = 0.029

# Root level tree



# Tree


Outlook
Sunny
Overcast
Rain
Humidity
Yes
Wind
High
Normal
Strong
Weak
No
Yes
No
Yes

# Inductive Bias

- Shorter trees are preferred over longer trees.
- Occam's razor: "Simpler" hypothesis are better than more complex hypothesis.
- Shorter trees are "simpler" because there are less number of them.

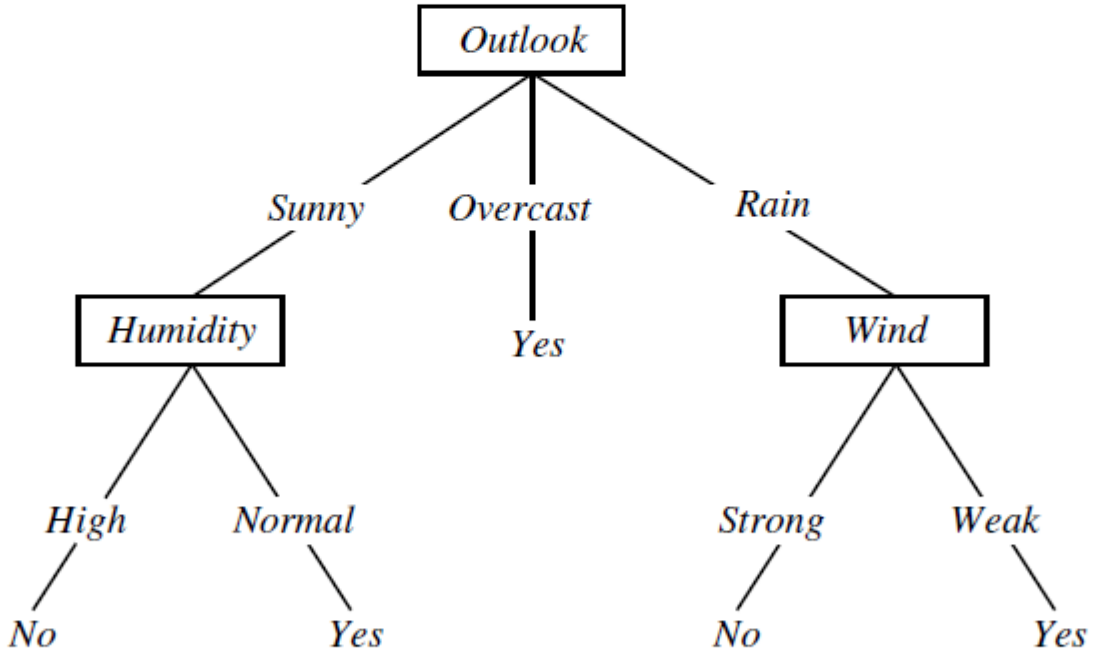
- Actually: shorter trees with attributes having more information gain are preferred.

# Overfitting

Consider adding noisy training example #15:

$Sunny,\ Hot,\ Normal,\ Strong,\ PlayTennis = No$

What effect on earlier tree?



# Overfitting

Consider error of hypothesis $h$ over

- training data: $error_{train}(h)$
- entire distribution $\mathcal{D}$ of data: $error_{\mathcal{D}}(h)$

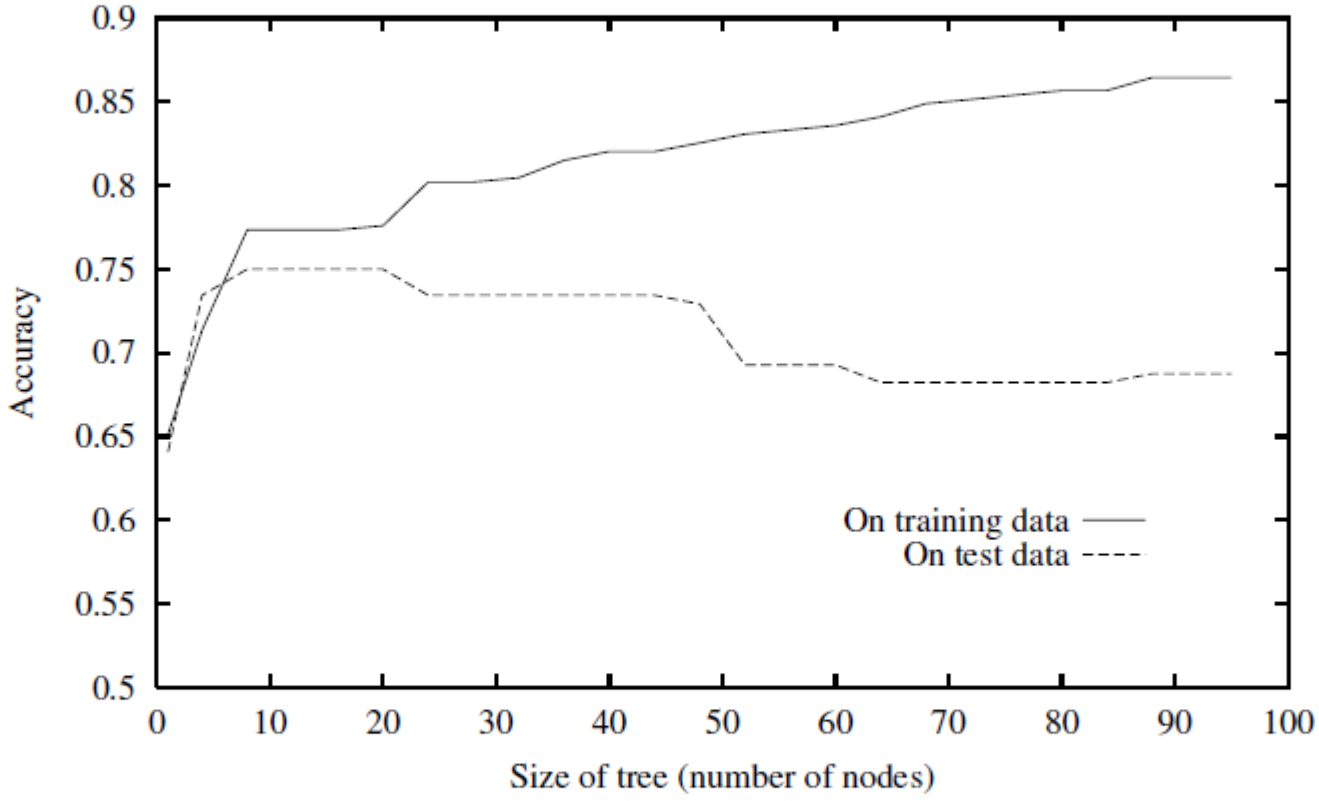
Hypothesis $h \in H$ **overfits** training data if there is an alternative hypothesis $h' \in H$ such that

$$error_{train}(h) < error_{train}(h')$$

and

$$error_{\mathcal{D}}(h) > error_{\mathcal{D}}(h')$$

# Overfitting



# Avoiding overfitting

How can we avoid overfitting?

- stop growing when data split not statistically significant
- grow full tree, then post-prune

- Which is computationally better ?

# Avoiding overfitting

Which tree is best ?

- Measure the accuracy over a separate validation set.
- Perform statistical tests over training data, e.g. chi square tests.
- Minimize an explicit performance measure which comprises of training set performance and “complexity” of the model, e.g. MDL.

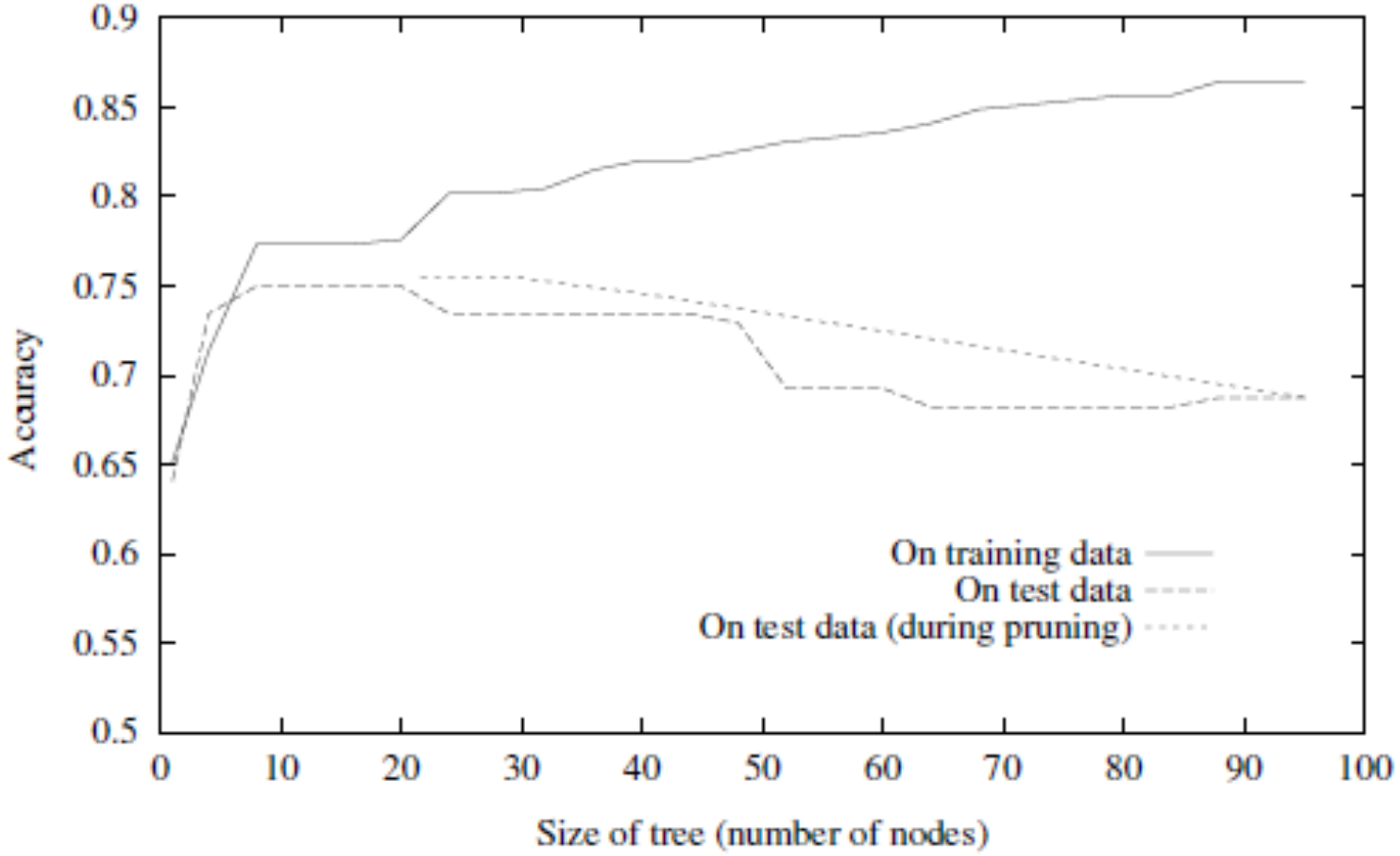
# Reduced error pruning

Split data into *training* and *validation* set

Do until further pruning is harmful:

1. Evaluate impact on *validation* set of pruning each possible node (plus those below it)
2. Greedily remove the one that most improves *validation* set accuracy

# Reduced error pruning



# Rule post pruning

1. Convert tree to equivalent set of rules
2. Prune each rule independently of others
3. Sort final rules into desired sequence for use

Perhaps most frequently used method (e.g., C4.5)

# Other issues

- Handling continuous valued attribute.
  - Create a split which produces the highest information gain.
- Handling large number of discrete valued attributes.

$$GainRatio(S, A) \equiv \frac{Gain(S, A)}{SplitInformation(S, A)}$$

$$SplitInformation(S, A) \equiv -\sum_{i=1}^{c} \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

# Other issues

- Handling missing attributes.
- Training set:
  - Assign a distribution based on exiting values.
  - Default branch for the set of attributed.
- Test set:
  - Default branch which is populated using missing values.

# ENSEMBLE METHODS

# Rationale for Ensemble Learning

- No Free Lunch thm: There is no algorithm that is always the most accurate
- Generate a group of base-learners which when combined have higher accuracy
- Different learners use different
  - Algorithms
  - Parameters
  - Representations (Modalities)
  - Training sets
  - Subproblems

# Voting

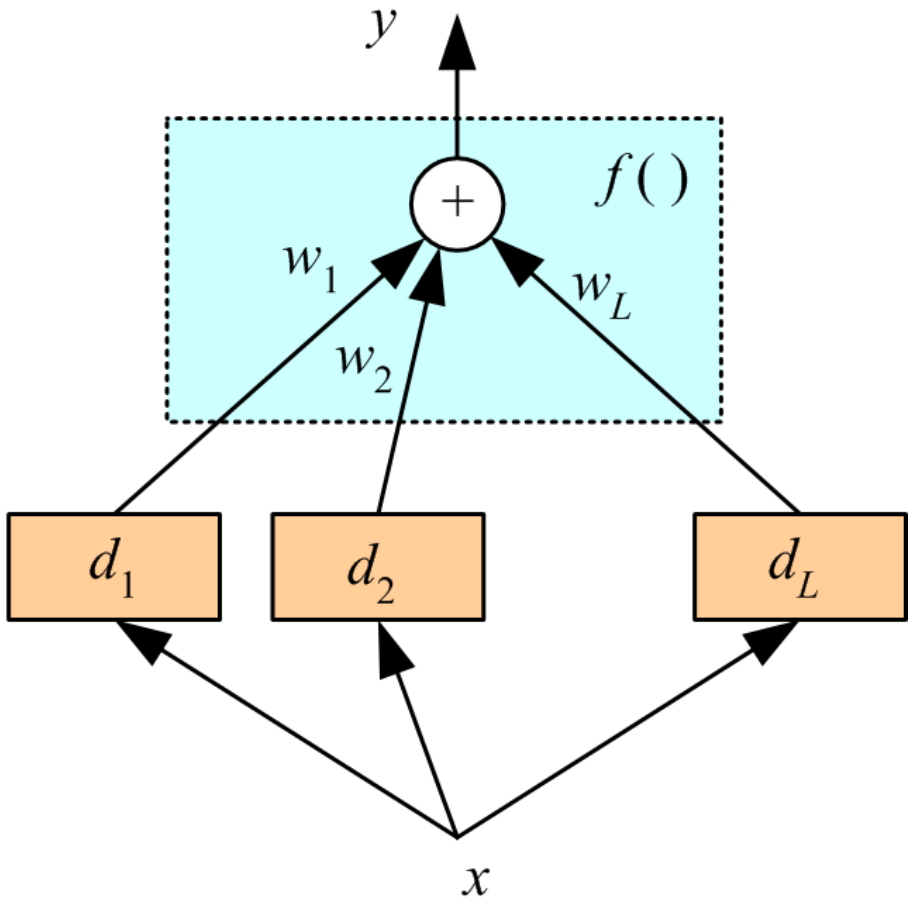
- Linear combination

$$y = \sum_{j=1}^{L} w_j d_j$$

$$w_j \geq 0 \text{ and } \sum_{j=1}^{L} w_j = 1$$

- Classification

$$y_i = \sum_{j=1}^{L} w_j d_{ji}$$



# BAGGING

# Ensemble methods

- A single decision tree does not perform well
- But, it is super fast
- What if we learn multiple trees?

We need to make sure they do not all just learn the same

# Bagging

If we split the data in random different ways, decision trees give different results, **high variance.**

**Bagging: B**ootstrap **agg**regat**ing** is a method that result in low variance.

If we had multiple realizations of the data (or multiple samples) we could calculate the predictions multiple times and take the average of the fact that averaging  multiple onerous estimations produce less uncertain results

# Bagging

Say for each sample *b*, we calculate $f^b(x)$, then:

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

How?

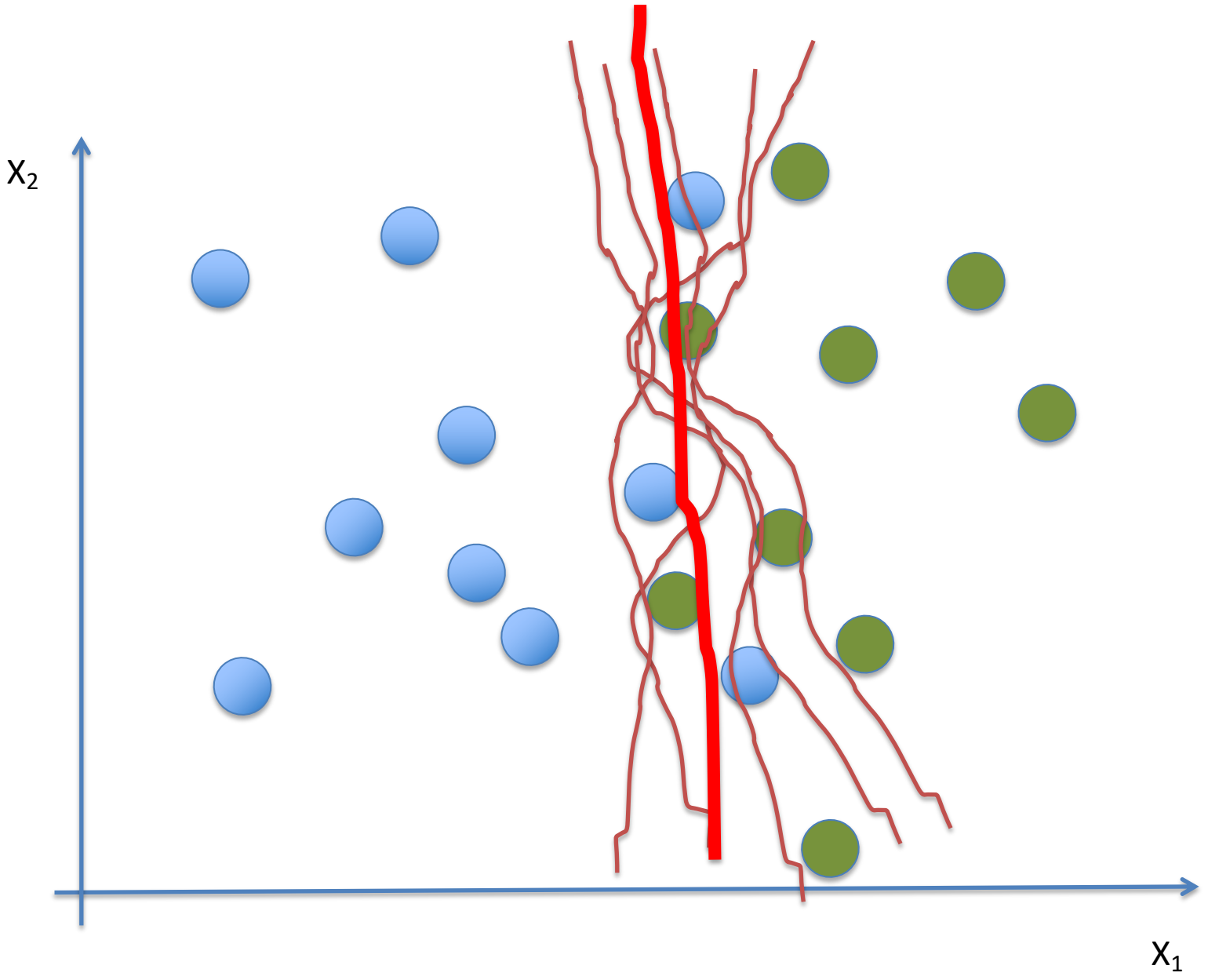
---

**Bootstrap**

Construct B (hundreds of) trees (no pruning)

Learn a classifier for each bootstrap sample and average them

Very effective


$X_2$
$X_1$

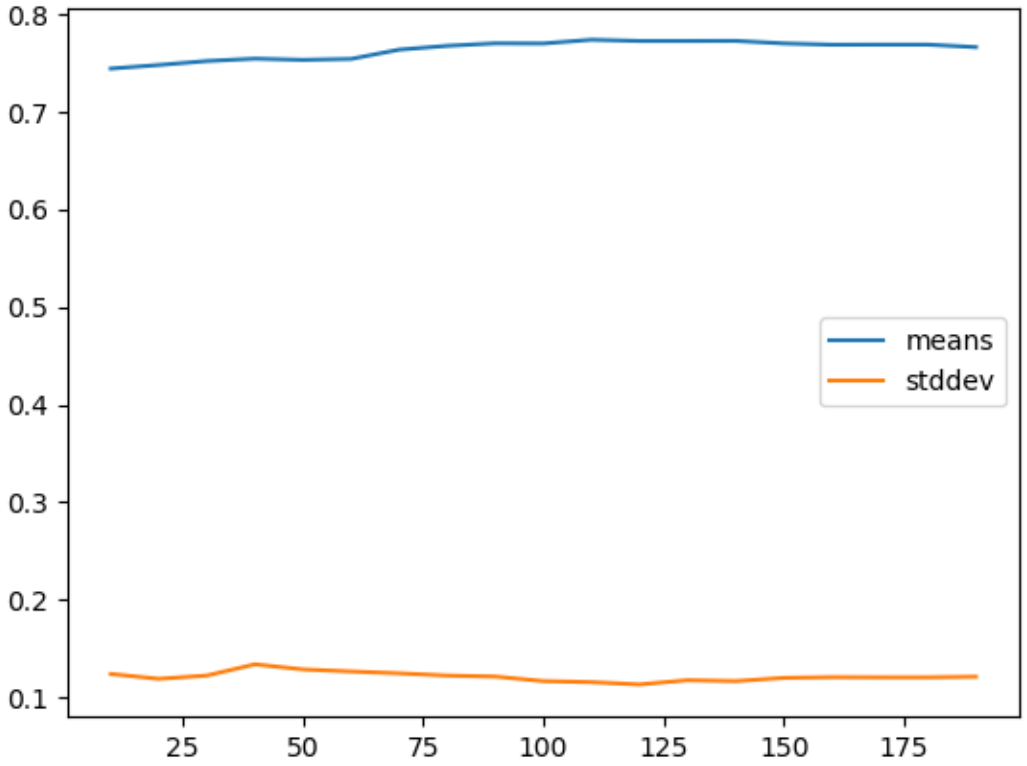
# Example

- Pima Indians Diabetes data:
- 768 examples, 8 features
- Features: 'preg', 'plas', 'pres', 'skin', 'test', 'mass', 'pedi', 'age'

# Example

```
for num_trees in range(start,stop,step):
    kfold = model_selection.KFold(n_splits=50,\
                                  random_state=int(time.time()))
    model = BaggingClassifier(base_estimator=cart,\
                              n_estimators=num_trees,\
                              random_state=seed)
    model = RandomForestClassifier(num_trees,\
                                   criterion="gini",\
                                   max_features=5,\
                                   random_state=int(time.time()))
    results = model_selection.cross_val_score(model, X, Y, cv=kfold)
    print(results.mean(), statistics.stdev(results))
    means.append(results.mean())
    stds.append(statistics.stdev(results))
```

# Example




# trees

# Out-of-Bag Error Estimation

- No cross validation?
- Remember, in bootstrapping we sample with replacement, and therefore **not all observations are used for each bootstrap sample**.
- We call them out-of-bag samples (OOB)
- We can predict the response for the *i-th* observation using each of the trees in which that observation was OOB and do this for *n* observations
- Calculate overall OOB MSE or classification error

# Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

# Bagging - issues

Each tree is identically distributed (i.d.)

➔ the expectation of the average of *B* such trees is the same as the expectation of any one of them

➔the bias of bagged trees is the same as that of the individual trees

i.d. and not i.i.d

# Bagging - issues

An average of *B* i.i.d. random variables, each with variance $\sigma^2$, has variance*: $\sigma^2/B$*

If i.d. (identical but not independent) and pair correlation $\rho$ is present, then the variance is:

$$\rho\,\sigma^2 + \frac{1-\rho}{B}\sigma^2$$

As *B* increases the second term disappears but the first term remains

Why does bagging generate correlated trees?

# Bagging - issues

Suppose that there is one very strong predictor in the data set, along with a number of other moderately strong predictors.

Then all bagged trees will select the strong predictor at the top of the tree and therefore all trees will look similar.

How do we avoid this?

# RANDOM FORESTS

# Random Forests

As in bagging, we build a number of decision trees on bootstrapped training samples each time a split in a tree is considered, a random sample of *m* predictors is chosen as split candidates from the full set of p predictors.

Note that if $m = p$, then this is bagging.

# Random Forests

Random forests are popular. Leo Breiman's and Adele Cutler maintains a random forest website where the software is freely available, and of course it is included in every ML/STAT package

http://www.stat.berkeley.edu/~breiman/RandomForests/

# Random Forests Algorithm

For b = 1 to B:

(a) Draw a bootstrap sample $Z^*$ of size *N* from the training data.

(b) Grow a random-forest tree to the bootstrapped data, by recursively repeating the following steps for each terminal node of the tree, until the minimum node size $\boldsymbol{n_{min}}$ 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.

Output the ensemble of trees.

To make a prediction at a new point *x* we do:

For regression: average the results

For classification: majority vote

# Random Forests Tuning

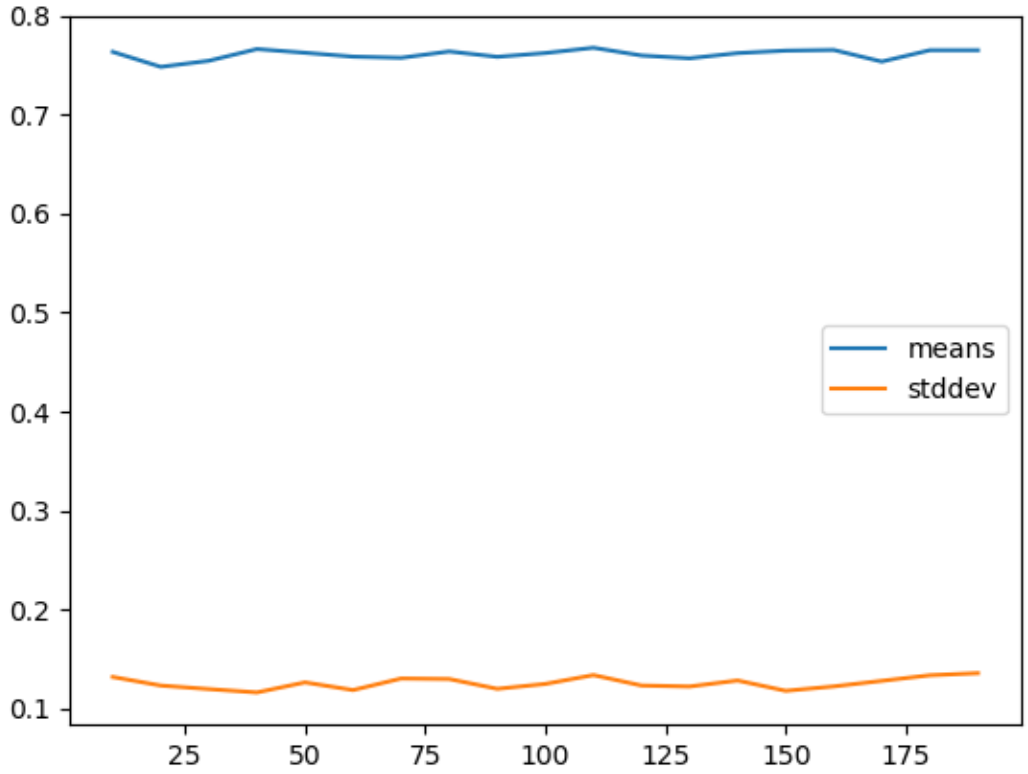
The inventors make the following recommendations:

- For classification, the default value for $m$ is $\sqrt{p}$ and the minimum node size is one.
- For regression, the default value for m is $p/3$ and the minimum node size is five.

In practice the best values for these parameters will depend on the problem, and they should be treated as tuning parameters.

Like with Bagging, we can use OOB and therefore RF can be fit in one sequence, with cross-validation being performed along the way. Once the OOB error stabilizes, the training can be terminated.

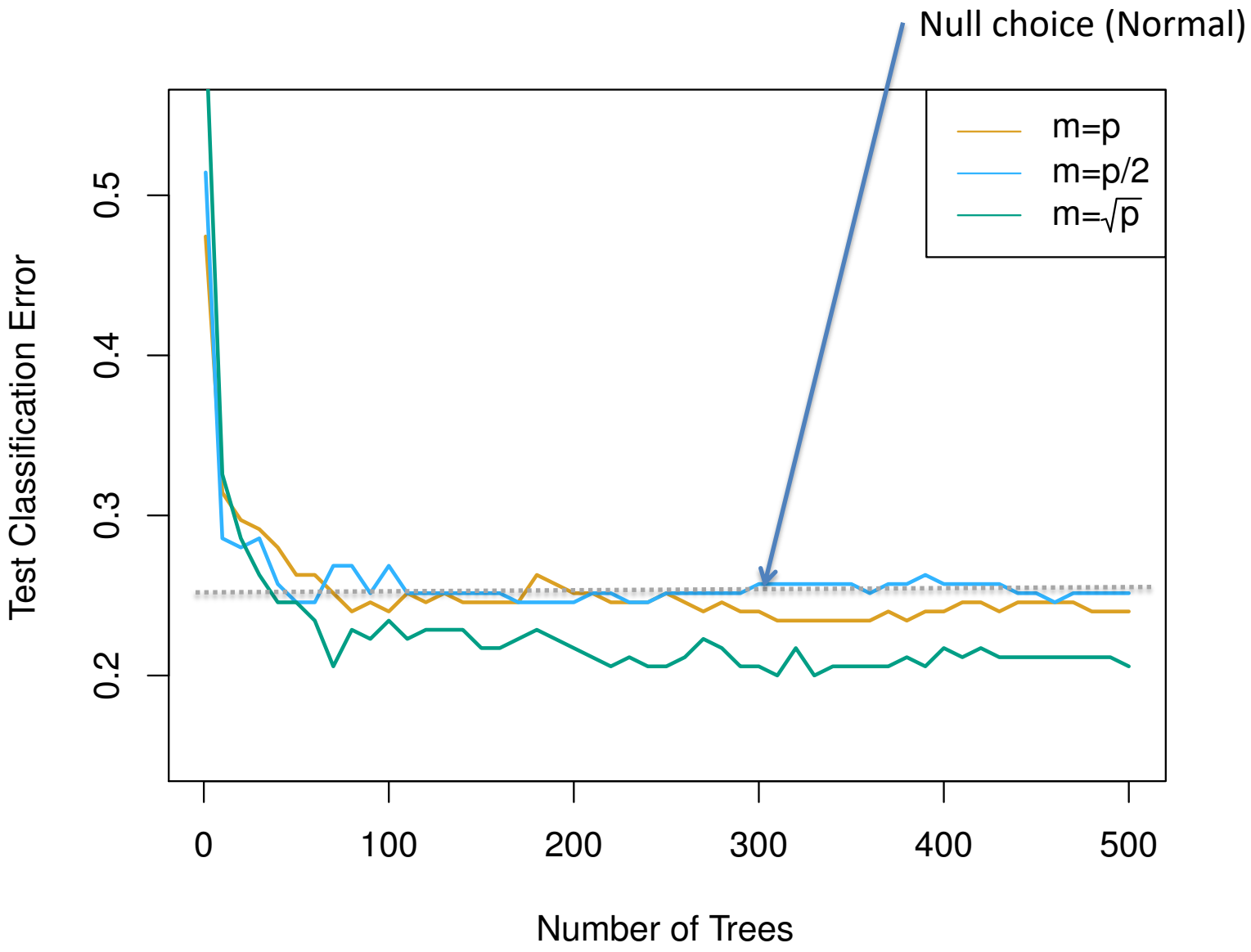
# Example



# Example

- 4,718 genes measured on tissue samples from 349 patients.
- Each gene has different expression
- Each of the patient samples has a qualitative label with 15 different levels: either normal or 1 of 14 different types of cancer.

Use random forests to predict cancer type based on the 500 genes that have the largest variance in the training set.


Null choice (Normal)
m=p
m=p/2
m=√p
Test Classification Error
0.2
0.3
0.4
0.5
0
100
200
300
400
500
Number of Trees

# Random Forests Issues

When the number of variables is large, but the fraction of relevant variables is small, random forests are likely to perform poorly when *m* is small

Why?

---

Because:

At each split the chance can be small that the relevant variables will be selected

For example, with 3 relevant and 100 not so relevant variables the probability of any of the relevant variables being selected at any split is ~0.25

# Can RF overfit?

Random forests "cannot overfit" the data wrt to number of trees.

Why?

---

The number of trees, *B* does not mean increase in the flexibility of the model