Discriminatory Models (contd.)

Machine Learning

Conditional vs Joint Probability Models I

- Using data to estimate parameters:
  - Means and s.d's of known theoretical distributions;
  - Weights in regression
  - Weights in threshold machines (perceptrons, MLPs)
- Coonceptually, we are identifying models for estimating $P(Y|X)$ (correctly, $P(Y|\mathrm{X})$

Conditional vs Joint Probability Models II

- Recap questions:
  1. What needs to be estimated to obtain $P(Y|X)$?
  2. What is captured by regularisation?
  3. How can we estimate the structure of the model for $P(Y|X)$?
  4. What does it mean for a model to be "correct"?

Conditional vs Joint Probability Models III

- Models for $P(Y|X)$ are called *discriminatory* models

Recap: $P(Y|X)$ as the basis of Classification I

- Given a set of training instances $(\mathrm{x}_1, y_1), (\mathrm{x}_2, y_2), \ldots, (\mathrm{x}_n, y_n)$ where $\mathrm{x}_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$
  - Often $\mathcal{X} = \Re^d$
  - $\mathcal{Y} = \{0, 1\}$ (or $\{0, 1, 2, \ldots, k\}$
  - For any x, we have $y = y(\mathrm{x})$. In general, x is taken as the value of a r.v. X and $y(\mathrm{x})$ is the value of the r.v. $Y = y(\mathrm{X})$.
- Bayes classifier:

$$\begin{aligned} h_B(X) &= 0 \quad \text{if } P(Y = 0|X) > P(Y = 1|X) \\ &= 1 \quad \text{otherwise} \end{aligned}$$

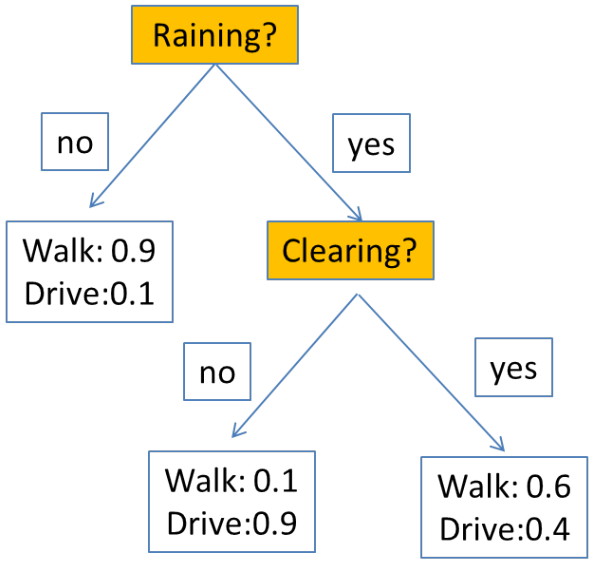
That is, the class with the maximum posterior probability is selected

Recap: $P(Y|X)$ as the basis of Classification II

- Recap questions:
  1. What is known about h_B?
  2. What is the difficulty with using h_B?
- The procedure for logistic regression estimates $P(Y|X)$ using a specific function (sigmoid)
- We will now look at some other ways of estimating $P(Y|X)$

Tree-Structured Discriminative Models I

- Here, we will examine the construction of *class probability trees* that combine logical tests with probabilistic estimation of $P(Y|X)$



- As always, there are 2 components to a model like this: a discrete structure consisting of the tree T (structure), and the probability tables at the leaves Θ (parameters). For simplicity, we will denote the pair (T, Θ) by $\mathcal{T}$

Tree-Structured Discriminative Models II

- So, the task of learning class probability trees is this: given a set of classes $\omega_1, \omega_2, \ldots, \omega_k$, and a sample of instances $D = (\mathrm{x}_1, y_1), (\mathrm{x}_2, y_2), \ldots, (\mathrm{x}_n, y_n) = D$, find $\mathcal{T}_1, \mathcal{T}_2, \ldots$ that can be used to calculate of the posterior probability $P(y' = \omega_i | \mathrm{x}', D)$ for any (new) instance x'
- Given any *single* $\mathcal{T} = (T, \Theta_T)$ the calculation of $P(y' = \omega_i | \mathrm{x}', D)$ is straightforward enough:
  - Update Θ_T using T and the data in D
  - The instance x' will land in some leaf l of the tree T
- Given more than one tree also this basic approach still works: we would simply have to take the weighted-vote of the $P(y' = \omega_i | \ldots)$ from each tree. The weights would be the posterior probability of the corresponding trees, given D:

$$P(y' = \omega | \mathrm{x}', D) = \sum_{\mathcal{T}} P(\mathcal{T} | D) P(y' = \omega | \mathrm{x}, D, \mathcal{T})$$

Tree-Structured Discriminative Models III

In general, it will usually be impractical to obtain all trees, and we will be only interested in getting a single tree, or a small set of trees. More on this later: before that, we need to look at the individual terms in the summation

- The posterior probability distribution over trees, given data d is given by Bayes over corresponding r.v.'s for the data and trees

$$P(T|D = d) \propto P(D = d|T)P(T)$$

As always, we need the likelihood and the prior. Let us start with a specific value of $T = t$

- Any specific value $T = t$ is actually composed of 2 parts, the tree structure π and its parameters θ. Let us call the corresponding r.v's Π and Θ

Tree-Structured Discriminative Models IV

- So, if we know the structure (say $\Pi = \pi$), the likelihood $P(d|\pi, \Theta)$ has to obtained from the p.d.f of Θ. Since π is fixed, the problem is in fact just $P(d|\Theta)$
- Now, suppose the specific structure π has K leaves. Let us assume an independent distribution $f_i(\theta_i)$ for each leaf (that is, there are K independent r.v's $\Theta_1, \Theta_2, \ldots \Theta_K$, each with its own p.d.f). That is $P(d|\Theta) = P(d|\Theta_1, \ldots, \Theta_k)$, where the Θ_i are independent of each other
- Also, let the data d result in d_1 instances in leaf l_1, d_2 instances in l_2 and so on. For Binomial data samples, with Beta priors on the Θ_i, we have calculated this likelihood before (see "Continuous Hypotheses, Discrete Data")

Tree-Structured Discriminative Models V

- Suppose data consisted of a Binomial sample containing s successes and t failures in $n = s + t$ trials with a Bernoulli r.v. Θ. Then a $Beta(a, b)$ prior p.d.f. for Θ $(= f(\Theta))$ results in a $Beta(a + s, b + t)$ posterior p.d.f. for Θ $(= f(\Theta|D = d))$
- Also:

$$\mathrm{E}_{\Theta|d}[\Theta] = \frac{a + s}{a + s + b + t}$$

Likelihood $P(d|\Theta)$ with a Beta Prior I

- The likelihood of observing d consisting of s heads and t tails with a Beta prior:

$$\begin{aligned} P(D = d|Beta(a,b)) &= \frac{(m-1)!}{(m+n-1)!}\frac{(a+s-1)!(b+t-1)!}{(a-1)!(b-1)!} \\ &= \frac{\Gamma(m)}{\Gamma(m+n)}\frac{\Gamma(a+s)\Gamma(b+t)}{\Gamma(a)\Gamma(b)} \\ &= \frac{\Gamma(a+b)}{\Gamma(a)\Gamma(b)}\frac{\Gamma(a+s)\Gamma(b+t)}{\Gamma(a+s+b+t} \\ &= \frac{B(a+s,b+t)}{B(a,b)} \end{aligned}$$

where:

$$B(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$$

Likelihood $P(d|\Theta)$ with a Beta Prior II

- Suppose instead the s and t were partitioned across K biased coins $(\Theta_1, \Theta_2, \ldots, \Theta_k)$ each with its own a_i, b_i and s_i, t_i, s.t. $m_i = a_i + b_i$, $n_i = s_i + t_i$, $s = sums_i$ and $t = \sum t_i$, then:

$$P(D|Beta(a_1, b_1); Beta(a_2, b_2); ldots) = \prod_{j=1}^{K} \frac{B(a_j + s_j, b_j + t_j)}{B(a_j, b_j)}$$

Return To: Posterior for Trees I

- Recall that we want to calculate the posterior probability of a class for a data instance x' using:

$$P(y' = \omega|\mathrm{x}', D) = \sum_{\mathcal{T}} P(\mathcal{T}|D)P(y' = \omega|\mathrm{x}', D, \mathcal{T})$$

- We know that the second term in the summation can be obtained easily (it is just the probability of ω in the leaf in which x' ends up)
- So, the hard part is obtainng the posterior probability $P(\mathcal{T}|D)$
- The posterior probability of a tree $\mathcal{T}$ given data D is:

$$\begin{aligned} P(\mathcal{T}|D) &= = P(D|\mathcal{T})P(\mathcal{T}) \\ &\propto P(D|T, \Theta_T)P(T)P(\Theta_T|T) \end{aligned}$$

Return To: Posterior for Trees II

- If the tree T has K leaves, then $\mathcal{T}$ effectively specifies K biased coins. Each leaf l_i (coin) has its own p.d.f $f_i(\theta_i)$ and has s_i data instances of ω_1 (*heads*) and t_i instances of ω_2 (*tails*).
- If we take the coins (leaves) as independent:

$$P(D = d|\Theta_1, \ldots, \Theta_k) = \prod_{i=1}^{K} \frac{B(s + a_i, t + b_i)}{B(a_i, b_i)}$$

- What about the prior? This is $P(\Pi, \Theta) = P(\Pi)P(\Theta|\Pi)$ For any particular value of $\Pi = \pi$, with K leaves and the assumption earlier, $P(\Theta|\pi) = P(\Theta_1|\pi)P(\Theta_2|\pi)\cdots P(\Theta_K|\pi)$. So what is $P(\Theta_i|\pi)$? This is simply $E(\Theta_i)$.
- If $\Theta_i \sim Beta(\alpha_i, \beta_i)$, then $P(\Theta|\pi)$ is just the product of the expectations of the different Beta functions $(= \frac{a_i}{a_i+b_i})$

Return To: Posterior for Trees III

- Finally, what is $P(\Pi)$? This is a prior on tree-structures. This usually some kind of prior preference (for example, for smaller trees). For example:

$$P(\Pi = \pi) \propto 2^{-|\pi|}$$

  gives a lower preference to trees of larger size ("regularisation")
- SO: we have finally have all the pieces for calculating $P(T|D)$ for a tree with a specific structure π, with known distribution (Beta/Dirichlet) over parameters for each of its leaves
- That is, if each tree $T = (\pi_T, \Theta_T)$ then we can find $P(T|D = d)$

Combinatorial Search for Class Probability Trees I

- BUT: to compute the exact estimate of $P(y' = \omega_i | \ldots)$ we need the weighted-sum of the probabilities of all possible trees (that is, all possible structures, with corresponding distributions). Then:

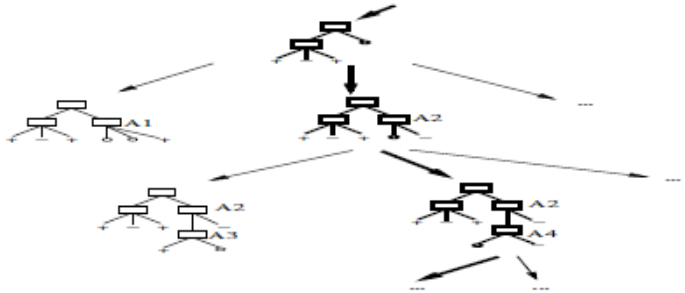
$$P(y' = \omega | \mathrm{x}', D) \;=\; \sum_T P(T|D) P(y' = \omega | \mathrm{x}', D, T)$$

  where $P(T|D)$ is as before

- But it is somewhat impractical to compute all T's. Instead, we will only approximating this using just 1 tree, or a small number of trees. Which ones?

Combinatorial Search for Class Probability Trees II

- Let us again look at this as a graph-search problem



- Each vertex in the graph is a classification tree. We will only consider the 2-class case ($\omega = \omega_1$ or ω_2), and that the x_i values are Boolean. That is, each vertex is a binary tree.
- A pair of vertices in the graph have an edge if the corresponding trees differ in just the following way: one of the leaf-nodes in one vertex has been replaced by a non-leaf node testing an x value that has not appeared earlier (and 2 leaves)

Combinatorial Search for Class Probability Trees III

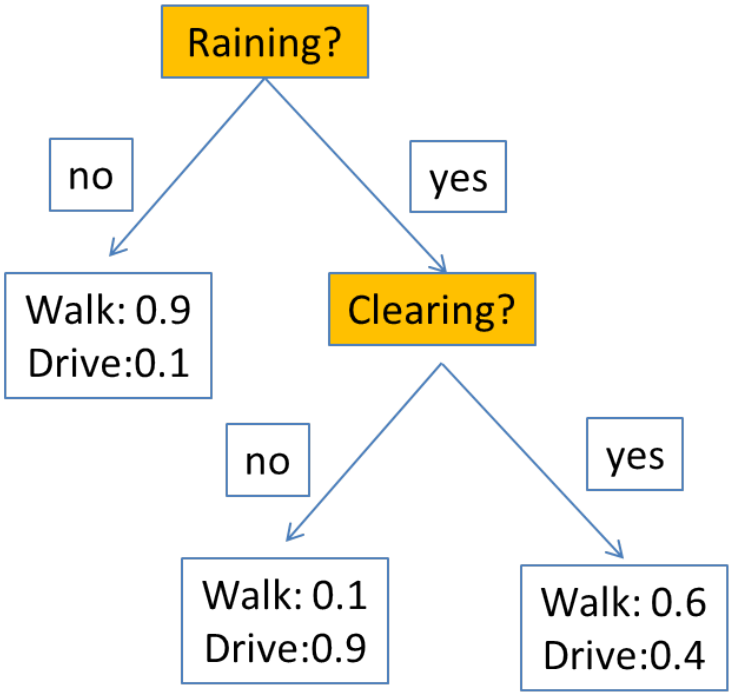
- This is the full space of all decision trees (is it?). We want to search for a single tree or a small number of trees in this space. How should we do this?
- Usual graph-search technique: greedy or beam search, starting with the vertex corresponding to the "empty tree" (single leaf node)
- Greedy choice: which one to select? The neighbour that results in the greatest increase in the likelihood $P(D|T)$
  - How?
  - Suppose T is changed to T' (that is, structure changes from π to π'). Simply use the ratio of $P(D|T')/P(D|T)$
    - If we search for trees of increasing depth, and use a depth-based prior, then $P(T)$ will increase by the same amount for all trees as we go from depth d to $d+1$.
    - So, we can ignore the ratio $P(T')/P(T)$

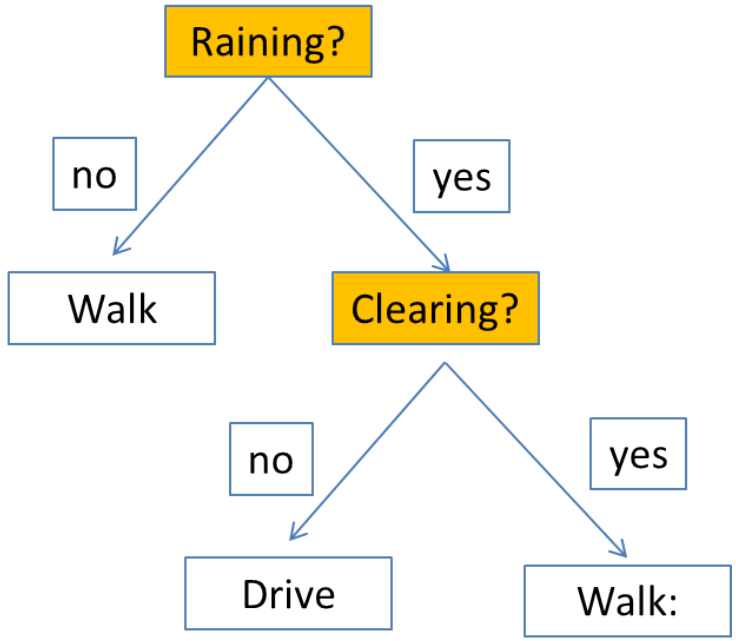
Combinatorial Search for Class Probability Trees IV

- FURTHER: Most of the calculation of $P(D|T)$ will cancel out: so, we will only need to do the local computation at the leaf that was converted into a non-leaf node

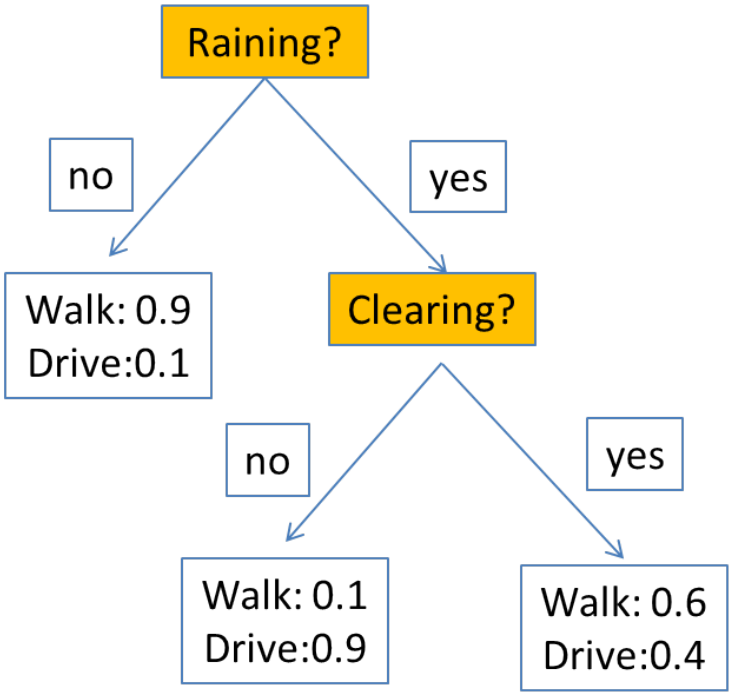
- RESULT: set of trees with (reasonably) high posterior probabilities given D: we can now use these to answer questions like $P(y' = \omega_1|\ldots)$? or even make a *decision* or a *classification* that $y' = \omega_1$, given input data x

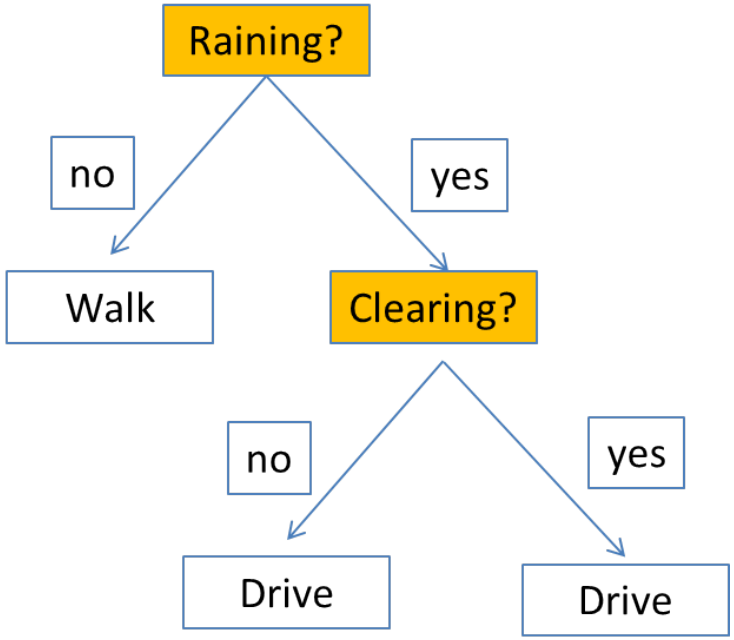
Decision Trees I



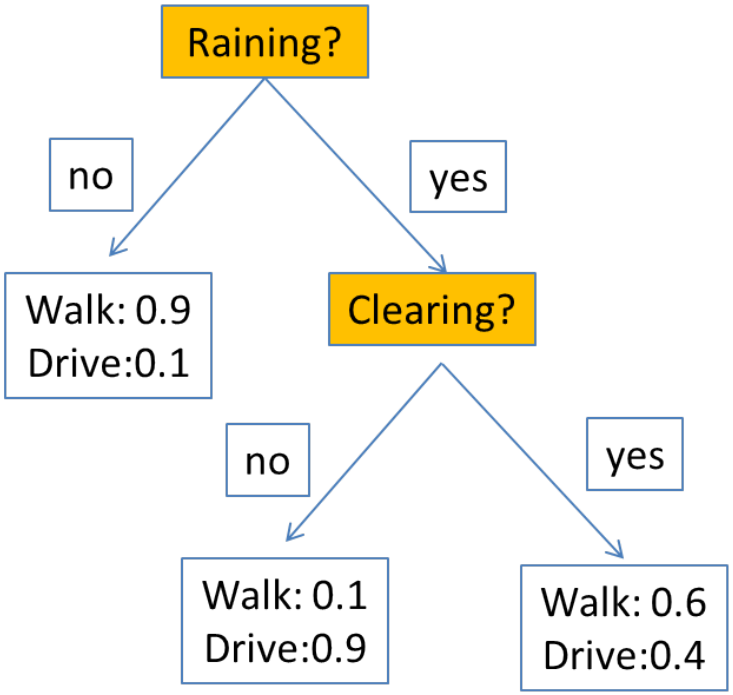


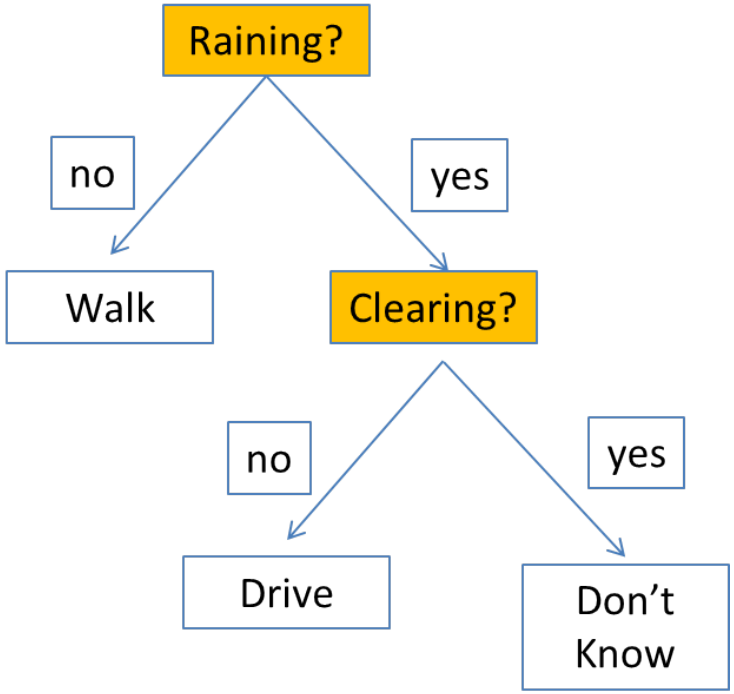
Decision Trees II





Decision Trees III





Decision Trees IV

- We can go directly to decions in the leaves, bsed on the posterior estimates of $P(Y|X)$ at the leaves. But this requires some additional machinery
  - A decision based on probability values
  - Costs of (wrong) decisions
- We will take a decision at each leaf to assign $Y = y'$ for where y' is the class with the highest probability in the leaf
- This is implicitly assuming unit costs. We can get other decision trees, using a different cost assignment (for example, assign $Y = \omega_2$ if $P(\omega_2|\mathrm{x}) > 0.2$). More on this when we look at Decision Theory.
- Assuming unit costs, the basic decision tree builder can exploit the fact that the majority class will be selected, by making greedy choices based on tests that result in leaves with (largely) one class value

Decision Trees V

- *Entropy* (as used in information theory) is widely used in a proxy for gain in posterior probability. For a 2-class problem (say the classes are *pos* and *neg*), given a set of instances S, some of which are *pos* and others *neg*:

$$Entropy(S) = -(p_{pos})\log_2(p_{pos}) - (p_{neg})\log_2(p_{neg})$$

  where: p_{pos} os the fraction of instances which are *pos* and p_{neg} is the fraction of instances which are *neg*

- For example:

$$\begin{aligned} Entropy([15+, 4-]) &= -(15/19)\log_2(15/19) - (4/19)\log_2(4/19) \\ &= 0.742 \end{aligned}$$

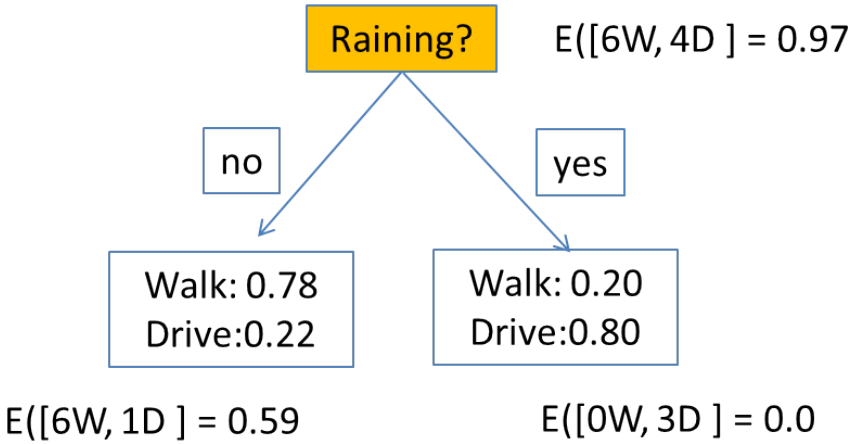
Decision Trees VI

- Entropy values are in $[0, 1]$. It is 0 is all members of S have same target value and is 1 if S contains an equal number of cases in each target class.
- Each local move in the search partitions a set of instances S in a leaf further using a test A. A local computation used as a proxy for gain in posterior probability is the *reduction* in entropy (which increases the probability of one class over the other):
- If using entropy

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

Decision Trees VII

- For example:



- So, in graph-search, go with neighbour that has highest *Gain* (or top-k neighbours)

Decision Trees VIII

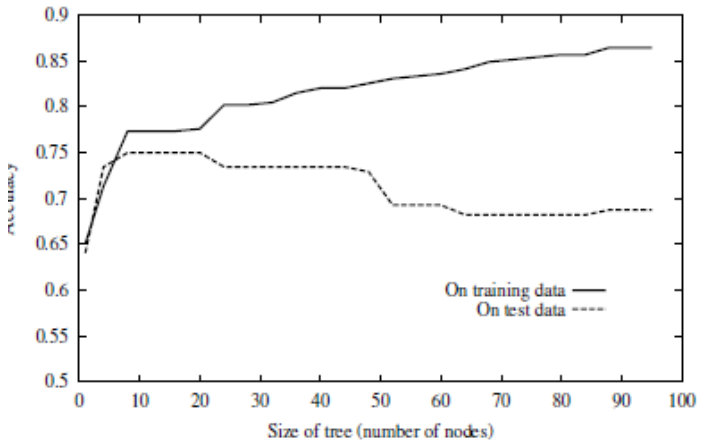
- Question:
  - What stops an entroy-based search?

Training-Estimate based Search I

- Greedy search can make mistakes. We know that it can end up in local minima — so a sub-optimal choice earlier might result in a better solution later (*i.e.* pick a test whose posterior gain (or information gain) is less than the best one
- But there is also another kind of problem. We know that training error is an optimistic estimate of the true error of the model, and that this optimism increases as the training error decreases
  - We will see why this is the case later (lectures on Evaluation)
  - Suppose we have two models M_1 and M_2 with training errors e_1 and e_2 and optimism o_1 and o_2. Let the true error of each be $E_1 = e_1 + o_1$ and $E_2 = e_2 + o_2$
  - If $e_1 < e_2$ and $E_1 > E_2$, then we will say that M_1 has overfit then training data

Training-Estimate based Search II

- So, a search method based purely on training data estimates may end overfitting the training data



- BUT WAIT: did'nt we say that picking the highest probability class will minimise the expected costs for a 0-1 loss function?
- YES: but thre are some additional conditions for that to hold including a large enough sample size, and a criterion called *uniform convergence*.

Training-Estimate based Search III

- Using the training error to estimate true loss is called *Empirical Risk Minimization*, or *ERM*. As the training set size gets large, and uniform convergence holds, the loss estimate using *ERM* can be a reasonable estimate of the true loss.
- BUT, this is a limiting result– it will not usually be the case for small samples. We will assume that in practice, the training error will be an optimistic estimate of true error
- So, what is to be done?
  - Do not guide your search on training-data estimates (use a *validation set* or *cross-validation estimates*: more later). This is called *reduced error pruning*
  - Use a correction factor, based on an estimate of *Opt*; or some statistical test (like χ^2) to stop the search. This is called *forward pruning* or *pre-pruning*

Training-Estimate based Search IV

- Do a second search, starting from the final model, going to other parts of the search graph based on validation or cross-validation estimates (techniques for *pruning decision trees* do this). This is not as expensive as using these estimates during the first search. This is called *post-pruning*.
- We will not look at pruning methods here: in effect they are ways of trying to find bbetter stopping points in the tree-space than the tree with lowest empirical risk

When are Decision Trees the Right Model?

- With Boolean values for the X and Y, the representation adopted by decision-trees allows us to represent Y as a Boolean function of the X
- Given d input Boolean variables, there are 2^d possible input values for these variables. Any specific function assigns $Y = 1$ to some subset of these, and $Y = 0$ to the rest
- Any Boolean function can be trivially represented by a tree. Each function assigns $Y = 1$ to some subset of the 2^d possible values of X. So, for each combination of values with $Y = 1$, have a path from root to a leaf with $Y = 1$. All other leaves have $Y = 0$
- This is nothing but a re-repsentation of the truth-table, and will have 2^d leaves. More compact trees may be possible, by taking into account what is common between one or more rows with the same Y value
- But, even for Boolean functions, there are some functions for which compact trees may not be possible (the parity and

Aside: Classification Rules I

- Each path from root to leaf in a decision tree can be converted into statements in logic:

$$walk \leftarrow \neg raining$$
$$drive \leftarrow (raining \wedge \neg clearing)$$
$$walk \leftarrow (raining \wedge clearing)$$

- Correctly, each rule also has an associated probability with it, and model is a set of weighted logical rules

$$p_1 : walk \leftarrow \neg raining$$
$$p_2 : drive \leftarrow (raining \wedge \neg clearing)$$
$$p_3 : walk \leftarrow (raining \wedge clearing)$$

Aside: Classification Rules II

- Each of the logical sentences can be seen as a *classification rule* that holds over some of the data
- Building decision trees is one way to find such rules. We can build such rule-sets directly: we will not look at other ways here, BUT: see next page

Aside: Classification Rules III

Questions. Here are some questions you should be able to answer about learning classification rules. Suppose you are given input data $\{(\mathrm{x}_i, y_i)\}$ in which the x_i are of d-dimensonal Boolean vectors, and y_i is a Boolean class value, and you want to construct rules of the form w : *Class* = ... **if** *Conj*, where w IS a probability, or weight; *Conj* is a conjunction of the d Boolean attributes. Then:

1. Discriminatory models consist of sets of such rules. What is the structure- and parameter-estimation problem here?
2. How many model structures can be constructed?
3. Can you describe a greedy procedure that constructs a discriminatory model 1-rule-at-a-time?

Aside: Classification Rules IV

4. Can you describe a graph showing an ordering over individual rules?

Special Case: Rules from Stumps I

- A simple rule-learner:
  - *1R* or OneR for "1-rule", uses a one-level decision-tree, (also known as a *Decision Stump*) expressed as a set of rules that test *one* attribute

```
For each attribute a
  For each value v of a make a rule:
    count how often each class appears
    find most frequent class c
    set rule to assign class c for attribute-value a = v
  Calculate error rate of rules for a
Choose set of rules with lowest error rate
```

Data set for *Weather*

outlook	temperature	humidity	windy	play
sunny	hot	high	false	no
sunny	hot	high	true	no
overcast	hot	high	false	yes
rainy	mild	high	false	yes
rainy	cool	normal	false	yes
rainy	cool	normal	true	no
overcast	cool	normal	true	yes
sunny	mild	high	false	no
sunny	cool	normal	false	yes
rainy	mild	normal	false	yes
sunny	mild	normal	true	yes
overcast	mild	high	true	yes
overcast	hot	normal	false	yes
rainy	mild	high	true	no

1R on *Weather* I

- 1R on *play*:

attribute	rules	errors	total errors
outlook	sunny $\rightarrow$ no	2/5	4/14
	overcast $\rightarrow$ yes	0/4	
	rainy $\rightarrow$ yes	2/5	
temperature	hot $\rightarrow$ no	2/4	5/14
	mild $\rightarrow$ yes	2/6	
	cool $\rightarrow$ yes	1/4	
humidity	high $\rightarrow$ no	3/7	4/14
	normal $\rightarrow$ yes	1/7	
windy	false $\rightarrow$ yes	2/8	5/14
	true $\rightarrow$ no	3/6	

- Two rules tie with the smallest number of errors, the first one is:

1R on *Weather* II

outlook:

sunny $->$ no

overcast $->$ yes

rainy $->$ yes

(10/14 instances correct)

- Things get more complicated with missing or numeric attributes
  - treat "missing" as a separate value
  - *discretize* numeric attributes by choosing breakpoints for threshold tests
  - Too many breakpoints causes overfitting, so parameter to specify minimum number of examples lying between two thresholds

1R on *Weather* III

humidity:

$< 82.5 - >$ yes

$< 95.5 - >$ no

$>= 95.5 - >$ yes

(11/14 instances correct)

Look Ahead: Ensembles of Stumps

- What do we do with these rules? Surprisingly, a very extensive study showed that for a large number of real-world problems, 1R was almost as effective as the best predictors
- Using many (1000s) of small decision trees is the basis of a class of *ensemble* methods called *Random Forests*, which uses a weighted average of the predictions from each tree
- Recall:

$$P(y' = \omega | \mathrm{x}', D) \; = \; \sum_{\mathcal{T}} P(\mathcal{T}|D) P(y' = \omega | \mathrm{x}, D, \mathcal{T})$$

- Some of the most accurate predictive models today are based on ensembles of of small trees

Look Ahead: Extensions

- So far, we have looked at mainly at building models for 2-class problems using Boolean-valued vectors. BUT:
  1. What happens if $x \in \Re^d$?
  2. What happens if $y \in \{0, 1, 2, \ldots, k\}$?
  3. What happens if we do not want axis-parallel splits?
  4. What happens if $y \in \Re$?
- We will look at these questions later

Summary

- Going from probabilities to decisions requires a *decision theory*
- Here we have seen how an example of this by first learning the parameters and structure of a *class probability tree*, and how to convert that into a decision tree using a loss function
- A decision tree is one way to build *classification models*. These can be seen as special kinds of rules in logic
- Classification models can also be seen as a set of rules. We can find a model for data by searching for the for the set of rules with highest posterior probability. In practice, this will require a greedy search