CS189
decision trees

Midtem: everything up until this lecture

decision tree: nonlinear method used for classification
- uses trees w/ 2 node types
  1) internal nodes test feature values (usually just one)
     - branch after
  2) leaf nodes
     - predict the class or regression value $h(x)$

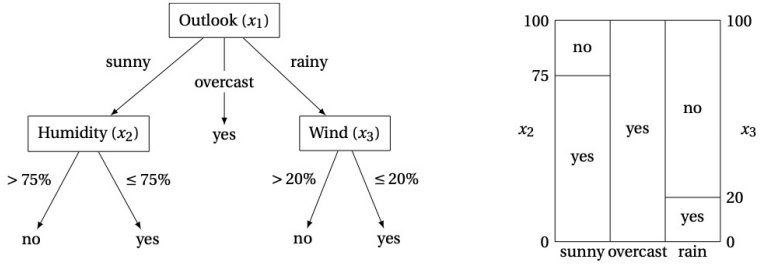
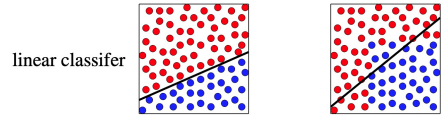
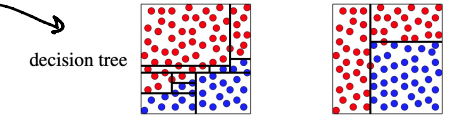


Figure 14.1: An example decision tree for when to go out for a picnic

- cuts X-space into rectangles
- works well w/ categorical and quantitative features
- interpretable results
- decision boundary can be arbitrarily complicated
- fast: only 1 feature per node



need more decision rules



Classification w/ Decision Trees:

- greedy, top-down heuristic
- recursive

let $S \subseteq \{1 \dots n\}$ be set of sample point indices

top-level call has $S = \{1 \dots n\}$ (consider all nodes)

GrowTree(S):

```
if (y_i = C for all i ∈ S and some class C) {
    return new leaf(C)        (leaves are pure → represent exactly 1 class)
} else {
    choose best splitting feature j and splitting value β
    S_ℓ = {i ∈ S : x_ij < β}        (≤ > also work)
    S_r = {i ∈ S : x_ij ≥ β}
    return new node(j, β, GrowTree(S_ℓ), GrowTree(S_r))
}
```

how do we choose the best split?

- try all splits:
  - for each set: $J(S)$ = cost of S
  - choose split that minimizes $J(S_\ell) + J(S_r)$
    - or, split that minimizes weighted avg. by # training points of child

(cardinality $|\cdot|$)

$$\frac{|S_\ell| J(S_\ell) + |S_r| J(S_r)}{|S_\ell| + |S_r| = |S|}$$

How do we choose $J(S)$?

Bad idea: label S w/ the class C that labels the most points in S

↳ $J(S)$ = # of pts. in S not in class C

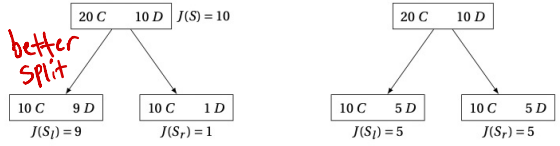


Figure 14.3: Two possible splits of a set S

Problem: $J(S_\ell) + J(S_r)$ same for both splits (many splits will have same total cost)

2nd idea: entropy

let Y be a random class variable, $P(Y=c) = p_c$

Surprise of Y being class C: $-\log_2 p_c$ (always nonnegative)

- ↳ event w/ prob 1 → 0 surprise
- ↳ event w/ prob 0 → infinite surprise

might involve fractional bits
↳ useful when compiling lots of events

Surprise: expected number of bits of information needed to transmit (information theory def.) which events happened to a recipient who knows the probabilities of the events

entropy of index set S is the **average surprise:**

$$H(S) = -\sum_c p_c \log_2 p_c, \qquad p_c = \frac{|\{i \in S : y_i = c\}|}{|S|}$$

↳ proportion of points in S that are class C

example scenarios:

- all points in S belong to same class? $H(S) = -1 \log_2 1 = 0$
- half class C, half class D $H(S) = -1/2 \log_2 1/2 - 1/2 \log_2 1/2 = 1$
- n points, all different classes: $H(S) = -\log_2 1/n = -\log_2 1 + \log_2 n = \log_2 n$

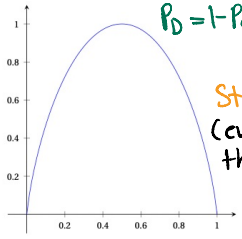
$$-\sum_{i=1}^{n} \frac{1}{n} \log_2 \frac{1}{n} = -\log_2 \frac{1}{n}$$

Entropy: expected # bits needed to transmit the class of sample point in S chosen uniformly at random

e.g. all points are class C → 0 bits needed
half C & half D → 1 bit
Specify one of N classes: $\log_2 N$

only need 1 dependent variable

$p_D = 1 - p_C$



Strictly concave (even w/ more than 2 classes)

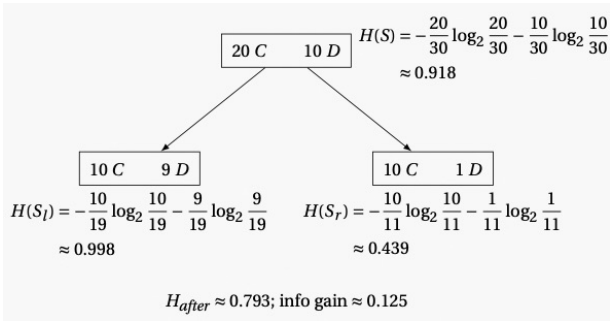
Figure 14.4: Plot of the binary entropy function $H(p_C) = -p_C \log_2 p_C - (1 - p_C) \log_2 p_C$ (2 classes)

Weighted avg. entropy: $H_{after} = \frac{|S_l| H(S_l) + |S_r| H(S_r)}{|S_l| + |S_r|}$
(after a split)

→ choose split that maximizes 'information gain' $H(S) - H_{after}$
(same as minimizing H_{after} since $H(S)$ is a constant w.r.t. split)

Info. gain always positive except: - one child is empty (0)
or
$-P(Y_i = C \mid i \in S_l) = P(Y_i = C \mid i \in S_r)$

split 2



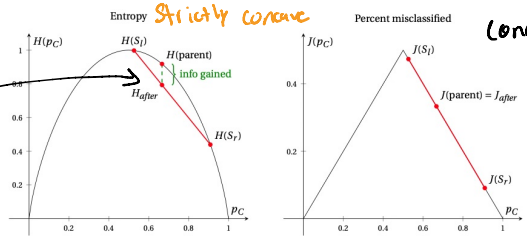


Figure 14.5: Comparison between the entropy function and the percent misclassified cost

Strictly concave

Concave, not strict

any point on line segment represents weighted average of entropies for suitable weights

info. gain: difference between weighted avg. entropy & parent entropy (directly above) } entropy
↳ always positive unless 2 child sets both have same p_C and lie at same point on the curve

W/ percent misclassified fn, curve isn't strictly concave (segment connecting pts. may lie on curve)
↳ split doesn't change # misclassified pts nor % pts. misclassified

biggest problem: many different splits can get same weighted avg → cost fn. doesn't distinguish quality of different splits well

*entropy isn't the only cost fn. that works well, other concave ones like polynomial $p(1-p)$ work

Choosing a Split:

- if X_i is a binary feature, children are $X_i = 0$ & $X_i = 1$
- if X_i has 3 or more discrete features, then the split depends on the application
- if X_i is quantitative:
  1) Sort values of X_i in S
  2) Split between each pair of unequal consecutive values
  3) take best split at very end

  Efficient: can use radix sort in linear time
  ↳ def use if n is huge

Note: as you scan sorted list left to right, update entropy in $O(1)$ time per point
↳ important for obtaining fast tree-building time

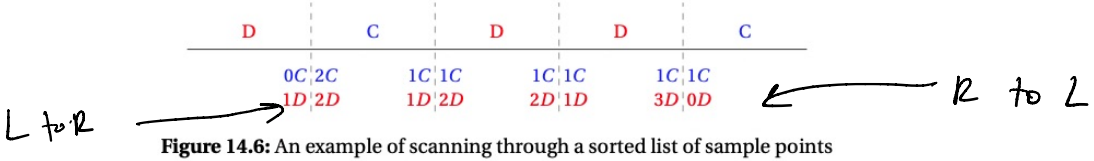


Figure 14.6: An example of scanning through a sorted list of sample points

Updating entropies: - C := # of class C sample points to the left of a potential split
- $\bar{C}$:= # sample pts. to right of split
- D = # points not in C to left of split
- d = # pts. not in C to right of split.

$$\rightarrow H(S_l) = \frac{-C}{C+D}\log_2\frac{C}{C+D} - \frac{D}{C+D}\log_2\frac{D}{C+D}$$

$$H(S_r) = \frac{-\bar{C}}{\bar{C}+d}\log_2\frac{\bar{C}}{\bar{C}+d} - \frac{d}{\bar{C}+d}\log_2\frac{d}{\bar{C}+d}$$

* $\log_2 0$ undefined, but ← we suppose $0\log_2 0 = 0$

choose split that minimizes weighted avg.

Weighted avg. $\Rightarrow \frac{-1}{n'}\left(C\log_2\frac{C}{C+D} + D\log_2\frac{D}{C+D} + \bar{C}\log_2\frac{\bar{C}}{\bar{C}+d} + d\log_2\frac{d}{\bar{C}+d}\right)$, $n' = C+D+\bar{C}+d$ = # pts. in tree node

Algorithms & Runtimes:

Classifying a test point: Walk down tree until leaf. return leaf's label
↳ Worst case: $O(\text{depth})$
↳ binary features: depth $\leq d$ (quantitative features may go deeper)
- Usually (not always): $\leq O(\log_2 n)$

Training:
Binary Features: try $O(d)$ splits at each node
- try all features $O(d)$, constant number $O(1)$ splits per feature
Quantitative features: try $O(n'd)$ splits, $n' = \#$ pts. in node
- try all features $O(d)$, try all $O(n')$ splits per feature
↳ i.e. scan through pts. linearly after radix sort
Both cases: $O(n'd)$ time to find & perform a split
↳ W binary features, still need $O(n')$ time to split sample pts.
↳ quantitative features just as fast due to clever way of computing entropy for each split
Each pt. participates in $O(\text{depth})$ nodes, $O(d)$ time in each node
↳ amortized analysis, charging $O(d \cdot \text{depth})$ time to each pt.

Running time $\leq O(nd \cdot \text{depth})$, nd size of design matrix, logarithmic depth