Machine Learning

CS 165B

Prof. Matthew Turk

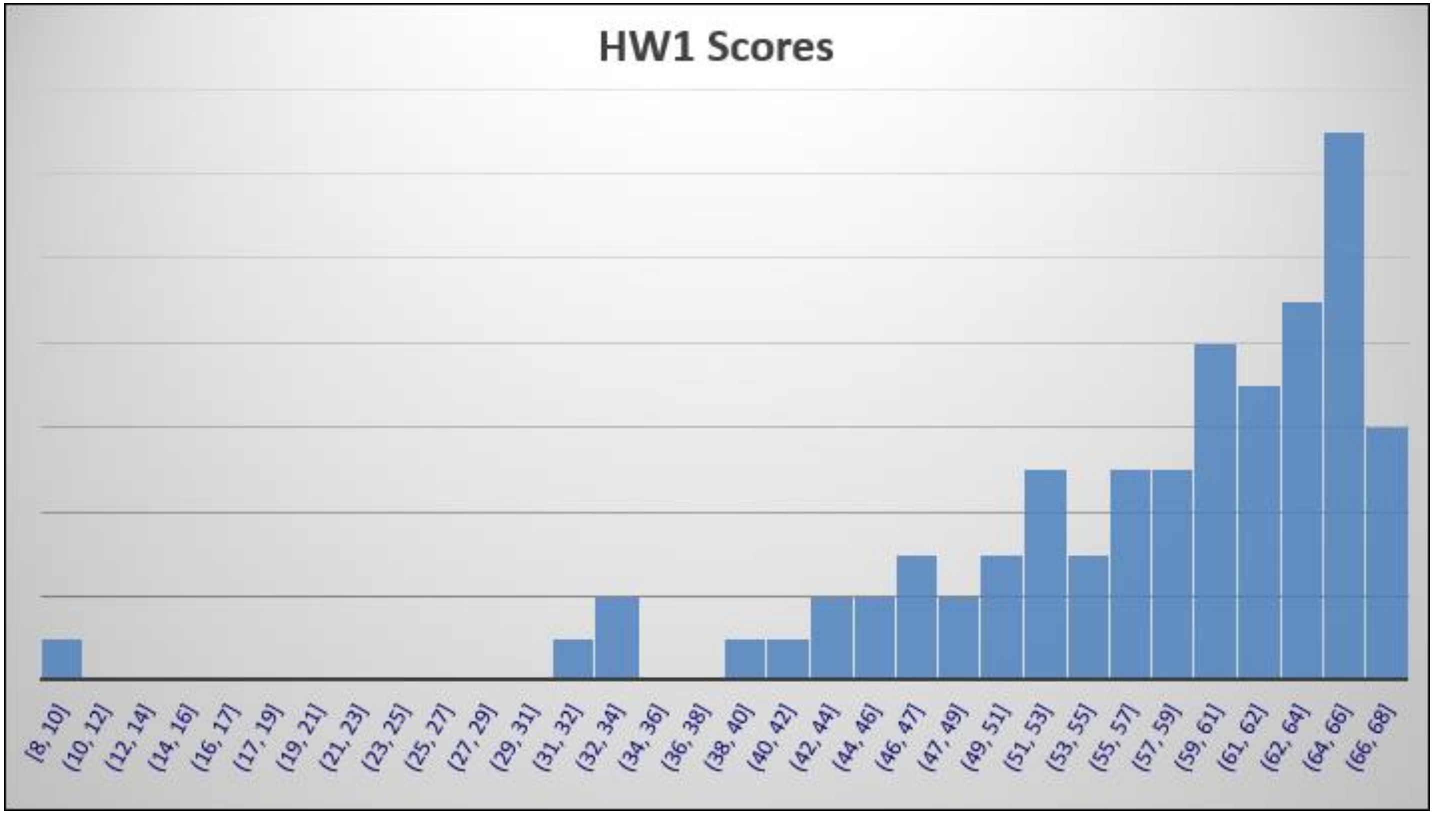
Wednesday, April 20, 2016

Today

- Decision trees
- Linear learning models (Ch 7)

Notes

- HW#1 scores
  - Ave = 56.5/68 = 83%
  - Median = 59/68 = 87%

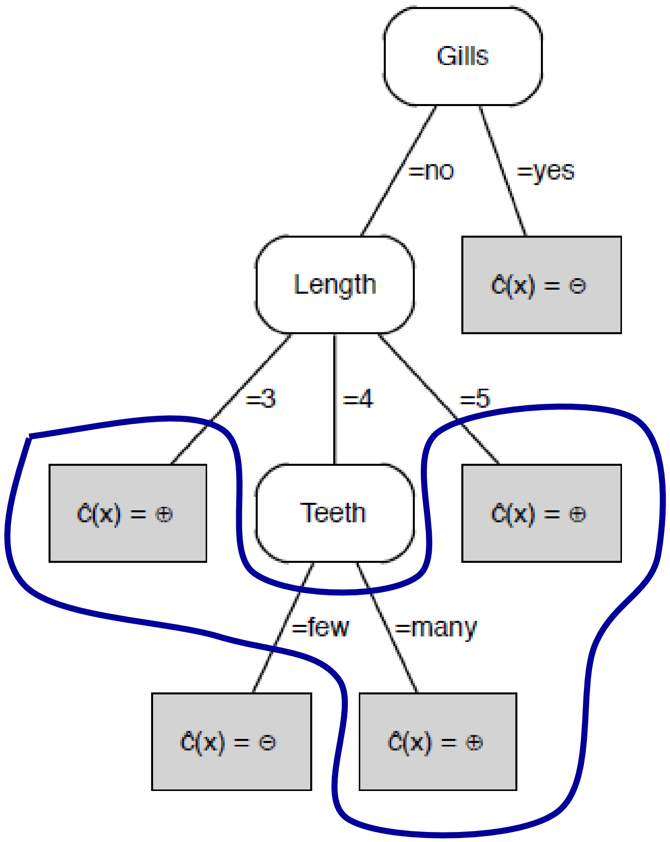


Notes

- HW#2 due 4:30pm Friday
- Midterm – Monday, May 2, in class
  - Covers material through next Wednesday
  - Brief review in class next Wednesday
  - Practice midterm will be supplied
  - Closed book/notes
    - Exception: You may bring one 8.5”x11” sheet of paper with your notes (both sides)
    - I’ll also provide some information, formulas, etc. (will be included with the practice midterm)
- GauchoSpace expiration notifications…

Decision trees

- A decision tree partitions the instance space by branching on feature values (literals), with leaves representing the learned concept
- Each leaf represents a conjunction of literals on its path
- The learned concept is the disjunction of the positive leaves
  - $L_1 \vee L_2 \vee L_3 \vee \ldots$
- Decision trees are maximally expressive – they can separate any consistently labeled data
  - Thus more powerful than the conjunctive hypothesis space we just discussed



Ideally, each leaf contains <u>only positives</u> or <u>only negatives</u> from the training data

Key question: Which features (and in what order) will accomplish this best?

Decision trees

Note: This is very similar to ID3, a well-known DT algorithm – in yesterday's discussion session

- Tree models can be used for classification, ranking, probability estimation, regression, and clustering
- Recursive generic tree learning procedure:

100%?
99%?
80%?

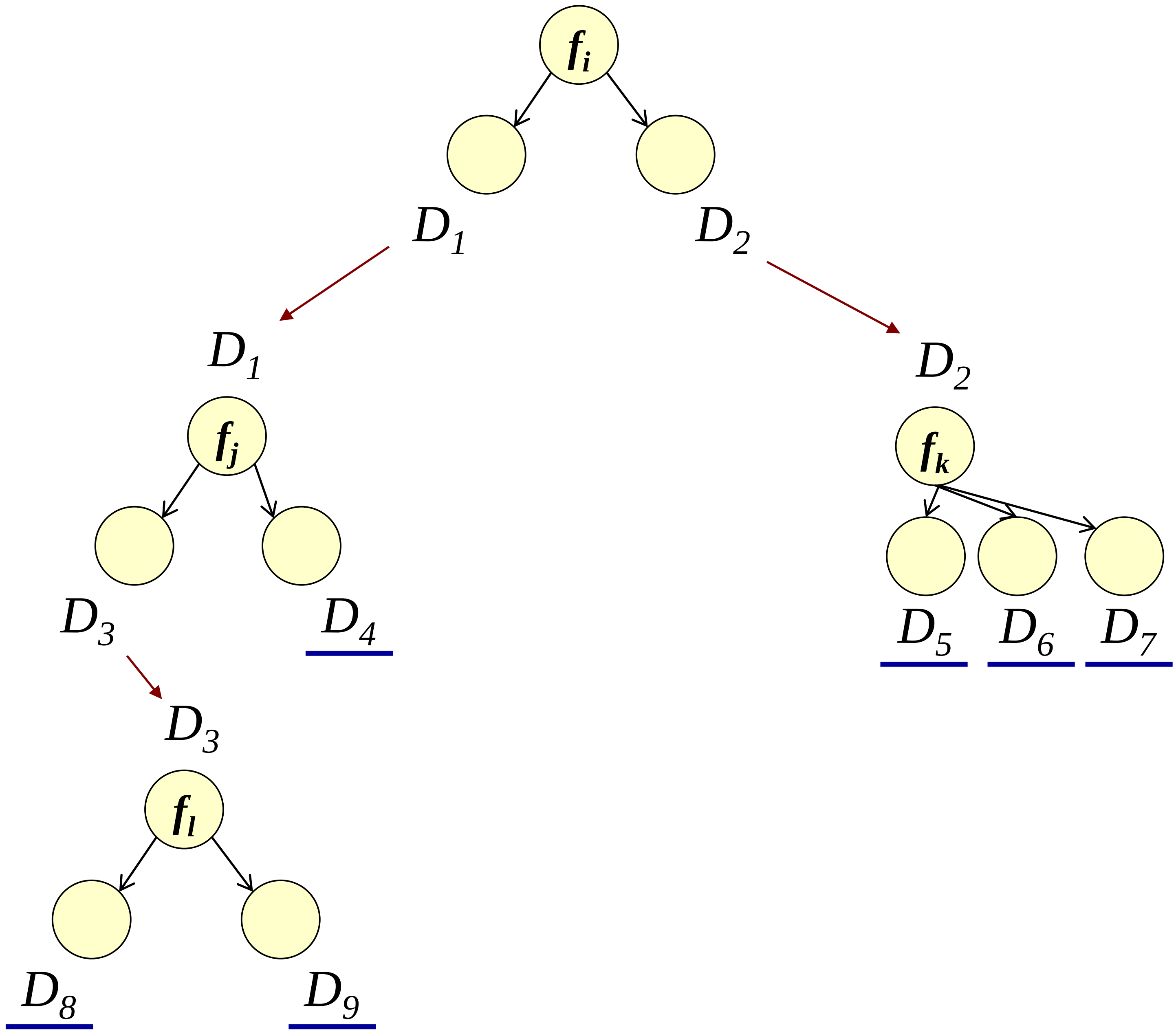
Most useful feature

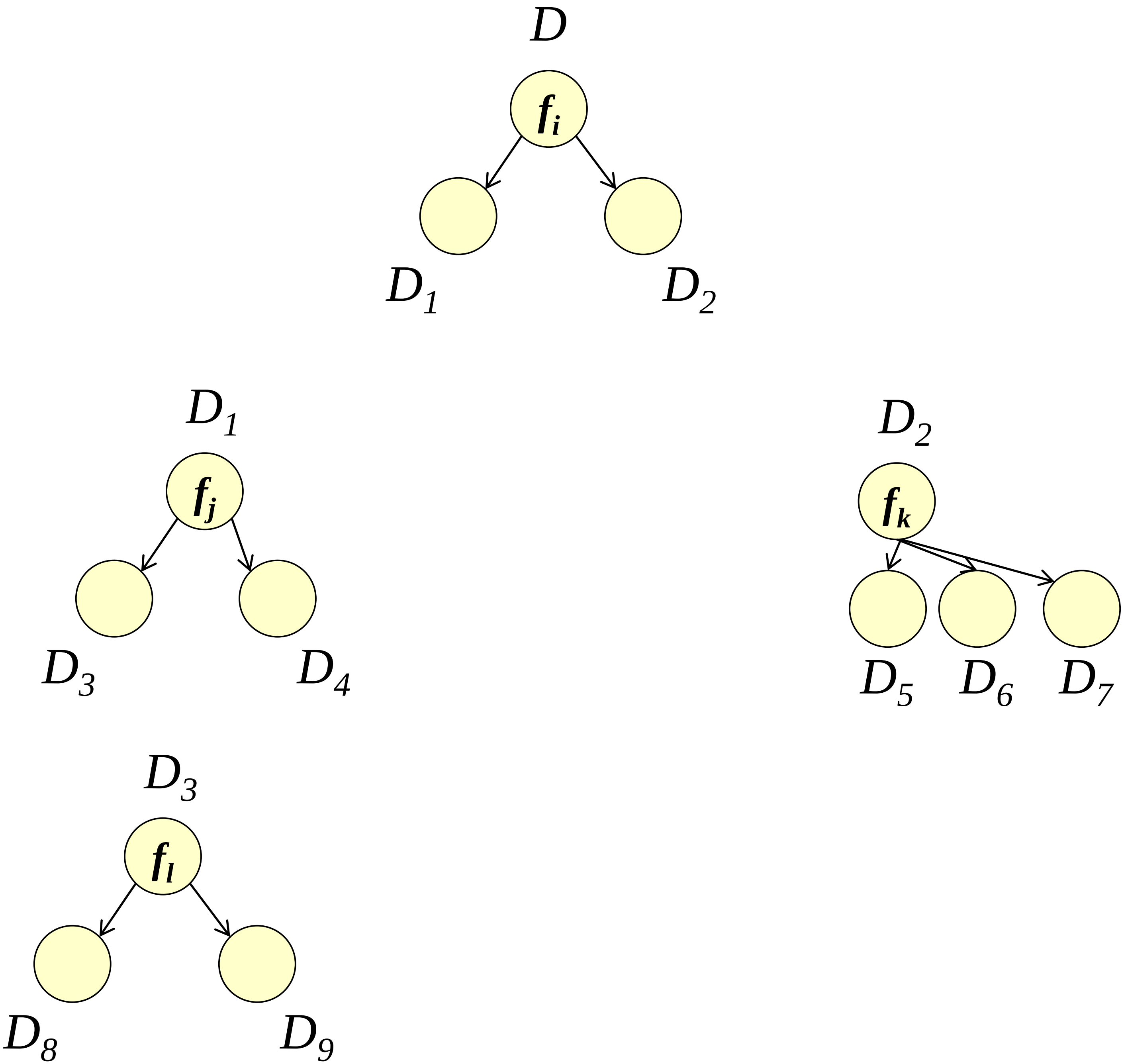
Algorithm GrowTree(D, F) – grow a feature tree from training data.

Input : data D; set of features F.
Output : feature tree T with labelled leaves.
if Homogeneous(D) **then return** Label(D);
$S \leftarrow$ BestSplit(D, F) ; // e.g., BestSplit-Class (Algorithm 5.2)
split D into subsets D_i according to the literals in S;
for each i **do**
  if $D_i \neq \emptyset$ **then** $T_i \leftarrow$ GrowTree(D_i, F) ;
  else T_i is a leaf labelled with Label(D);
end
return a tree whose root is labelled with S and whose children are T_i

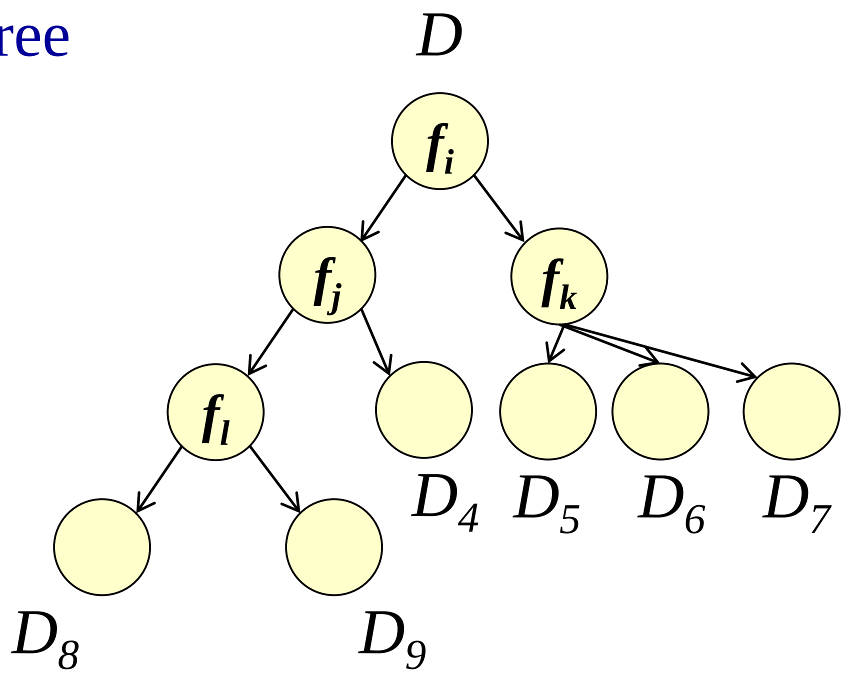
Divide-and-conquer approach: build a tree for each subset of the data, then merge into a single tree

Training Data D *Features* $F = \{ f_1, f_2, \dots \}$




D
f_i
D_1
D_2
D_1
f_j
D_3
D_4
D_2
f_k
D_5
D_6
D_7
D_3
f_l
D_8
D_9

Feature Tree



Decision trees

Note: This is very similar to ID3, a well-known DT algorithm – in yesterday's discussion session

- Tree models can be used for classification, ranking, probability estimation, regression, and clustering
- Recursive generic tree learning procedure:

100%?
99%?
80%?

Most useful feature

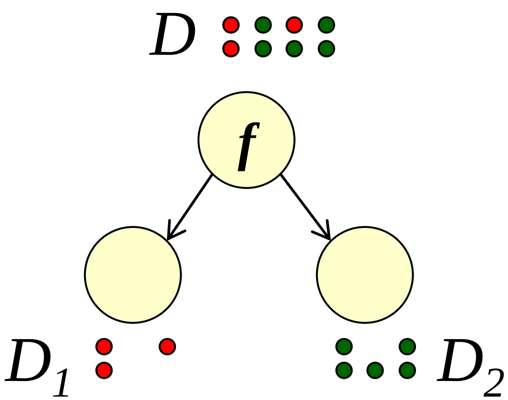
Algorithm GrowTree(D, F) – grow a feature tree from training data.

Input : data D; set of features F.
Output : feature tree T with labelled leaves.
if Homogeneous(D) **then return** Label(D);
$S \leftarrow \boxed{\text{BestSplit}(D,F)}$; // e.g., BestSplit-Class (Algorithm 5.2)
split D into subsets D_i according to the literals in S;
for each i **do**
  if $D_i \neq \emptyset$ **then** $T_i \leftarrow$ GrowTree(D_i, F) ;
  else T_i is a leaf labelled with Label(D);
end
return a tree whose root is labelled with S and whose children are T_i

Divide-and-conquer approach: build a tree for each subset of the data, then merge into a single tree

BestSplit

- BestSplit(D, F) – what feature $f \in F$ will produce the best split (partitioning) of the training data $D = \{ D_i \}$?
- What's a good split/partitioning?
  - One that produces **pure** partitions D_i, each of which contains only instances from a single class
  - E.g., in a binary classification problem, D_1 has only positive examples and D_2 has only negative examples



Both partitions are homogeneous

Pure negative

Pure positive

Which feature ***f*** is best for this?

How to measure partition purity if not completely homogeneous?

Impurity

- In the binary case, we have P positives and N negatives in the data
  - The best split would be a feature that divides the data D into two pure partitions: D_1 with the P positives and D_2 with the N negatives
- So a measure of partition impurity should be minimum when the data are 100% positives or negatives, and maximum when 50/50
- Like with empirical probabilities, we can estimate impurity by counting. We define the proportion of positives in D_i as:

$$\dot{p} = \frac{P}{P+N}$$

- Impurity is a function of $\dot{p}$
  - Should be zero when $\dot{p} = 0$ or 1, and maximum when $\dot{p} = 0.5$
  - We can write impurity as $\mathrm{Imp}(D)$ or $\mathrm{Imp}(\dot{p})$

Impurity functions

Minority class

$$\mathrm{Imp}(\dot{p}) = \min(\dot{p}, 1-\dot{p})$$

Gini index

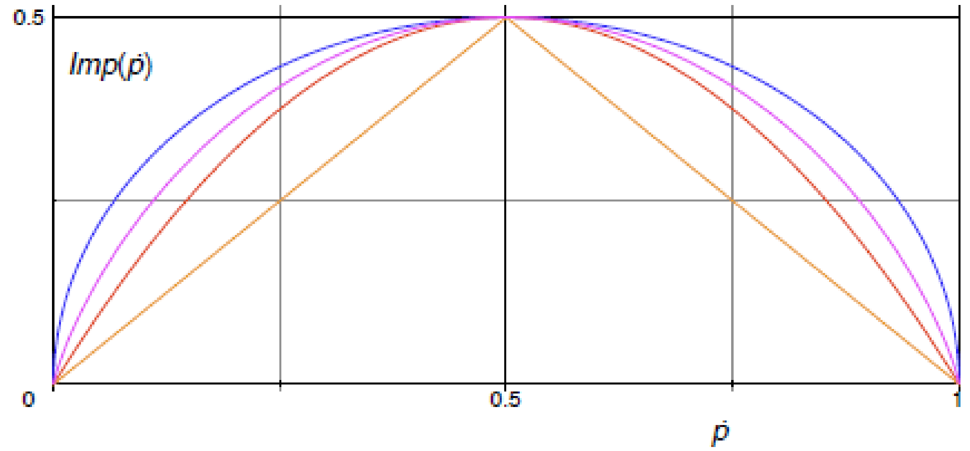
$$\mathrm{Imp}(\dot{p}) = 2\dot{p}(1-\dot{p})$$

Entropy

$$\mathrm{Imp}(\dot{p}) = -\dot{p}\log_2(\dot{p}) - (1-\dot{p})\log_2(1-\dot{p})$$

√Gini index

$$\mathrm{Imp}(\dot{p}) = \sqrt{2\dot{p}(1-\dot{p})}$$



The total impurity for a data partitioning is just the weighted sum of each partition's impurity

$$\mathrm{Imp}(\{D_1, \ldots, D_l\}) = \sum_{i=1}^{l} \frac{|D_i|}{|D|} Imp(D_i)$$

Impurity functions for $k > 2$

For more than two classes, the impurity functions are defined by the sum of the per-class impurities based on "one versus rest"

k-class Entropy

$$\mathrm{Imp}(\dot{p}_1,\ldots,\dot{p}_k) = \sum_{i=1}^{k} -\dot{p}_i \log_2(\dot{p}_i) \qquad \text{where} \quad \dot{p}_i = \frac{C_i}{\sum_{i=1}^{k} C_i}$$

k-class Gini index

$$\mathrm{Imp}(\dot{p}) = \sum_{i=1}^{k} \dot{p}_i (1 - \dot{p}_i)$$

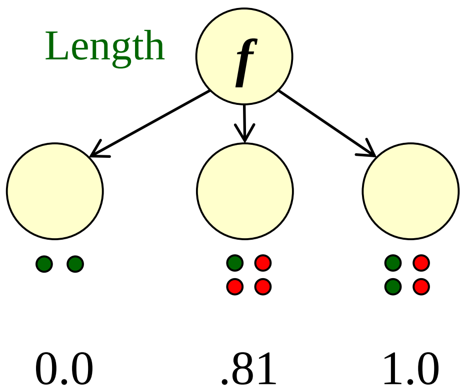
To split a parent node D into children $D_1,\ldots,D_L$ we can consider the purity gain = $\mathrm{Imp}(D) - \mathrm{Imp}(\{D_1,\ldots,D_L\})$

Reminder: What is k? What is L?

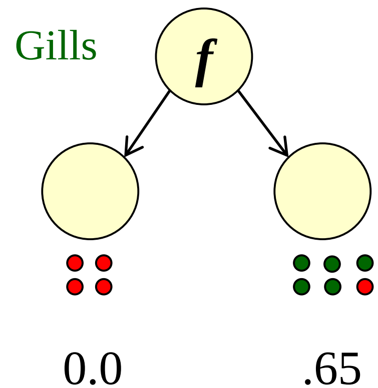
The number of classes

The number of values (literals) for a given feature F_i

Impurity example D



$$\frac{2(0.0)+4(.81)+4(1.0)}{10} = 0.72$$

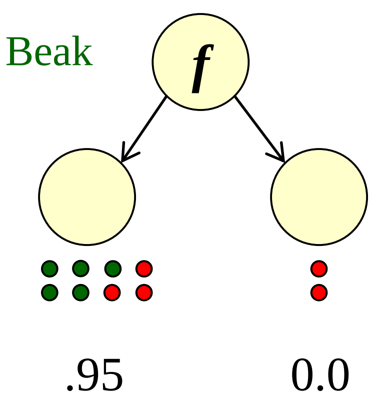


$$\frac{4(0.0)+6(.65)}{10} = 0.39$$

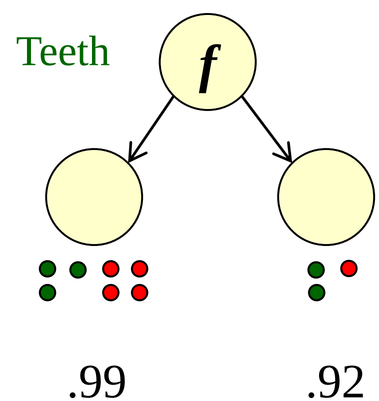
Using the entropy measure

$\text{Imp}(\dot{p}) =$
$- \dot{p} \log_2(\dot{p}) - (1-\dot{p}) \log_2(1-\dot{p})$

$$\text{Imp}(\{D_1, \dots, D_l\}) = \sum_{i=1}^{l} \frac{|D_i|}{|D|} Imp(D_i)$$



$$\frac{8(.95)+2(0.0)}{10} = 0.76$$



$$\frac{7(.99)+3(.92)}{10} = 0.97$$

Which of these is the best feature to use?

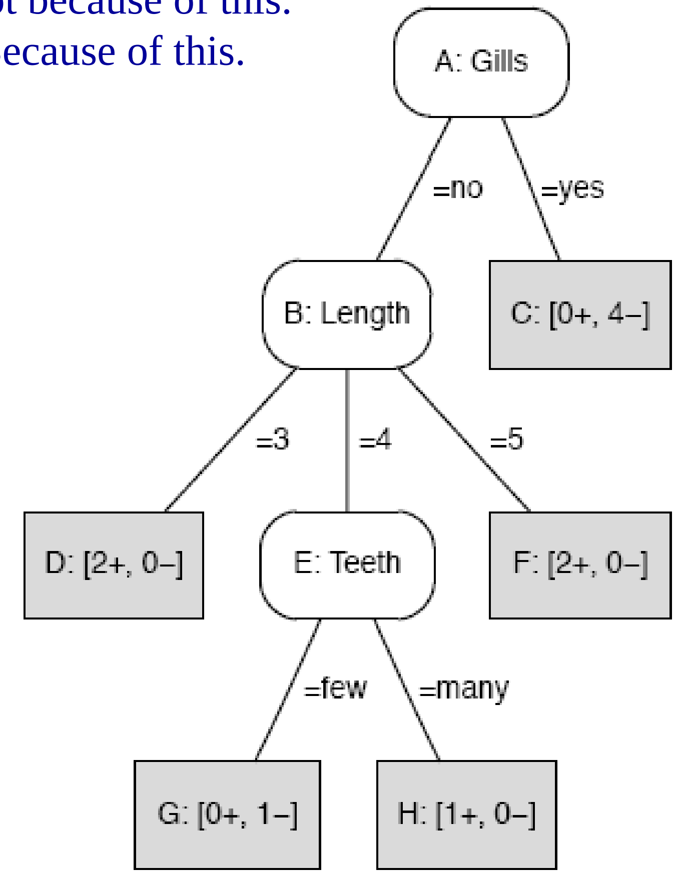
This is the Gills feature in our dolphin example

DT for dolphin example

Why is Gills good?
Not because of this.
Because of this.

Training data:

1. Length = 3 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = many
2. Length = 4 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = many
3. Length = 3 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = few
4. Length = 5 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = many
5. Length = 5 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = few
6. Length = 5 $\wedge$ Gills = yes $\wedge$ Beak = yes $\wedge$ Teeth = many
7. Length = 4 $\wedge$ Gills = yes $\wedge$ Beak = yes $\wedge$ Teeth = many
8. Length = 5 $\wedge$ Gills = yes $\wedge$ Beak = no $\wedge$ Teeth = many
9. Length = 4 $\wedge$ Gills = yes $\wedge$ Beak = no $\wedge$ Teeth = many
10. Length = 4 $\wedge$ Gills = no $\wedge$ Beak = yes $\wedge$ Teeth = few



Choose the best feature based on minimizing impurity of the remaining data

$$\mathrm{Imp}(\dot{p}) = -\,\dot{p}\log_2(\dot{p}) - (1-\dot{p})\log_2(1-\dot{p})$$

DT approach

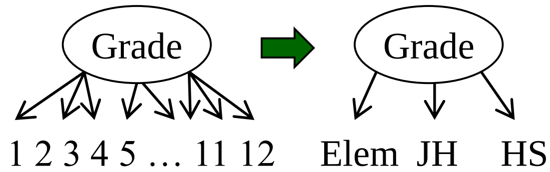
- We've described a greedy algorithm – it maximizes each individual choice, but it does not guarantee a global maximum
  - For this we would need the ability to backtrack and reconsider choices based on the total impurity

$$\mathrm{Imp}(\{D_1, \ldots, D_l\}) = \sum_{i=1}^{l} \frac{|D_i|}{|D|} Imp(D_i)$$

- However, it works rather well in practice!
- We can modify the strategy slightly to deal with "messy" (non-separable) data and to limit the size of the tree
  - By not requiring a homogeneous data partition before stopping and assigning a label – i.e., the Homogeneous(D) function
  - E.g., if we have a feature separates as {1000+, 3–}, that may be good enough – no need to keep checking additional features

Simplifying decision trees

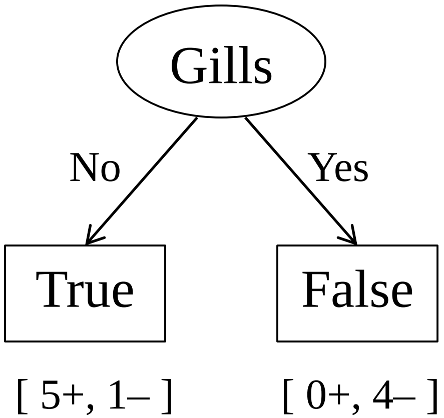
- Some ways to create simpler decision trees:
  - Merge feature labels and test the difference
  - Enforce a depth limit (maximum depth of ***d***)
  - Enforce a purity threshold – e.g., if the impurity of a node is < ε, turn the node into a leaf (don't expand it further)
  - Enforce a purity increment threshold – e.g., if a node expansion increases purity by less than δ, delete the expansion and turn the node into a leaf
  - Build the complete tree and then iteratively merge leaves based on lowest purity decrease up to a number of leaves N or a purity δ
  - Combinations of depth and purity measures



- If you took 165A, this should remind you of search strategies!
  - In fact, we've been discussion various ways of searching the hypothesis space to come up with a "good" hypothesis based on our data

Simplifying decision trees

We could simplify the dolphin decision tree to this:



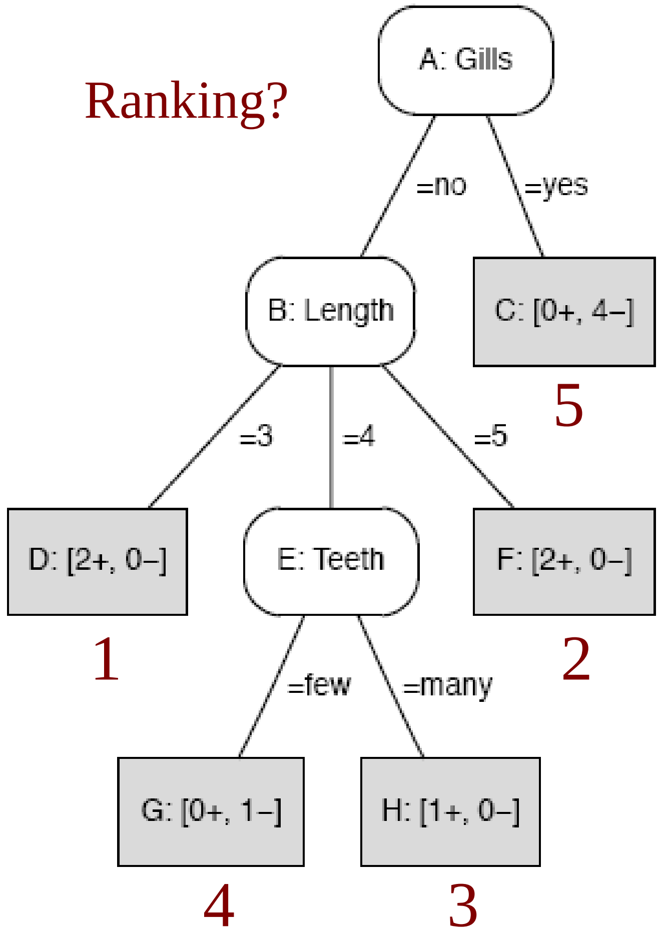
Does this generalize well?

Ranking trees

Note: We can’t tell the ranking from the tree structure; only from leaves and their data

- Since a decision tree divides the instance space into segments and we have data for each segment, we can turn the DT into a ranking model by evaluating and ordering the segments
- As before, we use empirical probabilities $\dot{p}$ – for segments i and j, order $i > j$ if $\dot{p}_i > \dot{p}_j$
  - May use Laplace correction or m-estimate for smoothing
- As before, we can computing ranking error rate and accuracy

Ranking?



(This would be a better example if it had more data and non-homogeneous leaves!)

Ranking trees

Dolphin Ranking: D F H G C

- Ranking is with respect to a particular class (e.g., *dolphin*)
- A ranking is on a set of *m* instances $X = \{ x_1, \ldots, x_m \}$
- A decision tree with N leaves will have N different ranks
  - Each instance will have one of those ranks, 1..N
  - So there are likely to be many ties if N is small or *m* is large

		Leaf
1.	Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many	**D**
2.	Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many	**H**
3.	Length = 3 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few	**D**
4.	Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = many	**F**
5.	Length = 5 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few	**F**
6.	Length = 5 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many	**C**
7.	Length = 4 ∧ Gills = yes ∧ Beak = yes ∧ Teeth = many	**C**
8.	Length = 5 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many	**C**
9.	Length = 4 ∧ Gills = yes ∧ Beak = no ∧ Teeth = many	**C**
10.	Length = 4 ∧ Gills = no ∧ Beak = yes ∧ Teeth = few	**G**

From the leaf rankings on the previous slide, the ranking of the 10 instances is:

Highest						Lowest
	1, 3	4, 5	2	10	6, 7, 8, 9	

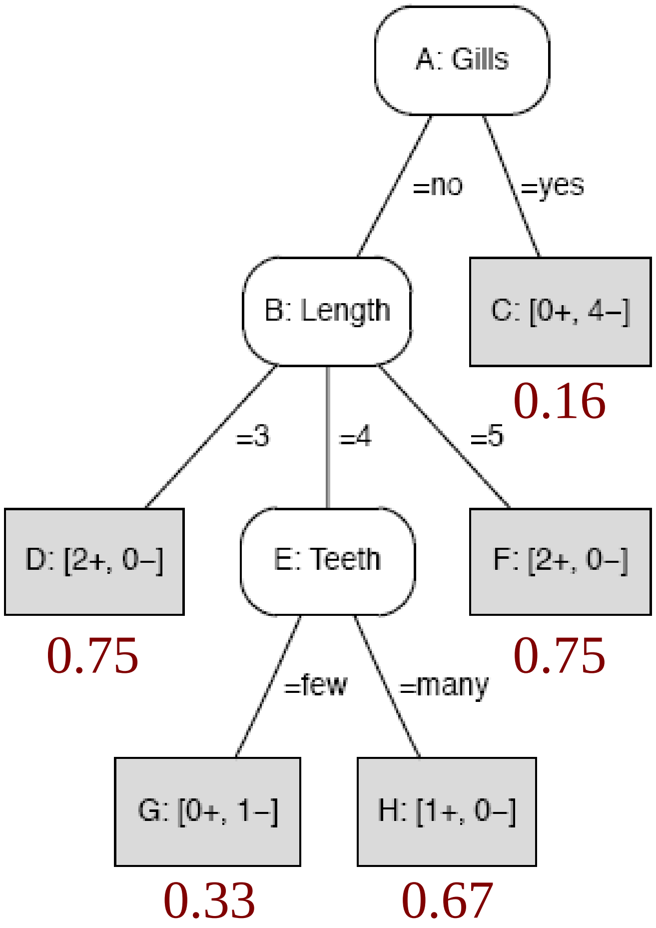
Ties

For this, accuracy = 100%

Probability estimation trees

- We can use those empirical probabilities (for each class) calculated for every leaf to create a probability estimation tree
  - A probability classifier, a.k.a. probability estimation
  - Since it's a 2-class problem, we can just show the probability for *dolphin*
- Using Laplace correction, what are the leaf probability estimates?

$$\text{Laplace correction} = \frac{N_i+1}{|S|+k}$$



P(dolphin) = { 0.75, 0.75, 0.67, 0.33, 0.16 }

Actually showing P(dolphin=true | leaf) or P(hypothesis | data)

P(¬dolphin) = { 0.25, 0.25, 0.33, 0.67, 0.84 }

Logical models – summary

- In concept learning and decision trees, we've mostly been discussing logical models, based on simple predicate (or first-order) logic
- Pros:
  - English-readable data
  - Intuitive representation and models for people to comprehend
  - Good for explaining the decision-making process
  - Some errors are obvious, easily found and debugged
  - Well suited for some problems
- Cons:
  - Not a good fit for massive amounts of data, for numerical data, for subtle concepts, for things that are difficult to articulate in language
    - I.e., for many of the most important problems ML is being applied to these days!

Where we're going from here

- From logical models back to geometry models and then on to probabilistic models
- There are many uses of logical models, especially decision trees, in machine learning applications
  - DTs are used in many current, practical machine learning systems
- But the focus for some time has moved toward methods that can crunch large amounts of numbers – more and more to statistical and probabilistic models and methods
- We'll continue to focus on classification and regression, as well as clustering, and we'll address these problems with a variety of "modern ML" tools and techniques
- Building a linear classifier may seem like a long way from creating intelligent machines that learn and think – but it's not as far as you may think!