k-Nearest Neighbor

k-Nearest Neighbor

- In k-NN, given a labeled training set
- And, given a point that we want to classify
  - That is, a point not in training set
- We'll let training data "**vote**"
- Who gets to vote?
  - That is, which data points in the training set get to vote?
- And how to count (weight) the votes?

k-NN

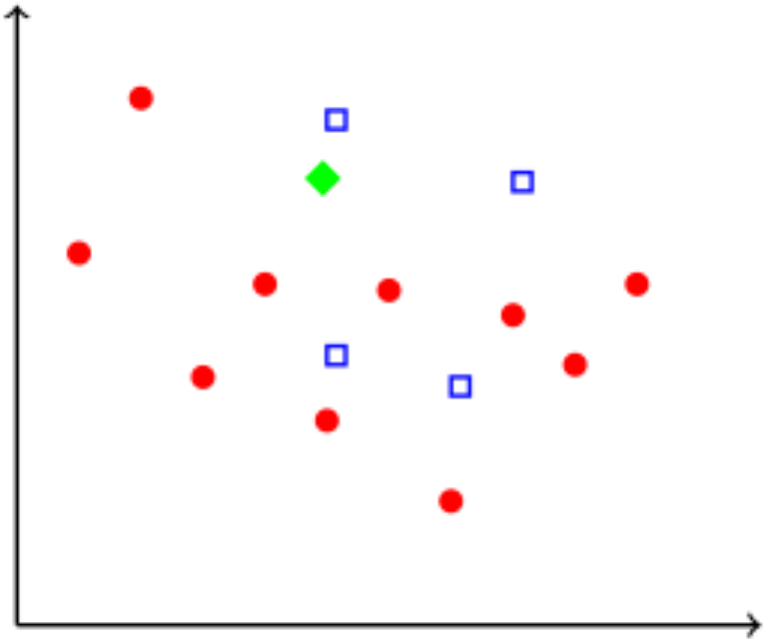
- Which data points get to vote?
  - If it's a "national" election, and majority rules, the most numerous class always wins
  - That's not very informative, so no universal suffrage in **k-NN**
- But what about "local" elections?
  - Only **the training data nearby** gets to vote
  - This might be more useful…

k-Nearest Neighbors

- Given a set of labeled training data…
- And given a point to classify
- Classify based on **k nearest neighbors**
  - Where "neighbors" are in training set
  - Value of k is specified in advance
- The simplest ML method known to man
  - And, not to mention, woman
  - Simple wrt training, since **no training**…

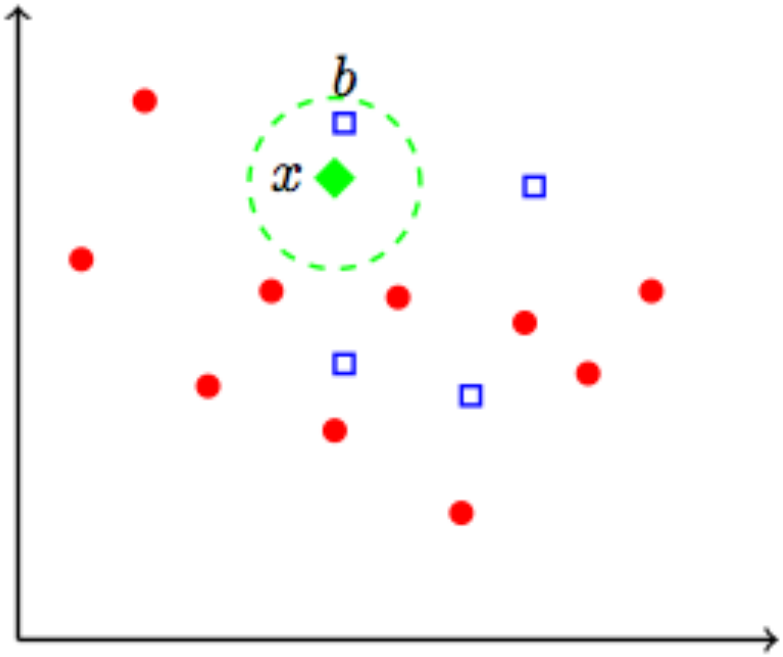
k-NN Example

- Red circles and blue squares are training data
- Note that we assume **labeled** training data
- Suppose we want to classify green diamond…
- And we want to keep it as simple as possible



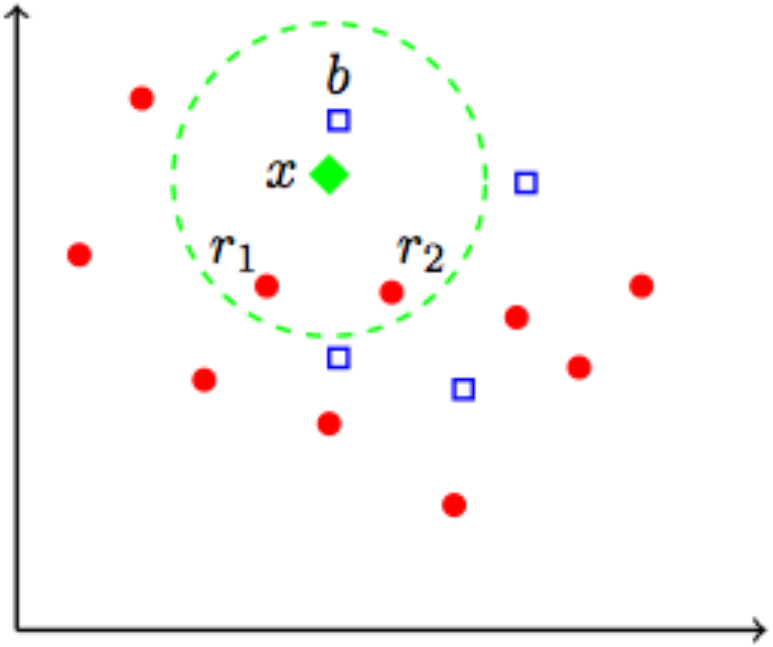
1-NN Example

- 1-NN
  - Or more simply, **nearest neighbor**
- Since blue square b is nearest…
- Classify green diamond x as "blue"



3-NN Example

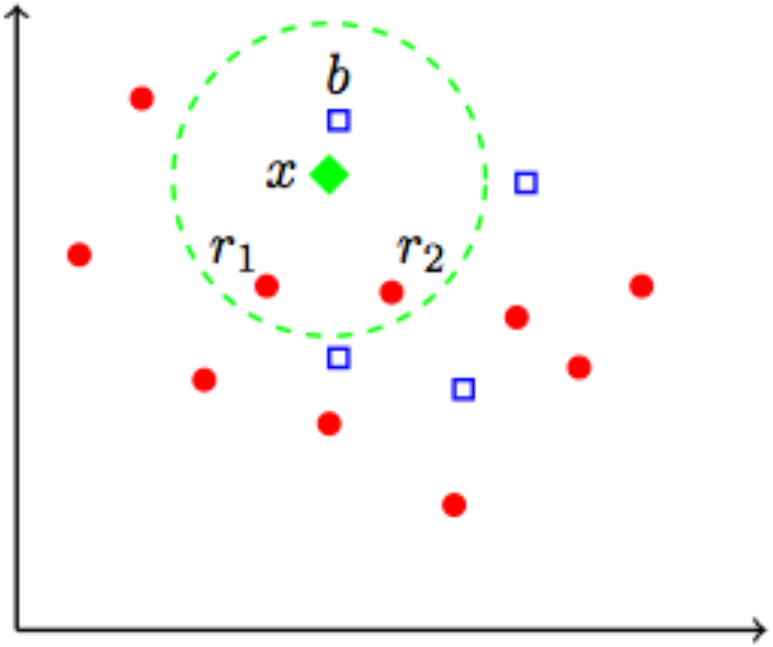
- 3-nearest neighbors
- Classify based on 3 nearest points
- 3 nearest to x are…
  - 2 red points, r_1, r_2, and 1 blue point, b
- Using 3-NN, we classify x as "red"



k-NN Variations

- We could also **weight** by distance
  - E.g., use 1/d(x,b) for each blue b nearest neighbor, and 1/d(x,r) for red r
  - Sum these by color, biggest sum wins…
- We might **weight** by class frequency
  - Suppose training set has B blue and R red, with R > B
  - Weight each red as 1, each blue as R/B (total # of red over total # of blue in the dataset)
- Might also consider a fixed radius

k-NN Weighted by Distance

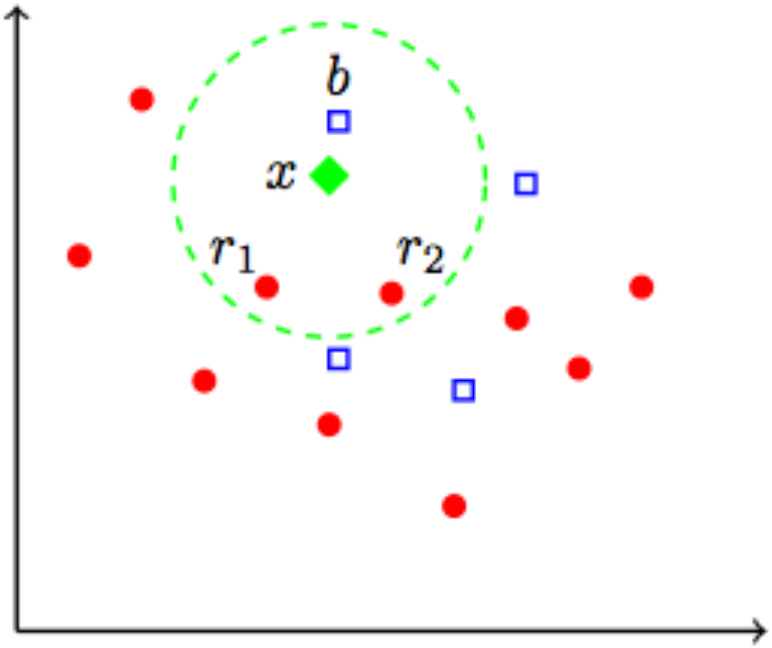


- Suppose use **1/d(x,y)**
- For this weight, 3-NN classifies x as "blue"…
  - *Assuming*

$1/d(x,b) > 1/d(x,r_1) + 1/d(x,r_2)$

k-NN Weighted by Frequency

- Suppose we weight by frequency
- Then each red is 1
- And each blue is 2.5 (10 reds / 4 blues)
- In this case, 3-NN classifies x as blue
  - Blue "score" is 2.5
  - Red "score" is 2.0



k-NN Advantages

- k-NN is a "lazy learning" algorithm
  - **No training** (none, nada, zippo) required
  - All computation deferred to scoring phase
- In limit, k-NN tends to (near) optimal…
  - …as size of training set grows
- Works for **multi-classification**
  - I.e., not restricted to binary classification

k-NN Disadvantages

- Scoring not entirely straightforward
  - In naïve approach, distances to all points needed for each score computation
  - Can use fast neighbor search algorithms (e.g., Knuth's "post office problem")…
  - …but then lose some of the simplicity
- Very sensitive to local structure
  - Random variations in local structure of training set can have undesirable impact

Bottom Line

- k-NN is as simple as it gets
  - And simple is good, provided that it works
- Training is non-existent
- Scoring is somewhat more involved
  - But still conceptually simple
- Lots of variations on the theme
- Can be combined with other techniques
  - E.g., k-NN used in scoring phase of PCA

Random Forest

Random Forest

- Do you remember bagging?

$$F\big(\mathcal{S}(x;V_1,\Lambda),\mathcal{S}(x;V_2,\Lambda),\ldots,\mathcal{S}(x;V_\ell,\Lambda)\big)$$

  - Bagging is used when generating a random forest, where each individual scoring function is based on a decision tree structure

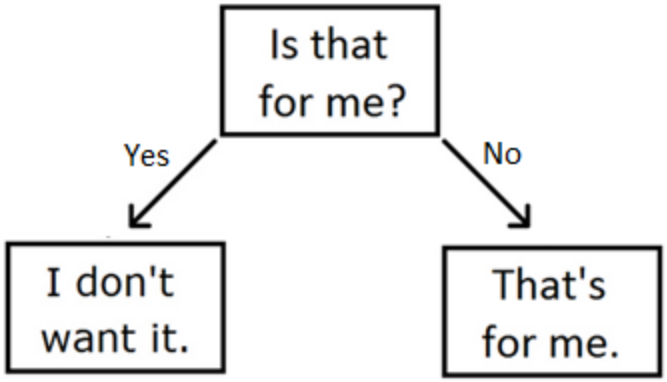
Random Forest

- **Random forest** (RF) is generalization of a **decision tree**
- Decision tree is really, really simple
  - Very intuitive and can be useful
- So, why do we need to generalize?
- **Decision trees** tend to overfit data
- **Random forest** reduces overfitting
  - But lose some of the intuitive simplicity

Decision Trees

- A **decision tree** is just what it says…
  - Tree that is used to make decisions
  - Kind of like a flow chart
- Each node is a test condition
- Each branch is outcome of test represented by corresponding node
- Leaf nodes contain the final decision
  - Simple, simple, simple, …

Cat's Decision Tree



- This one-level decision tree is also called a "decision stump"

Decision Trees

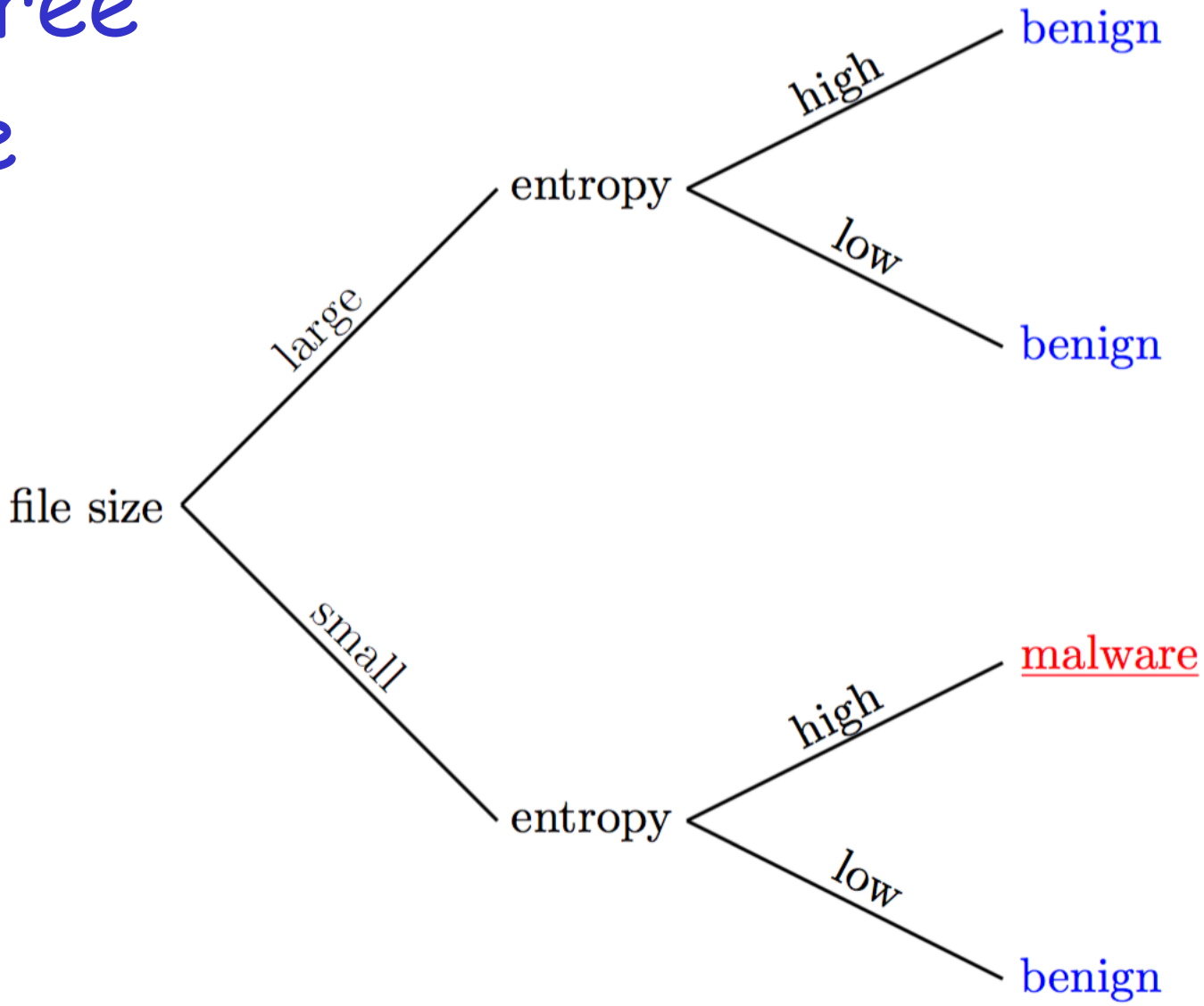
- Advantages?
  - Can be constructed with little/no data and can be tested if/when data available
  - Easy to understand, easy to use, easy to combine with other learning methods, …
- Disadvantages?
  - Constructing optimal tree is NP complete
  - **Overfitting**, complex trees, how to prune?
  - Some concepts not easy to fit to trees

Decision Tree Example

- Suppose that we have labeled training data for malware and benign samples
  - Features: **file size** and **entropy**
    - We observe that malware tends to be smaller in size with higher entropy
    - As compared to benign samples
- Easy to construct decision tree(s)
  - Next slides…

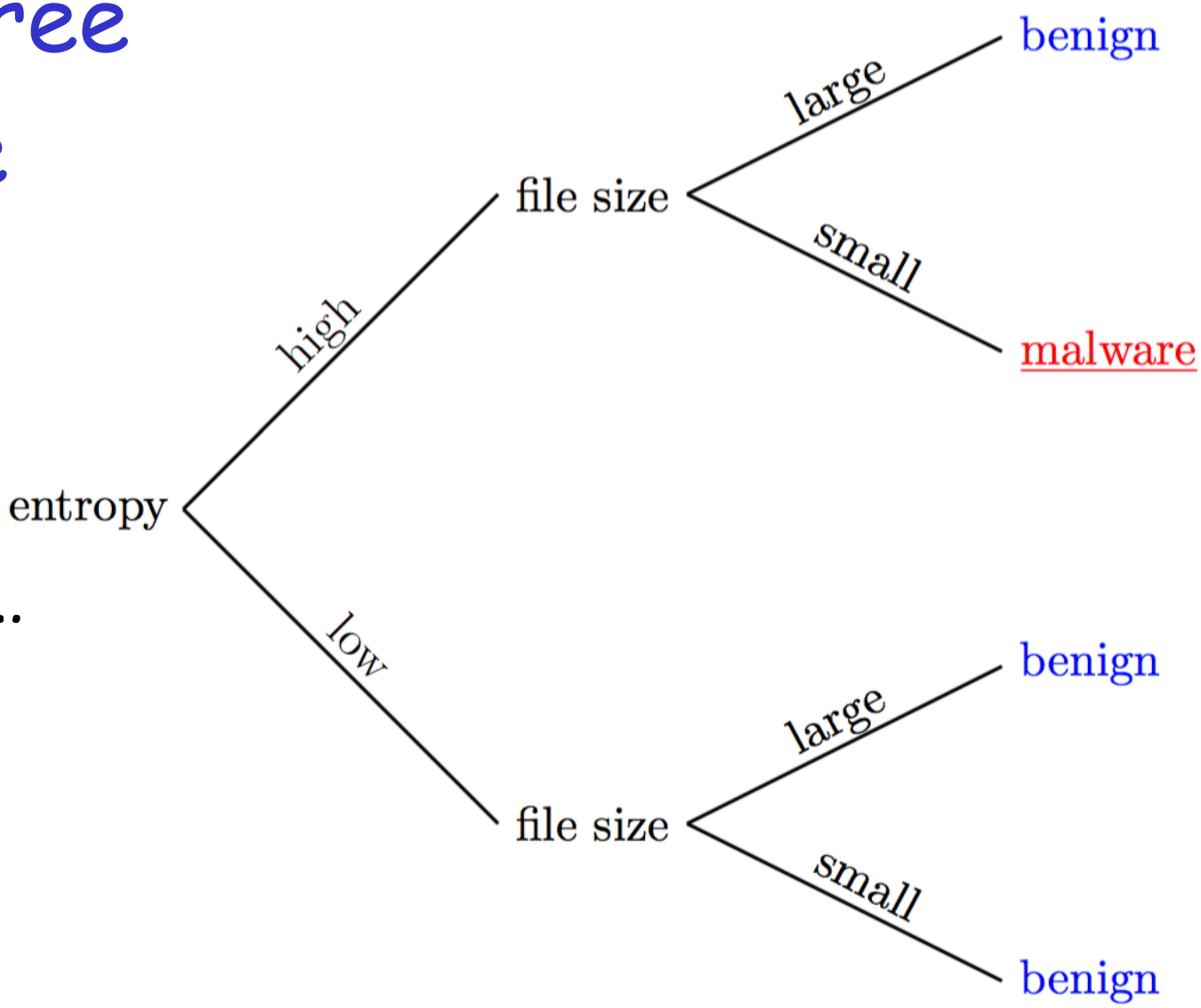
Decision Tree Example

- Large vs small, high vs low thresholdsbased on data



Decision Tree Example

- Other order works too…
- Which is better?



Decision Tree Decisions

- Generating optimal decision tree can be hard, so what to do?
- Approximate (how?)
- We'll use a simple greedy approach
  - Choose feature that provides most **information gain** and split data
    - From remaining features, select the one that provides **most information gain**
    - Continue until gain is below a threshold

Decision Tree

- Why greedy?
  - Want to use best classifiers first, so smaller trees have most the useful info
  - Want most info gain closer to root (good when we want to prune the tree)
  - Fast and efficient to construct, since no backtracking or other complex algorithm
  - Make use of most relevant information in training data

Information Gain

- "Gain" can be measured using **entropy**
  - Recall, entropy measures uncertainty
- Information gain for feature A ?
  - Entropy reduction if data is split on A

- We want to maximize information gain
  - Compute gain for each remaining feature
  - Split on feature with biggest info gain
  - Repeat until gain is below some threshold

Information Gain

- Let $P(x_i)$ be probability of outcome x_i
- Then **entropy** of $X = (x_1, x_2, \ldots, x_n)$ is:

$$H(X) = \sum_{i=1}^{n} -P(x_i) \log_2 P(x_i)$$

  - "How many yes/no questions we need to ask to guess the outcome?"
    - "Binary" question
  - The '-' gives a positive output from values in [0-1]

Information Gain

$$H(X) = \sum_{i=1}^{n} -P(x_i) \log_2 P(x_i)$$

- "How many yes/no questions we need to ask to guess the outcome?"

- If we have: "AAAAAAAA" How many questions?
  - 0
- If we have: "AABBCCDD" How many questions?
  - 2
- If we have: "ABCCDDDD" How many questions?
  - $\frac{1}{2} * 1 + \frac{1}{4} * 2 + \frac{2}{8} * 3 = 1.75$

Number of questions times the probability

Information Gain

$$H(X) = \sum_{i=1}^{n} -P(x_i) \log_2 P(x_i)$$

- "How many yes/no questions we need to ask to guess the outcome?"

- If we have: "ABCCDDDD" How many questions?
  - $\frac{1}{2} * 1 + \frac{1}{4} * 2 + \frac{2}{8} * 3 = 1.75$
    - First question in this case would be "Is it D?", because it could allow us to be done with just 1 question 50% of the time!

Information Gain

- Let $P(x_i)$ be probability of outcome x_i
- Then **entropy** of $X = (x_1, x_2, \ldots, x_n)$ is:

$$H(X) = \sum_{i=1}^{n} -P(x_i) \log_2 P(x_i)$$

- Suppose that we have 10 malware and 10 benign samples
- Measure file size, entropy, and number of distinct opcodes for each

Training Data

- Notation
  - S(X) is size
  - H(X) entropy
  - D(X) opcodes
- How to set thresholds?
  - Midway between averages…

i	Malware			Benign		
	$S(X_i)$	$H(X_i)$	$D(X_i)$	$S(Y_i)$	$H(Y_i)$	$D(Y_i)$
1	120	7	32	120	4	22
2	120	7	28	130	5	23
3	100	6	34	140	5	26
4	130	5	33	100	5	21
5	100	6	35	110	6	20
6	100	5	27	140	7	20
7	100	6	32	140	3	28
8	120	6	33	100	4	21
9	100	8	32	100	4	24
10	110	6	34	120	7	25
Average	110	6	32	120	5	23

- Notation
  - S(X) is size
  - H(X) entropy
  - D(X) opcodes
- For example, for Size ->

i	Malware			Benign		
	$S(X_i)$	$H(X_i)$	$D(X_i)$	$S(Y_i)$	$H(Y_i)$	$D(Y_i)$
1	120	7	32	120	4	22
2	120	7	28	130	5	23
3	100	6	34	140	5	26
4	130	5	33	100	5	21
5	100	6	35	110	6	20
6	100	5	27	140	7	20
7	100	6	32	140	3	28
8	120	6	33	100	4	21
9	100	8	32	100	4	24
10	110	6	34	120	7	25
Average	110	6	32	120	5	23

Average is 110 (MW) and 120 (BN)

- We can set the threshold to **115**

  Note that 115 is in between 110 and 120

Malware < 115

- Notation
  - S(X) is size
  - H(X) entropy
  - D(X) opcodes
- Now we have two sets
  - Benign and malware samples based on size

i	Malware			Benign		
	$S(X_i)$	$H(X_i)$	$D(X_i)$	$S(Y_i)$	$H(Y_i)$	$D(Y_i)$
1	120	7	32	120	4	22
2	120	7	28	130	5	23
3	100	6	34	140	5	26
4	130	5	33	100	5	21
5	100	6	35	110	6	20
6	100	5	27	140	7	20
7	100	6	32	140	3	28
8	120	6	33	100	4	21
9	100	8	32	100	4	24
10	110	6	34	120	7	25
Average	110	6	32	120	5	23

$T_m = \{X_3, X_5, X_6, X_7, X_9, X_{10}, Y_4, Y_5, Y_8, Y_9\}$
classified as malware

$T_b = \{X_1, X_2, X_4, X_8, Y_1, Y_2, Y_3, Y_6, Y_7, Y_{10}\}$
classified as benign

- Formula:

$$H(X) = \sum_{i=1}^{n} -P(x_i)\log_2 P(x_i)$$

i	Malware			Benign		
	$S(X_i)$	$H(X_i)$	$D(X_i)$	$S(Y_i)$	$H(Y_i)$	$D(Y_i)$
1	120	7	32	120	4	22
2	120	7	28	130	5	23
3	100	6	34	140	5	26
4	130	5	33	100	5	21
5	100	6	35	110	6	20
6	100	5	27	140	7	20
7	100	6	32	140	3	28
8	120	6	33	100	4	21
9	100	8	32	100	4	24
10	110	6	34	120	7	25
Average	110	6	32	120	5	23

$T_m = \{X_3, X_5, X_6, X_7, X_9, X_{10}, Y_4, Y_5, Y_8, Y_9\}$ classified as malware

$T_b = \{X_1, X_2, X_4, X_8, Y_1, Y_2, Y_3, Y_6, Y_7, Y_{10}\}$ classified as benign

➢ C $H(T_m) = H(T_b) = -\frac{3}{5} \cdot \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) = 0.9710$

Information Gain

- Entropy computed as

$$H(T_m) = H(T_b) = -\frac{3}{5} \cdot \log_2\left(\frac{3}{5}\right) - \frac{2}{5} \cdot \log_2\left(\frac{2}{5}\right) = 0.9710$$

- **Information gain** ?
  - Entropy of parent node minus average weighted entropy of child nodes
- For "size" feature, **information gain** is
  - G_S = 1.0 – 0.9710 = 0.0290

Information Gain

- Similarly, we compute
  - $G_S = 0.0290$, $G_H = 0.1952$, $G_D = 0.5310$
- Conclusion?
  1. Want to make first split based on number of distinct opcodes, D(X)
  2. Then need to recalculate information gain for remaining features
     - To decide which to select for 2nd level

Bagging

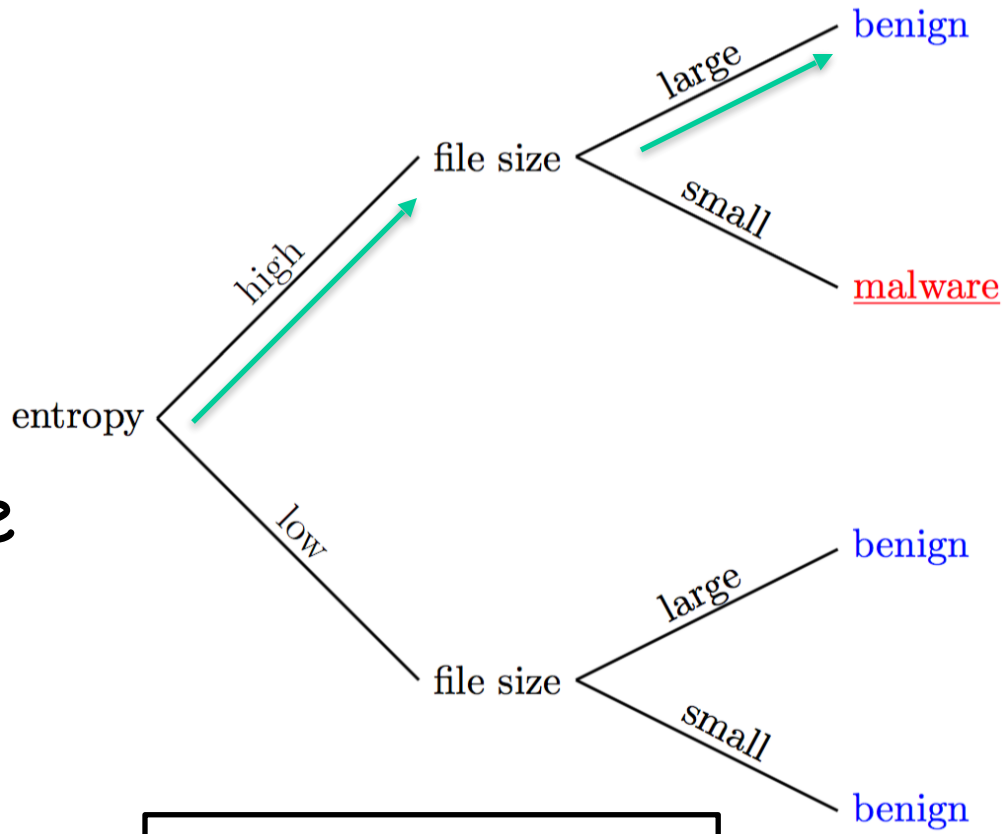
- Decision trees good, maybe too good…
  - Tend to overfit data
  - Overfitting is bad in ML (why?)
- What to do?
- **Bagging** (bootstrap aggregation)
  - Multiple decision trees on subsets of data
  - Then combine results (e.g. majority vote)
  - Easy way to reduce overfitting

Bagging Example

- Suppose we have sample U to classify
  - We measure S(U) = 116 and H(U) = 7
- Note that based on training data
  - 5.5 is threshold for entropy H(X)
  - 115 is threshold for size S(X)
- Suppose we classify using tree that splits first on entropy, then size
  - Ignoring opcode feature in this example

Bagging Example

- Then U is classified as benign
- But this is suspect…
  - Why?
- Suppose instead, use bagging
  - As on next slide…



H(U) = 7 > 5.5
S(U) = 116 > 115

Bagging

- Suppose we select subsets

$$A = \{1, 2, 3, 5, 10\},\ B = \{3, 5, 7, 9, 10\},\ \text{and}\ C = \{1, 2, 6, 8, 10\}.$$

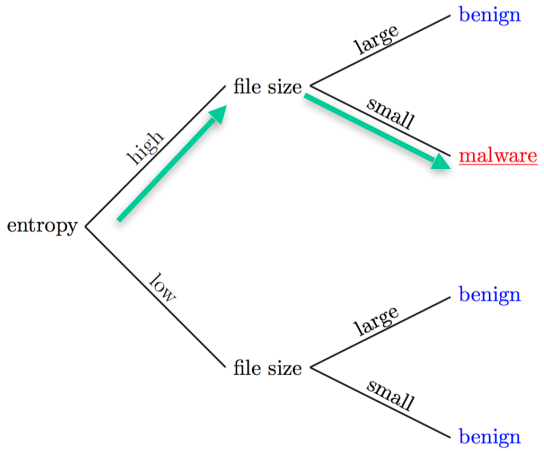
- Then subset A is:

i	Malware		Benign	
	$S(X_i)$	$H(X_i)$	$S(Y_i)$	$H(Y_i)$
1	120	7	120	4
2	120	7	130	5
3	100	6	140	5
5	100	6	110	6
10	110	6	120	7
Average	110	6.4	124	5.4

$$A = \{1, 2, 3, 5, 10\},\ B = \{3, 5, 7, 9, 10\},\ \text{and}\ C = \{1, 2, 6, 8, 10\}.$$

- For subset A …
- Threshold for S(X) is 117 and H(X) is 5.9
- And U classified as malware!!!

i	Malware		Benign	
	$S(X_i)$	$H(X_i)$	$S(Y_i)$	$H(Y_i)$
1	120	7	120	4
2	120	7	130	5
3	100	6	140	5
5	100	6	110	6
10	110	6	120	7
Average	110	6.4	124	5.4



H(U) = 7 > 5.9
S(U) = **116 < 117**

Bagging

- Easy to show that U is classified as…
  - Malware based on subset A
  - Benign based on subset B
  - Malware based on subset C
- So, by majority vote, **U is malware**
- Recall, U was benign based on all data
  - But that classification looked suspect
- Bagging better generalizes the data
  - At least in this case…

Random Forest

- **Random forest** uses bagging in 2 ways
  - Bagging of data (as on previous slide) …
  - … and bagging of features

How to bag features?

- Select subset of features and ordering
  - RF training algorithm use heuristic to do smart bagging

Random Forest and k-NN

- Interesting connection between **RF** and **k-NN** algorithms
- As usual, let (X_i, z_i), $i=1,2,\ldots,n$ be **training set**, and each z_i is -1 or $+1$
- Then define **weight function**

$$W_k(X_i, X) = \begin{cases} 1 & \text{if } X_i \text{ is one of the } k \text{ nearest neighbors to } X \\ 0 & \text{otherwise} \end{cases}$$

- And define

$$\text{score}_k(X) = \sum_{i=1}^{n} z_i \, W_k(X_i, X)$$

Decision Tree and k-NN

- For a given **decision tree**, define

$$W_t(X_i, X) = \begin{cases} 1 & \text{if } X_i \text{ is on the same leaf node as } X \\ 0 & \text{otherwise} \end{cases}$$

- And

$$\text{score}_t(X) = \sum_{i=1}^{n} z_i \, W_t(X_i, X)$$

- So what?

Decision Tree and k-NN

- Then **k-NN** is equivalent to…
  - Classify X as type +1 if $score_k(X) > 0$
  - And type −1 otherwise
- And **decision tree** (DT) is same as…
  - Classify X as type +1 if $score_t(X) > 0$
  - And type −1 otherwise

- **DT** and **k-NN** are neighborhood-based
  - But different neighborhood structure

Random Forest and k-NN

- **Random forest** is collection of DTs
  - So, same approach as on previous slides applies to RF
- Implies **RF** also neighborhood-based
  - Like decision tree…
  - …but neighborhood structure is significantly more complex
- Somewhat surprising connection…

Bottom Line

- Decision tree is a simple concept
- Bagging data generalizes decision tree
  - Less prone to overfit
- Random forest further generalizes concept of bagging
  - Bagging for both data and features
- Often, RF gives very good results
  - Perhaps more surprisingly, so does k-NN