Decision Trees

Professor Ameet Talwalkar

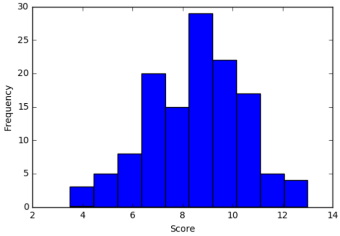
Outline

1. Administration
2. Review of last lecture
3. Decision tree

Homeworks

- HW1 due now
- HW2 will be available by next Monday (and possibly earlier)

Math Quiz



- Scores were out of 13 points
- Mean: 8.53; median: 8.75; standard deviation: 2.0
- Your grade is available on CCLE
- Will NOT count toward final grade, but good indication of material
  - Score less than 6: I have already contacted you
- Roughly 25 registered students have not taken the quiz!

Registration / PTEs

- I have increased class size to add all students who were on waitlist
  - Waitlist is now empty (and locked)
- I plan to give out PTEs in the next week
- If you're not registered, please continue to remain patient
  - I am confident that all qualified students will be able to enroll
  - Request PTE here: `https://goo.gl/forms/cpS1XcfWuVTileKI3`
  - Priority to students who take quiz by Friday (1/20) (check CCLE!)

Preview / Review

- I am aware that the lecture presentation can be fast at times
- Providing slides in advance of lecture usually not possible
- However, I cover material twice
  - e.g., today we'll first review nearest neighbor material before talking about decision trees
- This gives you two opportunities to be exposed to the material and ask questions

Outline

1. Administration

2. Review of last lecture
   - General setup for classification
   - Nearest neighbor classifier
   - Understanding learning algorithm
   - Practical Considerations

3. Decision tree

Multi-class classification

Classify data into one of the multiple categories

- Input (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- Learning goal: $y = f(\boldsymbol{x})$

Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

Multi-class classification

Classify data into one of the multiple categories

- Input (feature vectors): $\boldsymbol{x} \in \mathbb{R}^{\mathsf{D}}$
- Output (label): $y \in [\mathsf{C}] = \{1, 2, \cdots, \mathsf{C}\}$
- Learning goal: $y = f(\boldsymbol{x})$

Special case: binary classification

- Number of classes: $\mathsf{C} = 2$
- Labels: $\{0, 1\}$ or $\{-1, +1\}$

Example: Iris dataset

- 3 classes, corresponding to three types of Irises
- $D = 4$ corresponding to the length and width of the sepals and petals

More terminology

Training data)

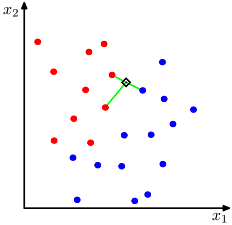
- N samples/instances: $\mathcal{D}^{\text{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- They are used for learning $f(\cdot)$

Test data

- M samples/instances: $\mathcal{D}^{\text{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\text{TRAIN}}$

Training data and test data should *not* overlap: $\mathcal{D}^{\text{TRAIN}} \cap \mathcal{D}^{\text{TEST}} = \emptyset$

Algorithm



(a)

Nearest neighbor

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$,

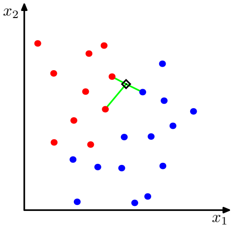
$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

Classification rule

$$y = f(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

Extension to KNN classification?

Algorithm



(a)

Nearest neighbor

$$\boldsymbol{x}(1) = \boldsymbol{x}_{\mathsf{nn}(\boldsymbol{x})}$$

where $\mathsf{nn}(\boldsymbol{x}) \in [\mathsf{N}] = \{1, 2, \cdots, \mathsf{N}\}$,

$$\mathsf{nn}(\boldsymbol{x}) = \arg\min_{n \in [\mathsf{N}]} \|\boldsymbol{x} - \boldsymbol{x}_n\|_2^2$$

Classification rule

$$y = f(\boldsymbol{x}) = y_{\mathsf{nn}(\boldsymbol{x})}$$

Extension to KNN classification?

- Every neighbor gets a vote; return the majority vote
- Randomly break ties

How good is NNC?

We answer this question in 3 steps:

1. We define a performance metric for a classifier/algorithm

How good is NNC?

We answer this question in 3 steps:

1. We define a performance metric for a classifier/algorithm
   - Expected Risk via 0/1 Loss

How good is NNC?

We answer this question in 3 steps:

1. We define a performance metric for a classifier/algorithm
   - Expected Risk via 0/1 Loss
2. We then propose an ideal classifier

How good is NNC?

We answer this question in 3 steps:

1. We define a performance metric for a classifier/algorithm
   - Expected Risk via 0/1 Loss
2. We then propose an ideal classifier
   - Bayes Optimal Classifier

How good is NNC?

We answer this question in 3 steps:

1. We define a performance metric for a classifier/algorithm
   - Expected Risk via 0/1 Loss
2. We then propose an ideal classifier
   - Bayes Optimal Classifier
3. We then compare NNC to the Bayes Optimal Classifier
   - Cover-Hart Inequality

Performance Metric

- Assume data $(\boldsymbol{x}, y)$ drawn from *unknown*, joint distribution $p(\boldsymbol{x}, y)$
- *0/1 loss function* measures mistake on a single data point

$$L(f(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } f(x) = y \\ 1 & \text{if } f(x) \neq y \end{cases}$$

Performance Metric

- Assume data $(\boldsymbol{x}, y)$ drawn from *unknown*, joint distribution $p(\boldsymbol{x}, y)$
- *0/1 loss function* measures mistake on a single data point

$$L(f(\boldsymbol{x}), y) = \begin{cases} 0 & \text{if } f(x) = y \\ 1 & \text{if } f(x) \neq y \end{cases}$$

- Empirical risk (on test set)

$$R_{\mathcal{D}}(f) = \frac{1}{\mathsf{M}} \sum_{m} L(f(\boldsymbol{x}_m), y_m)$$

- Expected risk

$$R(f) = \mathbb{E}_{(\boldsymbol{x},y)\sim p(\boldsymbol{x},y)} L(f(\boldsymbol{x}), y)$$

Bayes binary classifier

It 'cheats' by using the posterior probability $\eta(\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$

Bayes binary classifier

It 'cheats' by using the posterior probability $\eta(\boldsymbol{x}) = p(y = 1|\boldsymbol{x})$

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \eta(\boldsymbol{x}) \geq 1/2 \\ 0 & \text{if } \eta(\boldsymbol{x}) < 1/2 \end{cases} \quad \text{equivalently} \quad f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } p(y = 1|\boldsymbol{x}) \geq p(y = 0|\boldsymbol{x}) \\ 0 & \text{if } p(y = 1|\boldsymbol{x}) < p(y = 0|\boldsymbol{x}) \end{cases}$$

Bayes binary classifier

It 'cheats' by using the posterior probability $\eta(\boldsymbol{x}) = p(y=1|\boldsymbol{x})$

$$f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \eta(\boldsymbol{x}) \geq 1/2 \\ 0 & \text{if } \eta(\boldsymbol{x}) < 1/2 \end{cases} \quad \text{equivalently } f^*(\boldsymbol{x}) = \begin{cases} 1 & \text{if } p(y=1|\boldsymbol{x}) \geq p(y=0|\boldsymbol{x}) \\ 0 & \text{if } p(y=1|\boldsymbol{x}) < p(y=0|\boldsymbol{x}) \end{cases}$$

Unsurprisingly, it is optimal (we proved this)

Theorem

For any labeling function $f(\cdot)$, $R(f^*) \leq R(f)$.

Comparing NNC to Bayes optimal classifier

How well does NNC do asymptotically?

Theorem (Cover-Hart inequality)

For the NNC rule f^{NNC} for binary classification, we have,

$$R(f^*) \leq R(f^{\text{NNC}}) \leq 2R(f^*)$$

What does this tell us?

- Shows that as $n \to \infty$, NNC's expected risk is at worst twice that of the Bayes optimal classifier
- Provides theoretical justification, as NNC is nearly optimal asymptotically

Hyperparameters in NNC

Three practical issues related to NNC

Hyperparameters in NNC

Three practical issues related to NNC

- Choosing K, i.e., the number of nearest neighbors (default is 1)
- Choosing the right distance measure (default is Euclidean distance)
- Choosing the scale of each feature since distances depend on units (default is to normalize to zero mean and unit variance)

Those are not specified by the algorithm itself — resolving them requires empirical studies and are task/dataset-specific.

Tuning by using a validation dataset

Training data

- N samples/instances: $\mathcal{D}^{\mathrm{TRAIN}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{N}}, y_{\mathsf{N}})\}$
- They are used for learning $f(\cdot)$

Test data

- M samples/instances: $\mathcal{D}^{\mathrm{TEST}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{M}}, y_{\mathsf{M}})\}$
- They are used for assessing how well $f(\cdot)$ will do in predicting an unseen $\boldsymbol{x} \notin \mathcal{D}^{\mathrm{TRAIN}}$

Validation data

- L samples/instances: $\mathcal{D}^{\mathrm{VAL}} = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_{\mathsf{L}}, y_{\mathsf{L}})\}$
- They are used to optimize hyperparameter(s).

Training data, validation and test data should *not* overlap!

Recipe

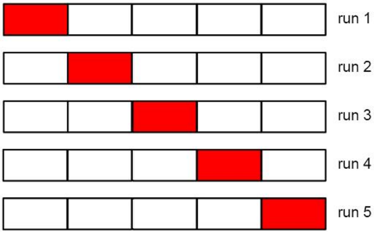
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - Train a model using $\mathcal{D}^{\text{TRAIN}}$
  - Evaluate the performance of the model on $\mathcal{D}^{\text{VAL}}$
- Choose the model with the best performance on $\mathcal{D}^{\text{VAL}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Cross-validation

What if we do not want to withhold an explicit validation set?

- We split the training data into S equal parts.
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that *on average*, the model performing the best

S $= 5$: 5-fold cross validation

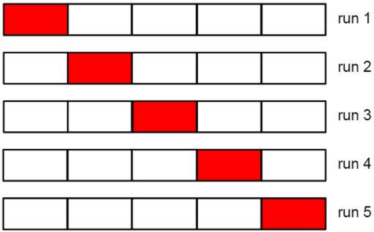


Cross-validation

What if we do not want to withhold an explicit validation set?

- We split the training data into S equal parts.
- We use each part *in turn* as a validation dataset and use the others as a training dataset.
- We choose the hyperparameter such that *on average*, the model performing the best

$S = 5$: 5-fold cross validation



Special case: when $S = N$, this will be leave-one-out.

Recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$

Recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - for every $s \in [1, \mathsf{S}]$
    - ★ Train a model using $\mathcal{D}_{\backslash s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
  - Average the S performance metrics

Recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - for every $s \in [1, \mathsf{S}]$
    - ★ Train a model using $\mathcal{D}_{\backslash s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
  - Average the S performance metrics
- Choose the hyperparameter corresponding to the best averaged performance

Recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - for every $s \in [1, \mathsf{S}]$
    - ★ Train a model using $\mathcal{D}_{\setminus s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
  - Average the S performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$

Recipe

- Split the training data into S equal parts. Denote each part as $\mathcal{D}_s^{\text{TRAIN}}$
- For each possible value of the hyperparameter (say $K = 1, 3, \cdots, 100$)
  - for every $s \in [1, \mathsf{S}]$
    - ★ Train a model using $\mathcal{D}_{\backslash s}^{\text{TRAIN}} = \mathcal{D}^{\text{TRAIN}} - \mathcal{D}_s^{\text{TRAIN}}$
    - ★ Evaluate the performance of the model on $\mathcal{D}_s^{\text{TRAIN}}$
  - Average the S performance metrics
- Choose the hyperparameter corresponding to the best averaged performance
- Use the best hyperparameter to train on a model using all $\mathcal{D}^{\text{TRAIN}}$
- Evaluate the model on $\mathcal{D}^{\text{TEST}}$

Things you need to know

NNC

- Advantages
  - Computationally, simple and easy to implement – just computing the distance
  - Theoretically, has good guarantees
- Disadvantages
  - Computationally intensive for large-scale problems: $O(\mathsf{N}D)$ for labeling a data point
  - We need to "carry" the training data around to perform classification (*nonparametric*).
  - Choosing the right distance measure, scaling, and K can be involved.

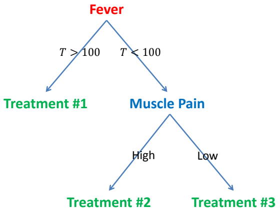
Crucial theoretical concepts loss function, expected risk, empirical risk, Bayes optimal

Crucial practical concepts hyperparameters, validation set, cross validation

Outline

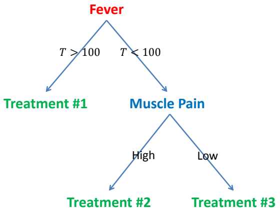
Many decisions are tree structures

Medical treatment

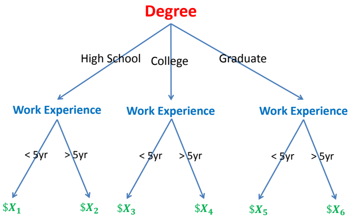


Many decisions are tree structures

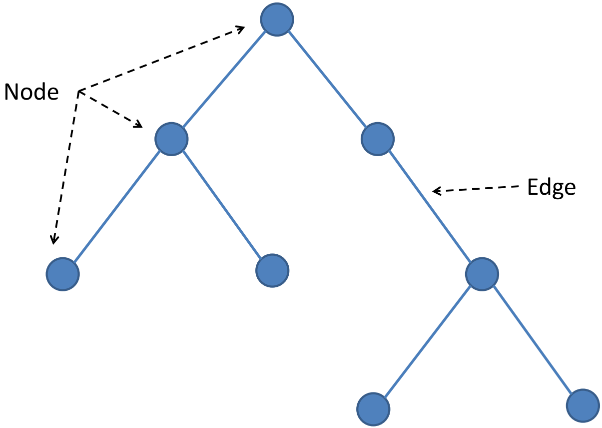
Medical treatment



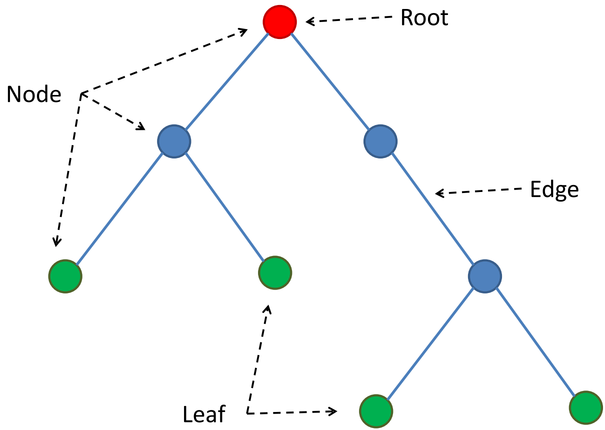
Salary in a company



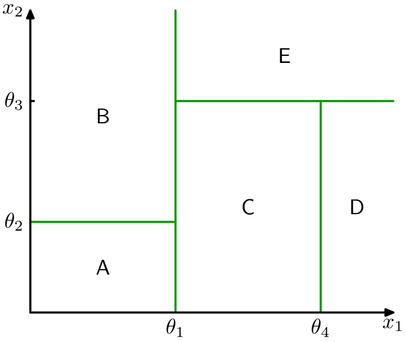
What is a Tree?

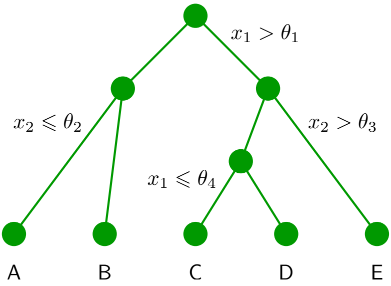


Special Names for Nodes in a Tree

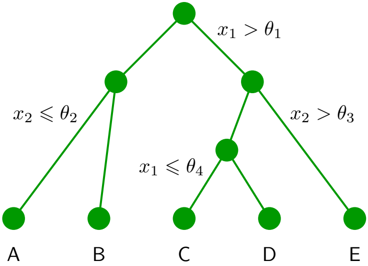


A tree partitions the feature space



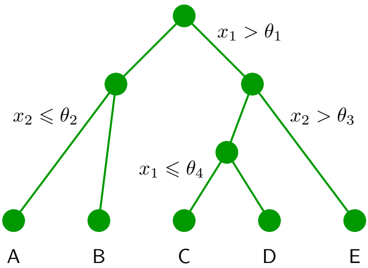


Learning a tree model



Three things to learn:

Learning a tree model



Three things to learn:

1. The structure of the tree.
2. The threshold values (θ_i).
3. The values for the leafs ($A, B, \ldots$).

A tree model for deciding where to eat

Choosing a restaurant

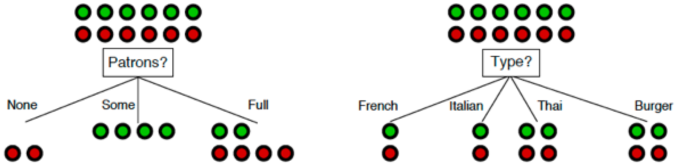
(Example from Russell & Norvig, AIMA)

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	*T*	*F*	*F*	*T*	*Some*	*$$$*	*F*	*T*	*French*	*0–10*	*T*
X_2	*T*	*F*	*F*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*30–60*	*F*
X_3	*F*	*T*	*F*	*F*	*Some*	*$*	*F*	*F*	*Burger*	*0–10*	*T*
X_4	*T*	*F*	*T*	*T*	*Full*	*$*	*F*	*F*	*Thai*	*10–30*	*T*
X_5	*T*	*F*	*T*	*F*	*Full*	*$$$*	*F*	*T*	*French*	*>60*	*F*
X_6	*F*	*T*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Italian*	*0–10*	*T*
X_7	*F*	*T*	*F*	*F*	*None*	*$*	*T*	*F*	*Burger*	*0–10*	*F*
X_8	*F*	*F*	*F*	*T*	*Some*	*$$*	*T*	*T*	*Thai*	*0–10*	*T*
X_9	*F*	*T*	*T*	*F*	*Full*	*$*	*T*	*F*	*Burger*	*>60*	*F*
X_{10}	*T*	*T*	*T*	*T*	*Full*	*$$$*	*F*	*T*	*Italian*	*10–30*	*F*
X_{11}	*F*	*F*	*F*	*F*	*None*	*$*	*F*	*F*	*Thai*	*0–10*	*F*
X_{12}	*T*	*T*	*T*	*T*	*Full*	*$*	*F*	*F*	*Burger*	*30–60*	*T*

Classification of examples is positive (T) or negative (F)

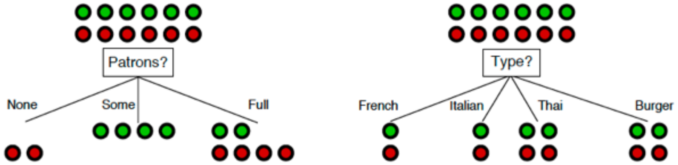
First decision: at the root of the tree

Which attribute to split?



First decision: at the root of the tree

Which attribute to split?



Patrons? is a better choice—gives **information** about the classification

Idea: use information gain to choose which attribute to split

How to measure information gain?

Idea:

Gaining information reduces uncertainty

the base can be 2 , though it is not essential (if the base is 2, the unit of the entropy is called "bit")

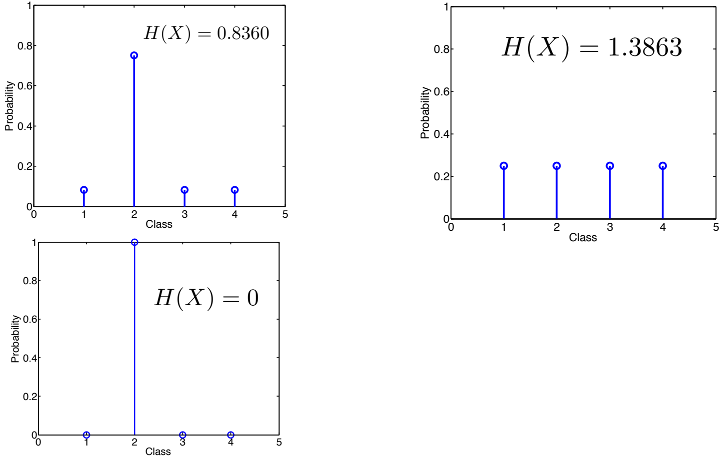
Use to entropy to measure uncertainty

If a random variable X has K different values, a_1, a_2, ...a_K, it is entropy is given by

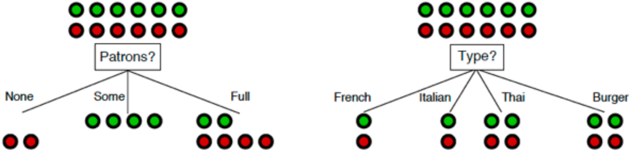
$$H[X] = -\sum_{k=1}^{K} P(X = a_k) \log P(X = a_k)$$

Examples of computing entropy

Entropy



Which attribute to split?



Patrons? is a better choice—gives **information** about the classification

Patron vs. Type?

By choosing Patron, we end up with a partition (3 branches) with smaller entropy, ie, smaller uncertainty (0.45 bit)

By choosing Type, we end up with uncertainty of 1 bit.

Thus, we choose Patron over Type.

Uncertainty if we go with "Patron"

For "None" branch

$$-\left(\frac{0}{0+2}\log\frac{0}{0+2}+\frac{2}{0+2}\log\frac{2}{0+2}\right)=0$$

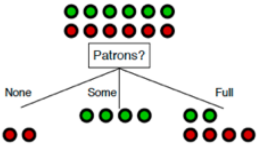
For "Some" branch

$$-\left(\frac{4}{4+0}\log\frac{4}{4+0}+\frac{4}{4+0}\log\frac{4}{4+0}\right)=0$$

For "Full" branch

$$-\left(\frac{2}{2+4}\log\frac{2}{2+4}+\frac{4}{2+4}\log\frac{4}{2+4}\right)\approx 0.9$$

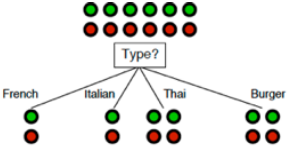
For choosing "Patrons"



weighted average of each branch: this quantity is called conditional entropy

$$\frac{2}{12}*0+\frac{4}{12}*0+\frac{6}{12}*0.9=0.45$$

Conditional entropy for Type



For "French" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Italian" branch

$$-\left(\frac{1}{1+1}\log\frac{1}{1+1}+\frac{1}{1+1}\log\frac{1}{1+1}\right)=1$$

For "Thai" and "Burger" branches

$$-\left(\frac{2}{2+2}\log\frac{2}{2+2}+\frac{2}{2+2}\log\frac{2}{2+2}\right)=1$$

For choosing "Type"

weighted average of each branch:

$$\frac{2}{12}*1+\frac{2}{12}*1+\frac{4}{12}*1+\frac{4}{12}*1=1$$

Conditional entropy

Definition. Given two random variables X and Y

$$H[Y|X] = \sum_k P(X = a_k) H[Y|X = a_k]$$

In our example

X: the attribute to be split

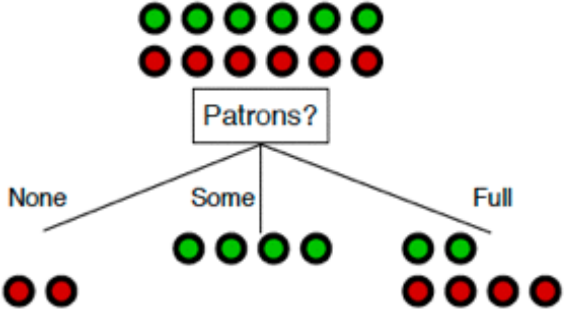
Y: Wait or not

When H[Y] is fixed, we need only to compare conditional entropy

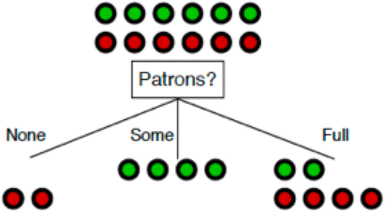
Relation to information gain

$$\text{GAIN} = H[Y] - H[Y|X]$$

What do we do next?



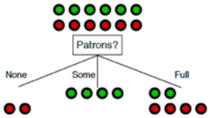
Do we split on “Non” or “Some”?



No, we do not

The decision is deterministic, as seen from the training data

next split?

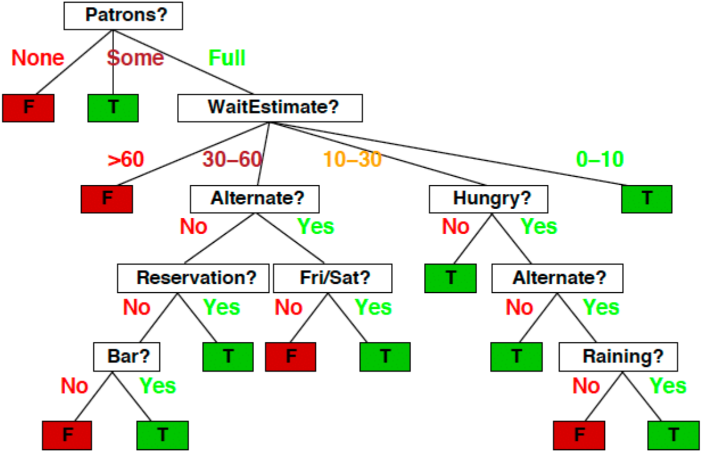


We will look only at the 6 instances with Patrons == Full

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
X_1	T	F	F	T	Some	$$$	F	T	French	0–10	T
X_2	T	F	F	T	Full	$	F	F	Thai	30–60	F
X_3	F	T	F	F	Some	$	F	F	Burger	0–10	T
X_4	T	F	T	T	Full	$	F	F	Thai	10–30	T
X_5	T	F	T	F	Full	$$$	F	T	French	>60	F
X_6	F	T	F	T	Some	$$	T	T	Italian	0–10	T
X_7	F	T	F	F	None	$	T	F	Burger	0–10	F
X_8	F	F	F	T	Some	$$	T	T	Thai	0–10	T
X_9	F	T	T	F	Full	$	T	F	Burger	>60	F
X_{10}	T	T	T	T	Full	$$$	F	T	Italian	10–30	F
X_{11}	F	F	F	F	None	$	F	F	Thai	0–10	F
X_{12}	T	T	T	T	Full	$	F	F	Burger	30–60	T

Classification of examples is positive (T) or negative (F)

Greedily we build the tree and get this



What is the optimal Tree Depth?

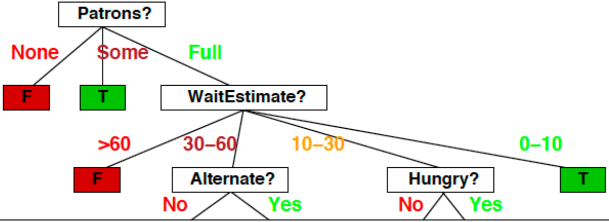
- We need to be careful to pick an appropriate tree depth

What is the optimal Tree Depth?

- We need to be careful to pick an appropriate tree depth
  - If the tree is too deep, we can overfit
  - If the tree is too shallow, we underfit
- Max depth is a hyperparameter that should be tuned by the data
- Alternative strategy is to create a very deep tree, and then to prune it (see Section 9.2.2 in ESL for details)

Control the size of the tree

We would prune to have a smaller one

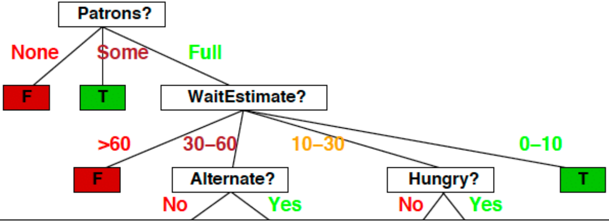


If we stop here, not all training sample would be classified correctly.

More importantly, how do we classify a new instance?

Control the size of the tree

We would prune to have a smaller one



If we stop here, not all training sample would be classified correctly.

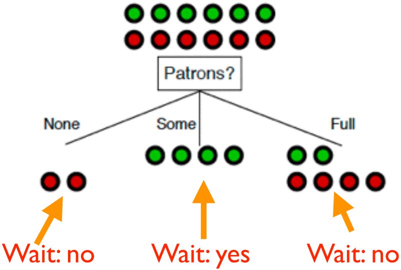
More importantly, how do we classify a new instance?

We label the leaves of this smaller tree with the majority of training samples' labels

Example

Example

We stop after the root (first node)



Computational Considerations

Numerical Features

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest
  - This takes $O(dn \log n)$ time

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest
  - This takes $O(dn \log n)$ time

Categorical Features

Computational Considerations

Numerical Features

- We could split on any feature with any threshold
- Can we do this efficiently?
  - Yes – for a given feature we only need to consider the n values in the training data!
  - If we sort each feature by these n values, we can quickly compute our impurity metric of interest
  - This takes $O(dn \log n)$ time

Categorical Features

- Assuming q distinct categories, there are $2^{q-1} - 1$ possible partitions
- Things simplify in the case of binary classification or regression,
  - suffices to consider only $q - 1$ possible splits (see Section 9.2.4 in ESL)

Summary of learning trees

Advantages of using trees

Summary of learning trees

Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Computationally efficient
- Handles both numerical and categorical data
- Parametric and thus compact: unlike NNC we don't need training data at test time
- Building block for various ensemble methods (more on this later)

Disadvantages

Summary of learning trees

Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Computationally efficient
- Handles both numerical and categorical data
- Parametric and thus compact: unlike NNC we don't need training data at test time
- Building block for various ensemble methods (more on this later)

Disadvantages

- Heuristic training techniques

Summary of learning trees

Advantages of using trees

- Can be interpreted by humans (as long as the tree is not too big)
- Computationally efficient
- Handles both numerical and categorical data
- Parametric and thus compact: unlike NNC we don't need training data at test time
- Building block for various ensemble methods (more on this later)

Disadvantages

- Heuristic training techniques
  - Finding partition of space that minimizes empirical error is NP-hard
  - We resort to greedy approaches with limited theoretical underpinnings