ECE5984 – Applications of Machine Learning
Lecture 10 – Applications of Decision Trees

Creed Jones, PhD

Course update

- HW3 is posted
  - Due Tuesday, March 1
- Project I has been posted
  - Due Tuesday, March 22
  - Don’t delay!
- Spring break in two weeks
  - I won’t have office hours during Spring break

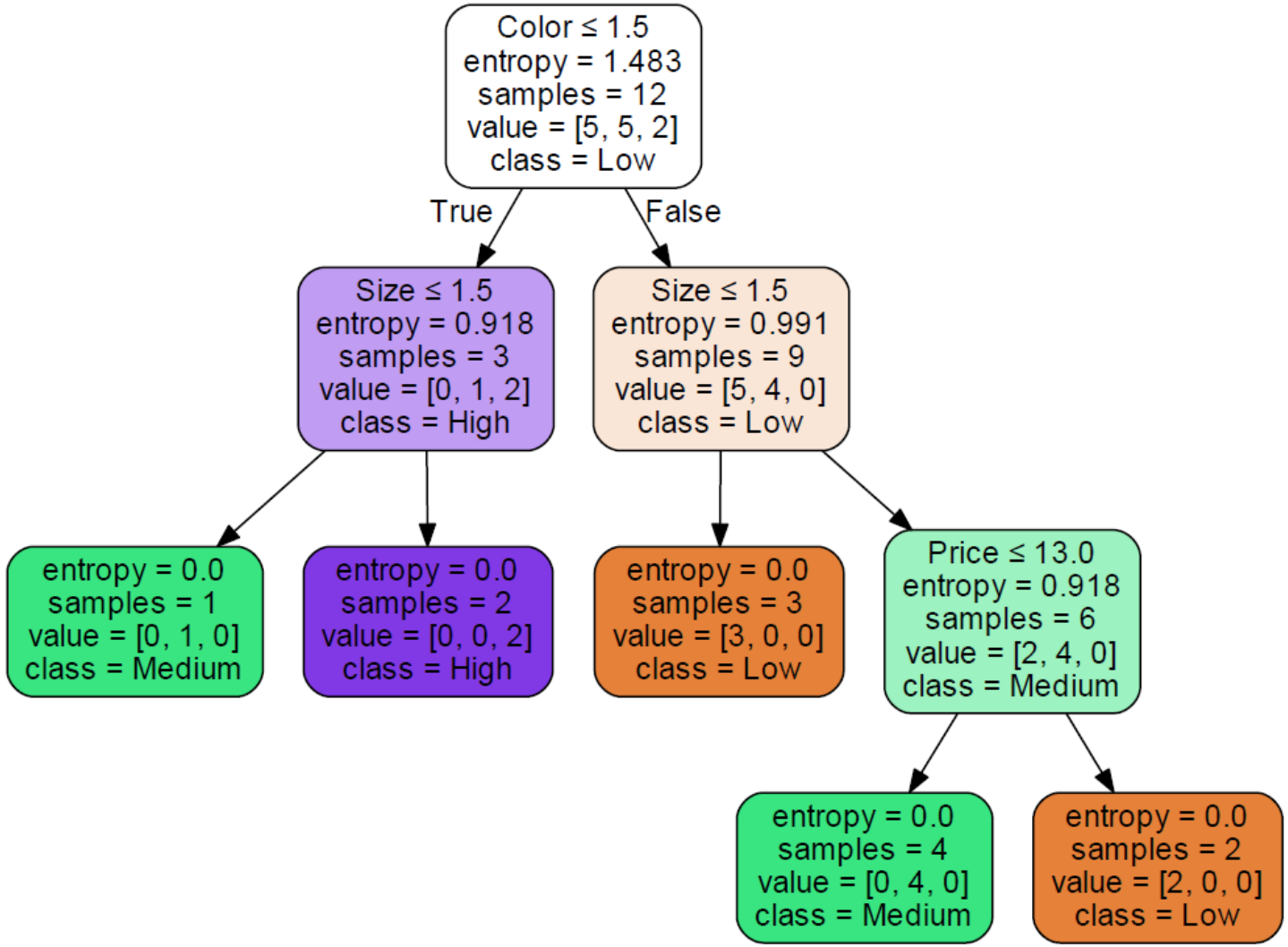
Today's Objectives

- Information Gain for Categorical Variables
- Other Impurity Measures
- Tree Pruning
- Decision Trees in Python
  - File I/O
  - Data Preparation
    - OneHot rendition
  - Selection Method
  - Tree Size and Shape
  - Performance
- Boosting
- Bagging
- Random Forest

A decision tree is used to perform classification by interrogating new samples using each decision and moving down through the tree

A terminal (leaf) node indicates a final classification

Where is the "intelligence" stored? In what form?



Information Gain for Categorical Variables

- The approach for calculating information gain for a categorical variable, specifically for calculating the remainder, is easily done for multiple values

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t, \mathcal{D}_{d=l})$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

Calculate the entropy for the target feature *Wait* in the dataset:

$$H(t,\mathcal{D}) = -\sum_{l \in levels(t)} \left[p(t=l)\log_2\left(p(t=l)\right)\right]$$

$$H(t,\mathcal{D}) = -\sum_{l \in levels(t)} \left[p(t=l)\log_2\left(p(t=l)\right)\right]$$

There are 12 instances in the dataset;
6 with *Wait*=F and 6 with *Wait*=T

$$H(t,\mathcal{D}) = -\sum_{l \in levels(t)} \left[p(t=l)\log_2\left(p(t=l)\right)\right] = -\left[p(F)\log_2\left(p(F)\right) + p(T)\log_2(pT)\right]$$

$$= -\left[\frac{6}{12}\log_2\left(\frac{6}{12}\right) + \frac{6}{12}\log_2\left(\frac{6}{12}\right)\right] = -\left[\log_2\left(\frac{1}{2}\right)\right] = -[-\log_2(2)] = 1\ bit$$

Calculate the remainder for the feature *Patrons* in the dataset:

$$rem(d,\mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t,\mathcal{D}_{d=l})$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

Patrons has three levels:

- None occurs twice; for both, *Wait*=F
- Some occurs four times; *Wait*=T for all four
- Full occurs six times; *Wait*=T for four of them

$$rem(PAT,\mathcal{D}) = \frac{|\mathcal{D}_{\text{PAT}=NONE}|}{|\mathcal{D}|} H(t,\mathcal{D}_{\text{PAT}=NONE}) + \frac{|\mathcal{D}_{PAT=SOME}|}{|\mathcal{D}|} H(t,\mathcal{D}_{PAT=SOME}) + \frac{|\mathcal{D}_{PAT=FULL}|}{|\mathcal{D}|} H(t,\mathcal{D}_{PAT=FULL})$$

$$= \frac{2}{12} \sum_{P=N,W\in\{T,F\}} -\left[p(W)\log_2\left(p(W)\right)\right] + \frac{4}{12} \sum_{P=S,W\in\{T,F\}} -\left[p(W)\log_2\left(p(W)\right)\right] + \frac{6}{12} \sum_{P=F,W\in\{T,F\}} -\left[p(W)\log_2\left(p(W)\right)\right]$$

$$= -\frac{2}{12}\left[\frac{0}{2}\log_2\left(\frac{0}{2}\right) + \frac{2}{2}\log_2\left(\frac{2}{2}\right)\right] - \frac{4}{12}\left[\frac{4}{4}\log_2\left(\frac{4}{4}\right) + \frac{0}{4}\log_2\left(\frac{0}{4}\right)\right] - \frac{6}{12}\left[\frac{4}{6}\log_2\left(\frac{4}{6}\right) + \frac{2}{6}\log_2\left(\frac{2}{6}\right)\right]$$

$$-\frac{1}{6}[\log_2(1)] - \frac{1}{3}[\log_2(1)] - \frac{1}{2}\left[\frac{2}{3}\log_2\left(\frac{2}{3}\right) + \frac{1}{3}\log_2\left(\frac{1}{3}\right)\right] = 0.459148\ bits$$

Calculate the information gain for the feature *Patrons* in the dataset:

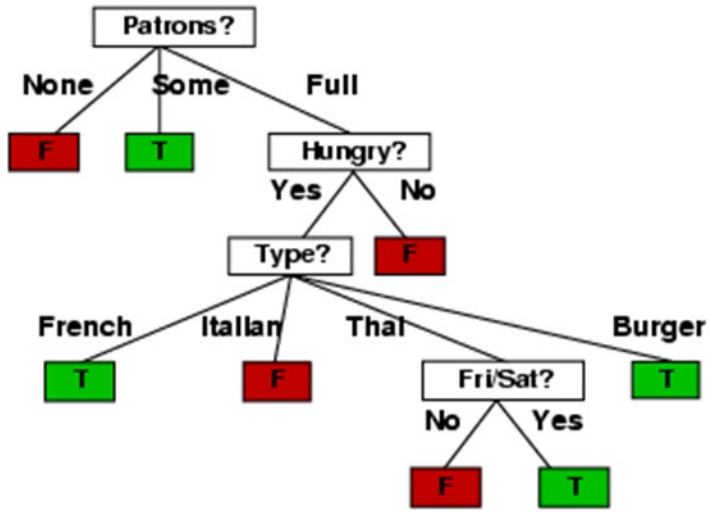
$$IG(Patrons, \mathcal{D}) = H(Wait, \mathcal{D}) - rem(Patrons, \mathcal{D})$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

Patrons has three levels:

- None occurs twice; for both, *Wait*=F
- Some occurs four times; *Wait*=T for all four
- Full occurs six times; *Wait*=T for four of them

$$H(Wait, \mathcal{D}) = 1\ bit$$
$$rem(Patrons, \mathcal{D}) = 0.459148\ bits$$
$$IG(Patrons, \mathcal{D}) = H(Wait, \mathcal{D}) - rem(Patrons, \mathcal{D}) = 0.540852\ bits$$



To check our understanding, calculate the information gains for both *Price* and *Hungry*

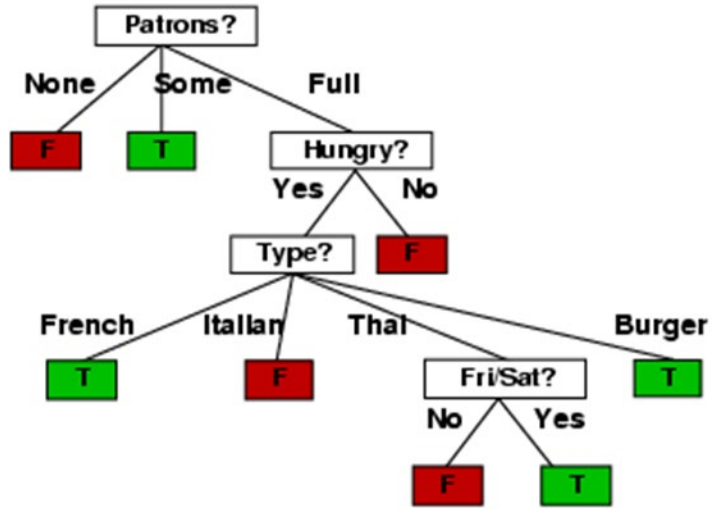
For the branch where Patron=Full

$$IG(f,\mathcal{D}') = H(Wait,\mathcal{D}') - rem(f,\mathcal{D}')$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

Patrons has three levels:

- None occurs twice; for both, *Wait*=F
- Some occurs four times; *Wait*=T for all four
- Full occurs six times; *Wait*=T for four of them



$$H(Wait,\mathcal{D}') = ?$$

$$rem(Price,\mathcal{D}') = ?$$
$$IG(Price,\mathcal{D}') = H(Wait,\mathcal{D}') - rem(Price,\mathcal{D}') = ?$$

$$rem(Hungry,\mathcal{D}') = ?$$
$$IG(Hungry,\mathcal{D}') = H(Wait,\mathcal{D}') - rem(Hungry,\mathcal{D}') = ?$$

Computing information gain involves the following 3 equations:

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} \left[p(t = l) \log_2 \left(p(t = l) \right) \right]$$

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t, \mathcal{D}_{d=l})$$

$$IG(d, \mathcal{D}) = H(t, \mathcal{D}) - rem(d, \mathcal{D})$$

$$H(t,\mathcal{D}') = -\left[p(F)\log_2\left(p(F)\right) + p(T)\log_2(pT)\right] = -\left[\frac{4}{6}\log_2\left(\frac{4}{6}\right) + \frac{2}{6}\log_2\left(\frac{2}{6}\right)\right] = 0.918296\ bit$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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

$$rem(Price,\mathcal{D}') = \frac{|\mathcal{D}'_{Price=\$}|}{|\mathcal{D}'|}H(t,\mathcal{D}'_{\text{Price=\$}}) + \frac{|\mathcal{D}'_{Price=\$\$}|}{|\mathcal{D}'|}H(t,\mathcal{D}'_{\text{Price=\$\$}}) + \frac{|\mathcal{D}'_{Price=\$\$\$}|}{|\mathcal{D}'|}H(t,\mathcal{D}'_{\text{Price=\$\$\$}})$$

$$= -\frac{4}{6}\left[\frac{2}{4}\log_2\left(\frac{2}{4}\right) + \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right] - \frac{0}{6}[0] - \frac{2}{6}\left[\frac{2}{2}\log_2\left(\frac{2}{2}\right) + \frac{0}{2}\log_2\left(\frac{0}{2}\right)\right] = -\frac{4}{6}\left[\log_2\left(\frac{1}{2}\right)\right] - 0 - \frac{2}{6}[\log_2(1)] = 0.6667\ bits$$

$$rem(Hungry,\mathcal{D}') = \frac{|\mathcal{D}'_{Hungry=T}|}{|\mathcal{D}'|}H(t,\mathcal{D}'_{Hungry=T}) + \frac{|\mathcal{D}'_{Hungry=F}|}{|\mathcal{D}'|}H(t,\mathcal{D}'_{Hungry=F})$$

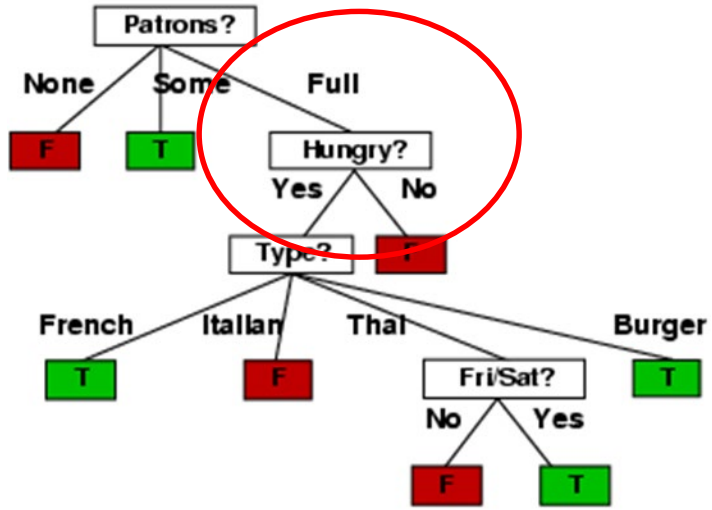
$$= -\frac{4}{6}\left[\frac{2}{4}\log_2\left(\frac{2}{4}\right) + \frac{2}{4}\log_2\left(\frac{2}{4}\right)\right] - \frac{2}{6}\left[\frac{2}{2}\log_2\left(\frac{2}{2}\right) + \frac{0}{2}\log_2\left(\frac{0}{2}\right)\right] = -\frac{4}{6}\left[\log_2\left(\frac{1}{2}\right)\right] - \frac{2}{6}[\log_2(1)] = 0.6667\ bits$$

$$IG(Price,\mathcal{D}) = H(Wait,\mathcal{D}) - rem(Price,\mathcal{D}) = 0.918296 - 0.6667 = 0.251596\ bits$$

$$IG(Hungry,\mathcal{D}) = H(Wait,\mathcal{D}) - rem(Hungry,\mathcal{D}) = 0.918296 - 0.6667 = 0.251596\ bits$$

$$IG(Price, \mathcal{D}) = H(Wait, \mathcal{D}) - rem(Price, \mathcal{D}) = 0.918296 - 0.6667 = 0.251596\ bits$$
$$IG(Hungry, \mathcal{D}) = H(Wait, \mathcal{D}) - rem(Hungry, \mathcal{D}) = 0.918296 - 0.6667 = 0.251596\ bits$$

Example	Attributes										Target
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*Wait*
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



Both *Price and Hungry* have the same information gain, so we could choose either
We choose *Hungry* as the next node in the tree because it's a binary feature and provides a simpler decision

Information Gain for Continuous Variables

- What if we want to use continuous variables in our decision tree?
- The number of values is generally large so how do we compute entropy and remainder?
  - Discretize by splitting the range of values
  - Perhaps into two groups; split at the midpoint
  - Perhaps into many groups, by choosing splits between adjacent values
  - Perhaps some other binning process
- The scikit-learn DecisionTreeClassifier uses either the "best" of a set of splits attempted, or a random split

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} H(t, \mathcal{D}_{d=l})$$

An alternative selection measure is the *information gain ratio*

- Entropy based information gain, preferences features with many values.
- One way of addressing this issue is to use **information gain ratio** which is computed by dividing the information gain of a feature by the amount of information used to determine the value of the feature:

$$GR(d, \mathcal{D}) = \frac{IG(d, \mathcal{D})}{-\sum_{l \in levels(d)} (P(d = l) \times log_2(P(d = l)))} \quad (1)$$

The Gini index is commonly used instead of entropy – many people prefer it, but results are usually similar

- Another commonly used measure of impurity is the **Gini index**:

$$Gini\,(t, \mathcal{D}) = 1 - \sum_{l \in levels(t)} P(t = l)^2 \qquad (2)$$

- The Gini index can be thought of as calculating how often you would misclassify an instance in the dataset if you classified it based on the distribution of classifications in the dataset.
- Information gain can be calculated using the Gini index by replacing the entropy measure with the Gini index.

Let's calculate the Gini index on a simple dataset (from the book); VEGETATION is the target

ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	false	steep	high	chaparral
2	true	moderate	low	riparian
3	true	steep	medium	riparian
4	false	steep	medium	chaparral
5	false	flat	high	conifer
6	true	steep	highest	conifer
7	true	steep	high	chaparral

$$
\begin{aligned}
&Gini\,(\text{VEGETATION}, \mathcal{D}) \\
&= 1 - \sum_{l \in \left\{ \begin{array}{l} \text{'chapparal'}, \\ \text{'riparian'}, \\ \text{'conifer'} \end{array} \right\}} P(\text{VEGETATION} = l)^2 \\
&= 1 - \left((3/7)^2 + \left(2/7\right)^2 + \left(2/7\right)^2 \right) \\
&= 0.6531
\end{aligned}
$$

The Gini index is used in the remainder and information gain calculations just as entropy was

The Gini index is considered a measure of the *impurity* of a portion of a dataset

The Gini index is always between 0 and 1, so it is sometimes easier to assess and compare

$$Gini(Vegetation, \mathcal{D}) = 0.6531$$

Split by Feature	Level	Part.	Instances	Partition Gini Index	Rem.	Info. Gain
STREAM	*'true'*	$\mathcal{D}_1$	$\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_6, \mathbf{d}_7$	0.625	0.5476	0.1054
	'false'	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_4, \mathbf{d}_5$	0.4444		
SLOPE	*'flat'*	$\mathcal{D}_3$	$\mathbf{d}_5$	0	0.4	0.2531
	'moderate'	$\mathcal{D}_4$	$\mathbf{d}_2$	0		
	'steep'	$\mathcal{D}_5$	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_6, \mathbf{d}_7$	0.56		
ELEVATION	*'low'*	$\mathcal{D}_6$	$\mathbf{d}_2$	0	0.3333	0.3198
	'medium'	$\mathcal{D}_7$	$\mathbf{d}_3, \mathbf{d}_4$	0.5		
	'high'	$\mathcal{D}_8$	$\mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_7$	0.4444		
	'highest'	$\mathcal{D}_9$	$\mathbf{d}_6$	0		

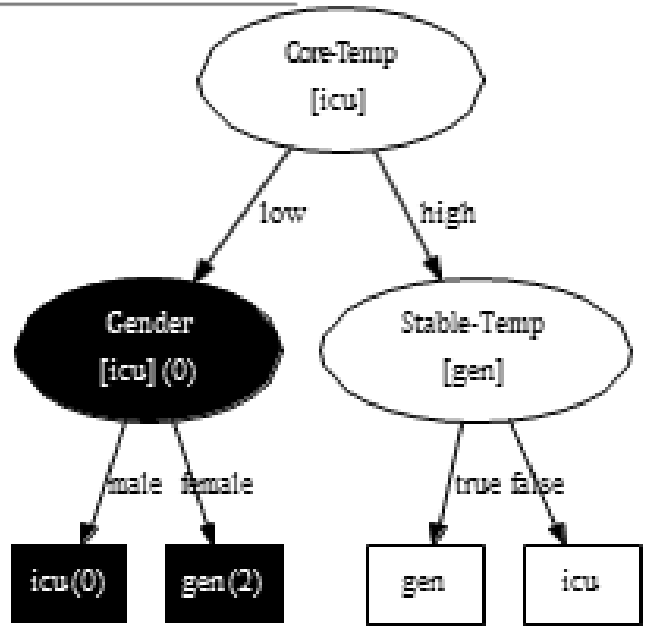
Tree Pruning

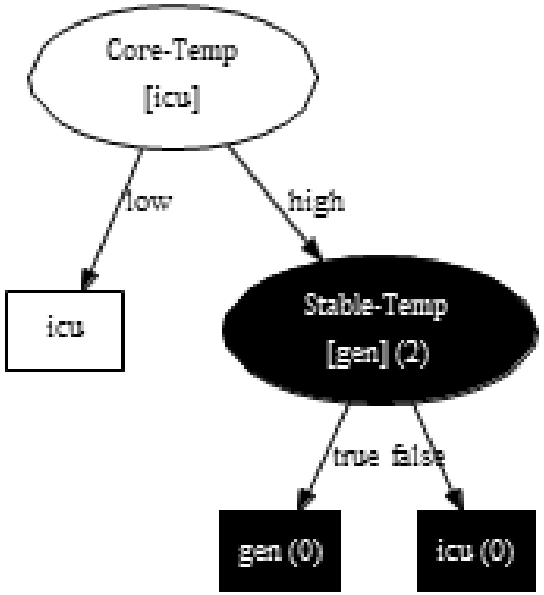
- "Over-fitting" in a decision tree occurs when we make an unnecessary split
  - Unneeded node
- More likely as we move down the tree
- So, one way to prevent overfitting is to prune the tree
- We can prune during tree generation:
  - Early stopping: limit the depth of the tree, or the size of partition to be split
  - χ^2 pruning will not create a decision that does not produce a statistically valid test
- We can prune after the fact, and use some criterion to identify and remove unneeded branches

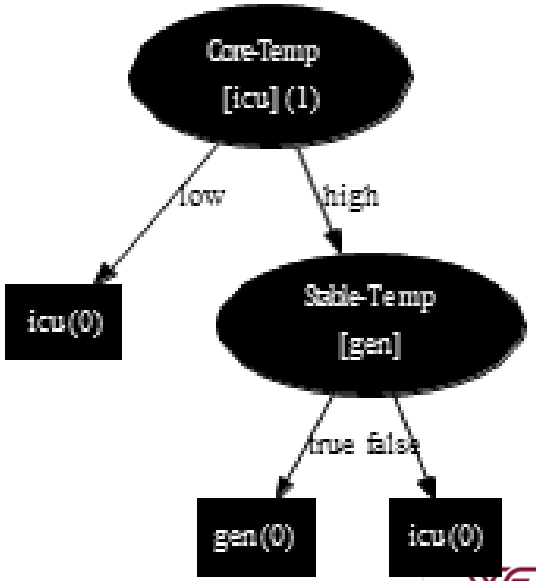
Reduced error pruning: measure accuracy on the validation set for both the full tree and the pruned tree. If the pruned tree is no worse than the fully grown tree, the node is a candidate for pruning

ID	Core-Temp	Stable-Temp	Gender	Decision
1	high	true	male	gen
2	low	true	female	icu
3	high	false	female	icu
4	high	false	male	icu
5	low	false	female	icu
6	low	true	male	icu

Reduced error pruning for a decision tree using the validation set shown. The subtree being considered for pruning in each iteration is highlighted in black. The prediction returned by each non-leaf node is listed in square brackets. The error rate for each node is given in parentheses.







DECISION TREES IN PYTHON/SCIKIT-LEARN

A wide range of data file I/O capabilities exist in the pandas tool (see pandas.io/docs/user_guide/io.html)

Format Type	Data Description	Reader	Writer
text	CSV	read_csv	to_csv
text	Fixed-Width Text File	read_fwf	
text	JSON	read_json	to_json
text	HTML	read_html	to_html
text	Local clipboard	read_clipboard	to_clipboard
	MS Excel	read_excel	to_excel
binary	OpenDocument	read_excel	
binary	HDF5 Format	read_hdf	to_hdf
binary	Feather Format	read_feather	to_feather
binary	Parquet Format	read_parquet	to_parquet
binary	ORC Format	read_orc	
binary	Msgpack	read_msgpack	to_msgpack
binary	Stata	read_stata	to_stata
binary	SAS	read_sas	
binary	SPSS	read_spss	
binary	Python Pickle Format	read_pickle	to_pickle
SQL	SQL	read_sql	to_sql
SQL	Google BigQuery	read_gbq	to_gbq

The scikit-learn decision tree models are designed to work with numeric features

- One approach would be to convert all variables to numerics
  - This is OK for interval and ordinals (preserving the sensible order)
  - SMALL = 1, MEDIUM = 2, LARGE = 3 works fine for a decision tree
  - SMALL = 8, MEDIUM = 12, LARGE = 16 is good too (ounces)
  - Don't do this for categoricals: {ANIMAL, VEGETABLE, MINERAL}
- The DecisionTreeClassifier() and many other tools will generate splits in the range of values that may not make sense (**`if (type < MINERAL)`**)
- It can be better to have three binary flags than one three-value categorical
  - isAnimal, isVegetable, isMineral : exactly one will be true for any example
- We discovered that this is called "one-hot encoding"

One-Hot rendition is often done for categoricals, so that we don't interpret them as ordinals!

- There is a practical limit on how many different categories can be transformed into binaries
  - In my experience, the US states/territories (54-ish) is about the limit

Convert the categorical data

	Age	Workclass
1	41	Private
2	33	State-gov
3	27	Federal-gov

Get dummy

	Age	Work Class: Private	Work Class: State-gov	Work Class: Federal-gov
1	41	1	0	0
2	33	0	1	0
3	27	0	0	1

It's common to create an "other" bin for rare cases:

ICECREAM = {iceCreamChoc, iceCreamVan, iceCreamStraw, iceCreamOther}

scikit-learn contains the OneHotEncoder class for performing this transformation

- OneHotEncoder(categories='auto', drop=None, sparse=True,
  dtype=<class 'numpy.float64'>, handle_unknown='error')

- See https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html

- Categories can be generated automatically or given as an input
- One category can be "dropped" (since it's redundant; one category must be 1)
- The output datatype can be specified
- Any missing values can be represented by all zeroes in the one-hot outputs – or will generate an error (the default behavior)

The size of a decision tree can be controlled in a number of ways – to increase generalization

- **max_depth**: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
- **min_samples_split**: The minimum number of samples required to split an internal node:
  - If int, then consider min_samples_split as the minimum number.
  - If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
- **min_samples_leaf**: The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches.
  - If int, then consider min_samples_leaf as the minimum number.
  - If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
- **min_weight_fraction_leaf**: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
- **max_features**: The number of features to consider when looking for the best split:
  - If int, then consider max_features features at each split.
  - If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.
  - If "auto", then max_features=sqrt(n_features).
  - If "sqrt", then max_features=sqrt(n_features).
  - If "log2", then max_features=log2(n_features).
  - If None, then max_features=n_features.
- **max_leaf_nodes**: Grow a tree with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.
- **min_impurity_decrease**: A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

Performance of a decision tree model can be measured in several ways

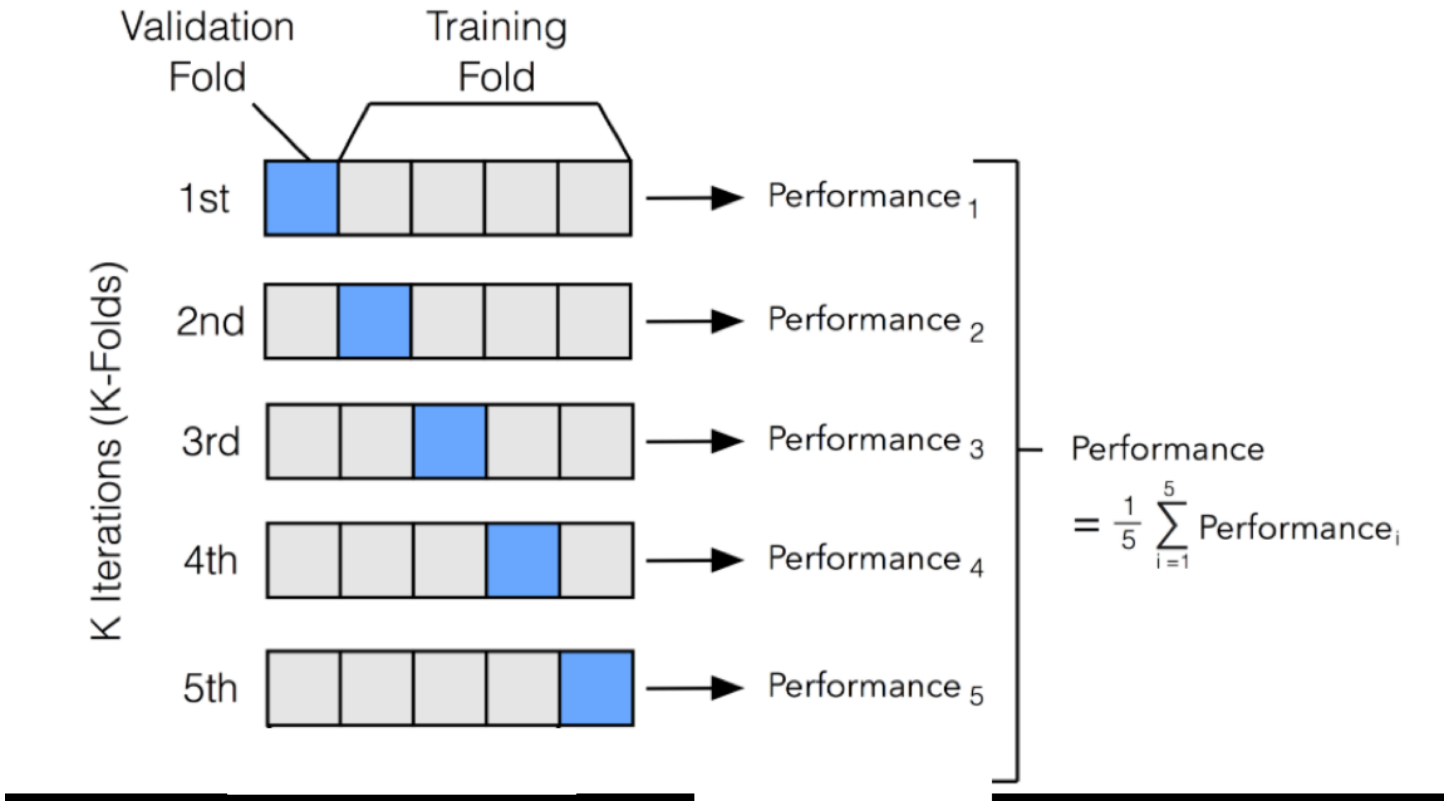
- Simple accuracy on a per-sample ratio correct basis is returned by the score(tX, ty) method

```
clf = tree.DecisionTreeClassifier(criterion="entropy", splitter="best")
clf = clf.fit(X, y)
print("Training set score = ", clf.score(X, y))
print("Test set score = ", clf.score(testX, testy))
```

('Training set score = ', 1.0)
('Test set score = ', 0.857142857142857)

- K-fold cross-validation is accomplished by the cross_val_score() class
- More info at https://scikit-learn.org/stable/modules/model_evaluation.html#scoring-parameter

k-fold cross-validation splits the dataset into *k* pieces; *k* models are each trained on *k-1* pieces combined and tested on the kth piece; scores are combined



- k-fold can also be stratified by some key variable
  - Ensuring each fold has the same distribution
- Hold-out validation is the limiting case
  - For a dataset of size N:
  - Form N folds, each with the deletion of one sample
  - Test each model on the one sample that was removed
  - Average the performance results

Selection of k-fold and other samples for cross-validation is done in scikit-learn using the *Model Selection* classes

- See https://scikit-learn.org/stable/modules/classes.html#module-sklearn.model_selection

• GroupKFold([n_splits])	K-fold iterator variant with non-overlapping groups.
• GroupShuffleSplit([…])	Shuffle-Group(s)-Out cross-validation iterator
• KFold([n_splits, shuffle, …])	K-Folds cross-validator
• LeaveOneGroupOut	Leave One Group Out cross-validator
• LeavePGroupsOut(n_groups)	Leave P Group(s) Out cross-validator
• LeaveOneOut	Leave-One-Out cross-validator
• LeavePOut(p)	Leave-P-Out cross-validator
• PredefinedSplit(test_fold)	Predefined split cross-validator
• RepeatedKFold([n_splits, …])	Repeated K-Fold cross validator.
• RepeatedStratifiedKFold([…])	Repeated Stratified K-Fold cross validator.
• ShuffleSplit([n_splits, …])	Random permutation cross-validator
• StratifiedKFold([n_splits, …])	Stratified K-Folds cross-validator
• StratifiedShuffleSplit([…])	Stratified ShuffleSplit cross-validator
• TimeSeriesSplit([n_splits, …])	Time Series cross-validator

Ensemble models consist of a set of related models whose outputs are aggregated

- The models must be different in some way for this to be useful:
  - Similar architecture, trained on different data
  - Different architecture

- Boosting iteratively creates new models to add to the ensemble – typically, biased towards error cases in the existing ensemble

- Bagging creates a family of models on subsets of the dataset (called bootstrap samples), using random sampling with replacement

Boosting works by iteratively creating models and adding them to the ensemble

- The iteration stops when a predefined number of models have been added.
- When we use **boosting** each new model added to the ensemble is biased to pay more attention to instances that previous models miss-classified.
- This is done by incrementally adapting the dataset used to train the models. To do this we use a **weighted dataset**

Weighted Dataset

- Each instance has an associated weight $\mathbf{w}_i \geq 0$,
- Initially set to $\frac{1}{n}$ where n is the number of instances in the dataset.
- After each model is added to the ensemble it is tested on the training data and the weights of the instances the model gets correct are decreased and the weights of the instances the model gets incorrect are increased.
- These weights are used as a distribution over which the dataset is sampled to created a replicated training set, where the replication of an instance is proportional to its weight.

During each training iteration the algorithm:

1. Induces a model and calculates the total error ϵ by summing the weights of the training instances for which the predictions made by the model are incorrect.
2. Increases the weights for the instances misclassified using:

$$w[i] \leftarrow w[i]\left(\frac{1}{2\epsilon}\right) \tag{5}$$

3. Decreases the weights for the instances correctly classified:

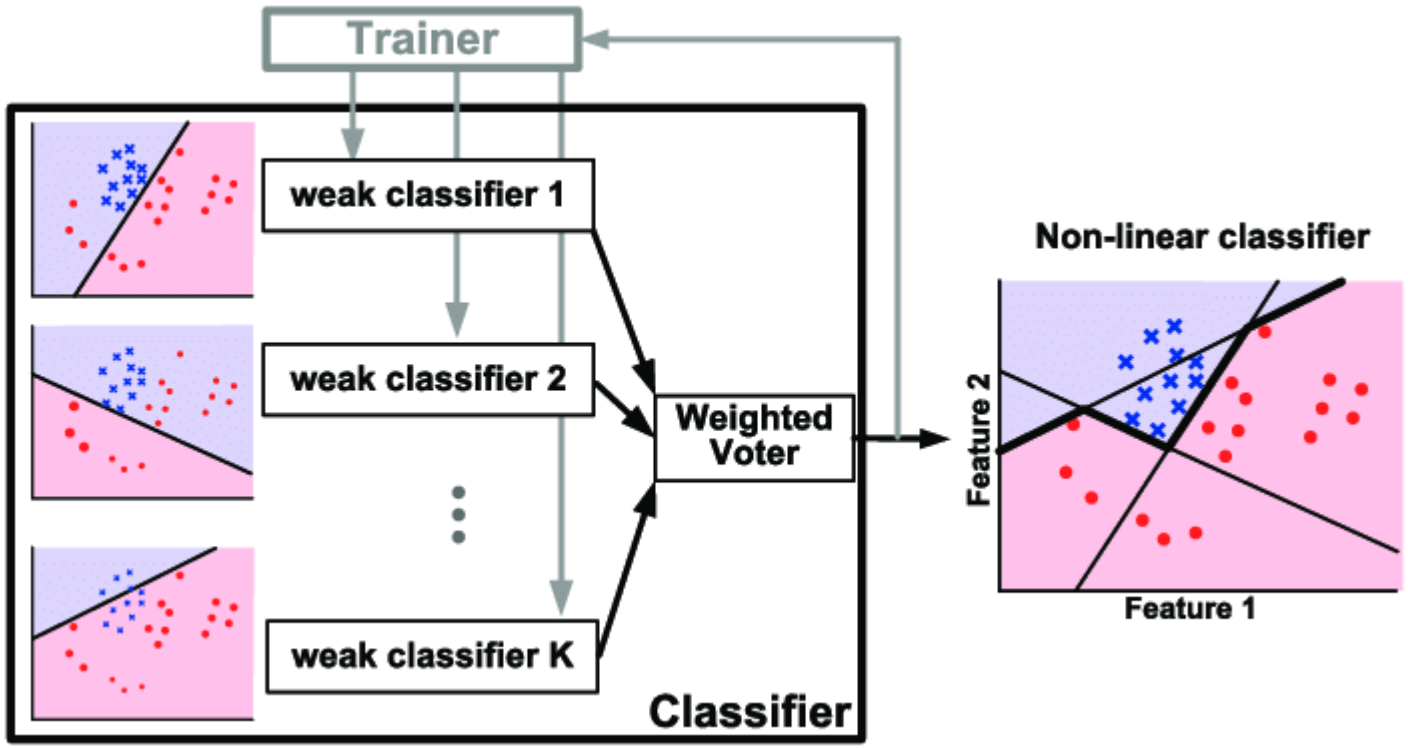
$$w[i] \leftarrow w[i]\left(\frac{1}{2(1-\epsilon)}\right) \tag{6}$$

4. Calculate a **confidence factor**, α, for the model such that

$$\alpha = \frac{1}{2} log_e\left(\frac{1-\epsilon}{\epsilon}\right) \tag{7}$$

- Once the set of models have been created the ensemble makes predictions using a weighted aggregate of the predictions made by the individual models.
- The weights used in this aggregation are simply the confidence factors associated with each model.

AdaBoost is a *meta-algorithm* for combining several "weak" classifiers into a more robust ensemble



- Outputs from several classifiers are weighted and summed
- The result does a better job of a complex task

- How are the classifiers weighted?
- Is there a difference in training?

In AdaBoost, each classifier is weighted according to it's total error; also, during training, certain samples receive higher weights ("boost") if the error is high

For iteration m=1,...,M:

(1) Fit weak classifiers to the data set and select the one with the lowest weighted classification error:

$$\epsilon_m = E_{w_m}[1_{y \neq f(x)}]$$

(2) Calculate the weight for the m_th weak classifier:

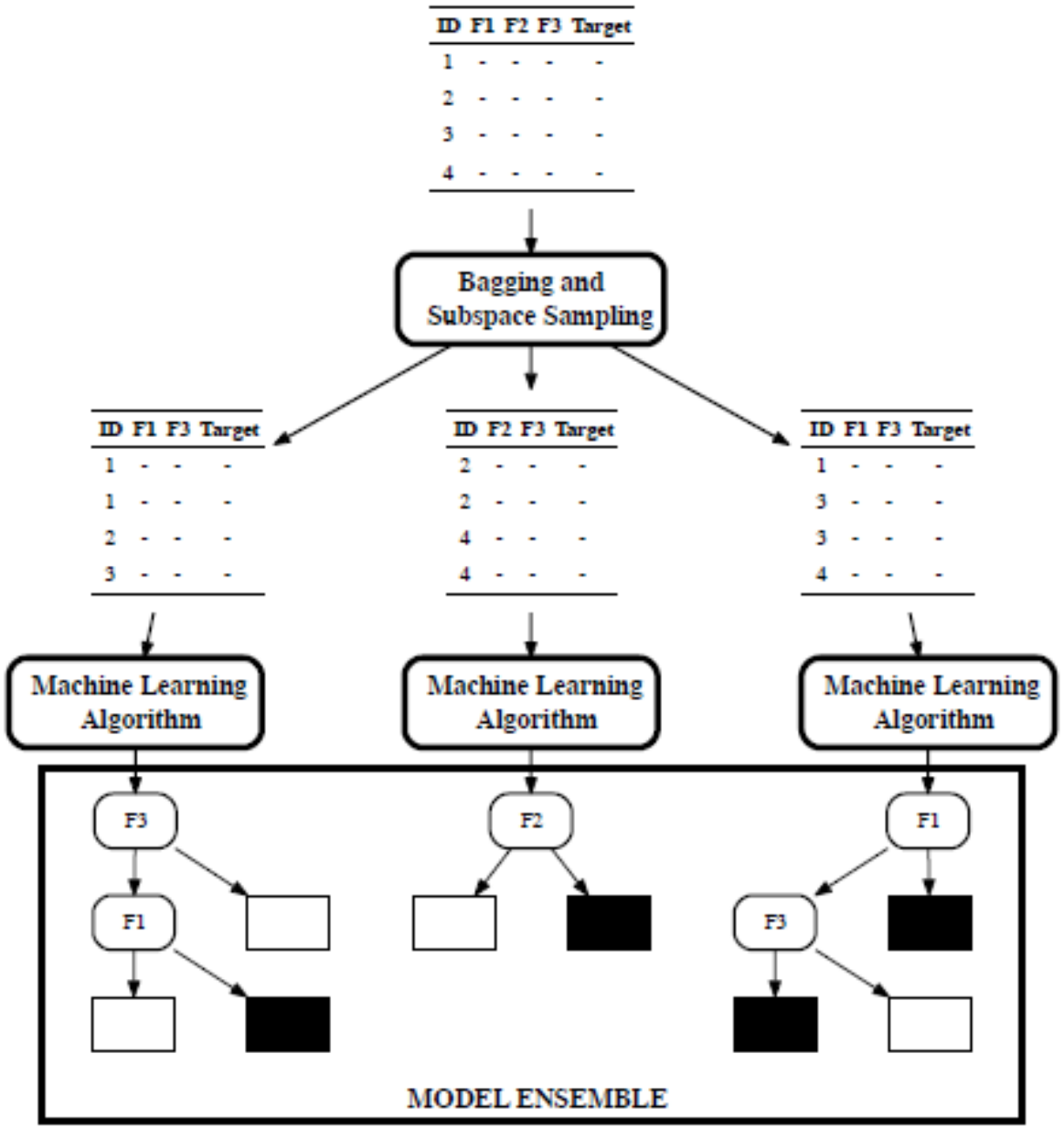
$$\theta_m = \frac{1}{2} ln(\frac{1-\epsilon_m}{\epsilon_m}).$$

Bagging

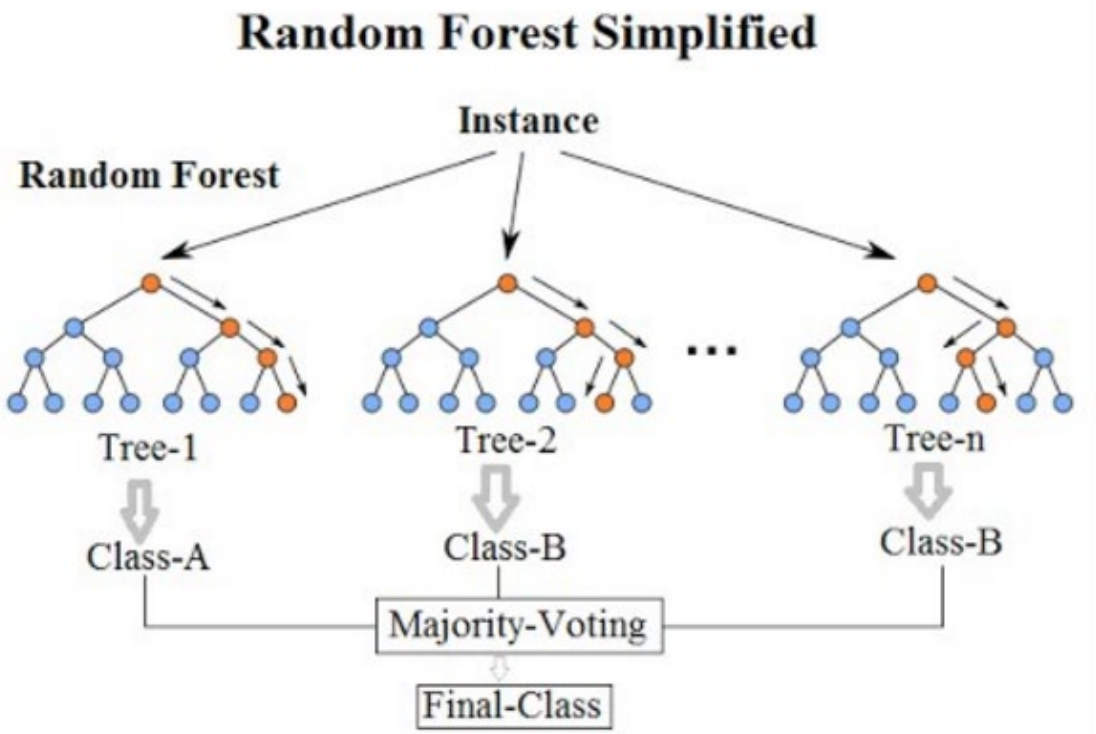
- When we use **bagging** (or **bootstrap aggregating**) each model in the ensemble is trained on a random sample of the dataset known as **bootstrap samples**.
- Each random sample is the same size as the dataset and **sampling with replacement** is used.
- Consequently, every bootstrap sample will be missing some of the instances from the dataset so each bootstrap sample will be different and this means that models trained on different bootstrap samples will also be different

- When bagging is used with decision trees each bootstrap sample only uses a randomly selected subset of the descriptive features in the dataset. This is known as **subspace sampling**.
- The combination of bagging, subspace sampling, and decision trees is known as a **random forest** model.

The process of creating a model ensemble using bagging and subspace sampling



In a Random Forest, many trees are trained on subsets of the data, produced by sampling with replacement, and results are combined



- "This bootstrapping procedure leads to better model performance because it decreases the variance of the model, without increasing the bias. This means that while the predictions of a single tree are highly sensitive to noise in its training set, the average of many trees is not, as long as the trees are not correlated. Simply training many trees on a single training set would give strongly correlated trees (or even the same tree many times, if the training algorithm is deterministic); bootstrap sampling is a way of de-correlating the trees by showing them different training sets."
  - Wikipedia

There are some heuristic guidelines for use of ensembles of trees

- Which approach should we use? Bagging is simpler to implement and parallelize than boosting and, so, may be better with respect to ease of use and training time.
- Empirical results indicate:
  - boosted decision tree ensembles were the best performing model of those tested for datasets containing up to 4,000 descriptive features.
  - Random forest ensembles (based on bagging) performed better for datasets containing more that 4,000 features.

Today's Objectives

- Information Gain for Categorical Variables
- Other Impurity Measures
- Tree Pruning
- Decision Trees in Python
  - File I/O
  - Data Preparation
    - OneHot rendition
  - Selection Method
  - Tree Size and Shape
  - Performance
- Boosting
- Bagging
- Random Forest