Decision trees: additional concepts

Mauricio A. Álvarez, PhD

Scalable Machine Learning,
University of Sheffield

Contents

Contents

Nodes in a decision tree

- A decision tree consists of:
  1. a **root node** (or starting node),
  2. **interior nodes**
  3. and **leaf nodes** (or terminating nodes).

- Each of the non-leaf nodes (root and interior) in the tree specifies a test to be carried out on one of the query's descriptive features.

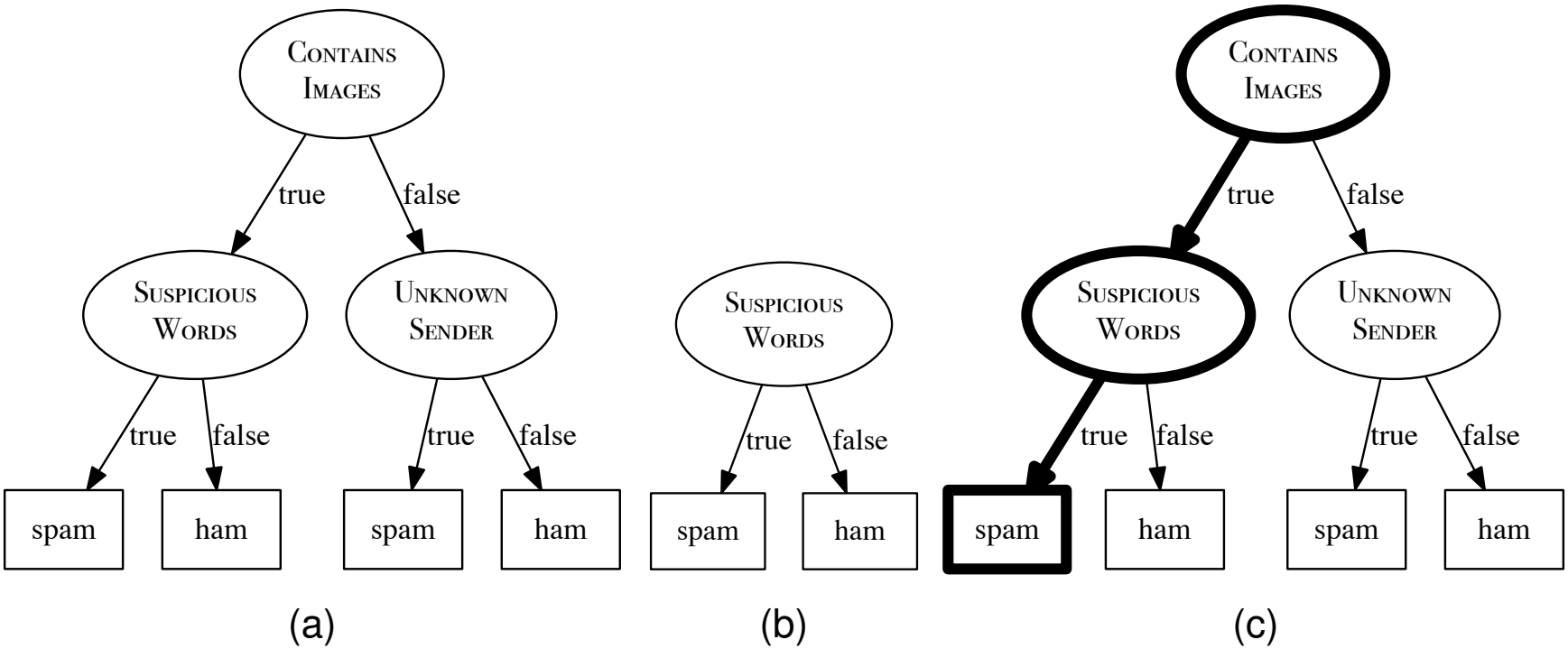
- Each of the leaf nodes specifies a predicted classification for the query.

An email spam prediction dataset

An email spam prediction dataset.

ID	Suspicious Words	Unknown Sender	Contains Images	Class
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Two decision trees and a query instance



(a) and (b) show two decision trees that are consistent with the instances in the spam dataset. (c) shows the path taken through the tree shown in (a) to make a prediction for the query instance: SUSPICIOUS WORDS = *'true'*, UNKNOWN SENDER = *'true'*, CONTAINS IMAGES = *'true'*.

Mathematical definition of entropy

- Shannon's model of entropy is a weighted sum of the logs of the probabilities of each of the possible outcomes when we make a random selection from a set.

$$H(t) = -\sum_{i=1}^{I} (P(t = i) \times log_2(P(t = i)))$$

Information Gain

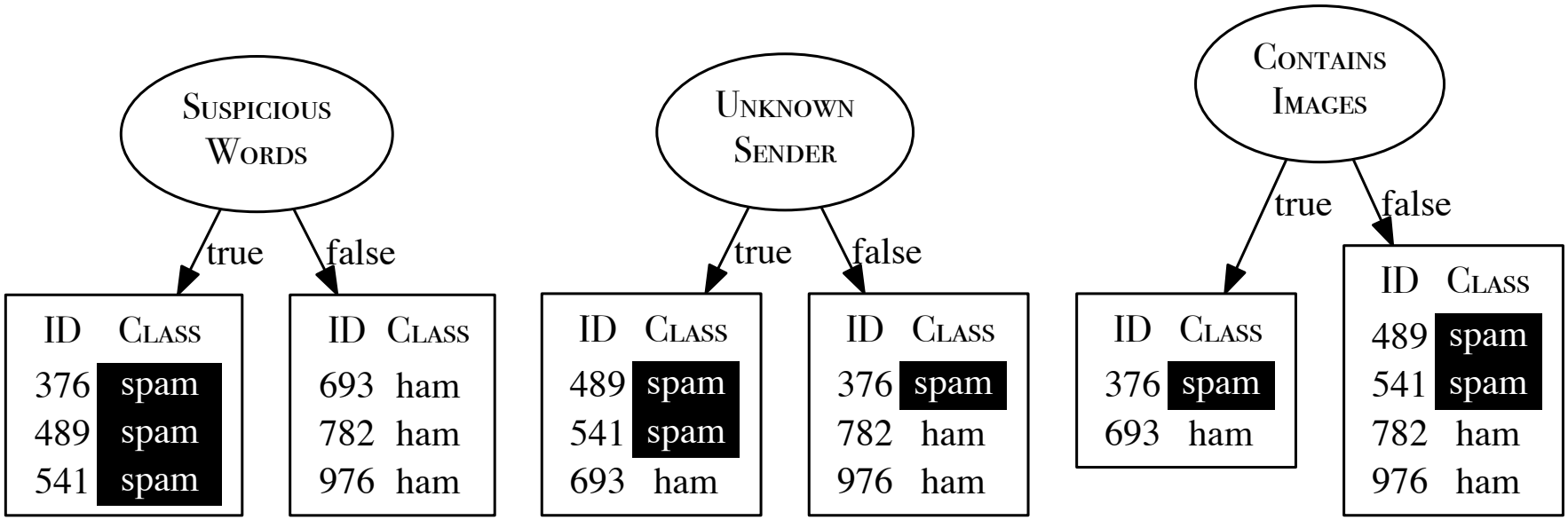
- The information gain of a descriptive feature can be understood as a measure of the reduction in the overall entropy of a prediction task by testing on that feature.

The spam dataset again

An email spam prediction dataset.

ID	Suspicious Words	Unknown Sender	Contains Images	Class
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Partitions in the spam dataset



How the instances in the spam dataset split when we partition using each of the different descriptive features from the spam dataset table.

Computing the information gain

1. Compute the entropy of the original dataset with respect to the target feature.

2. For each descriptive feature, create the sets that result by partitioning the instances in the dataset using their feature values, and then sum the entropy scores of each of these sets.

3. Subtract the remaining entropy value (computed in step 2) from the original entropy value (computed in step 1) to give the information gain.

Computing the information gain

Computing information gain involves the following 3 equations:

$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l)))$$

$$rem(d, \mathcal{D}) = \sum_{l \in levels(d)} \underbrace{\frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|}}_{\text{weighting}} \times \underbrace{H(t, \mathcal{D}_{d=l})}_{\substack{\text{entropy of} \\ \text{partition } \mathcal{D}_{d=l}}}$$

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

Example with the spam dataset

- As an example we will calculate the information gain for each of the descriptive features in the spam email dataset.

Step 1: Entropy for the target feature

- Calculate the entropy for the target feature in the dataset.

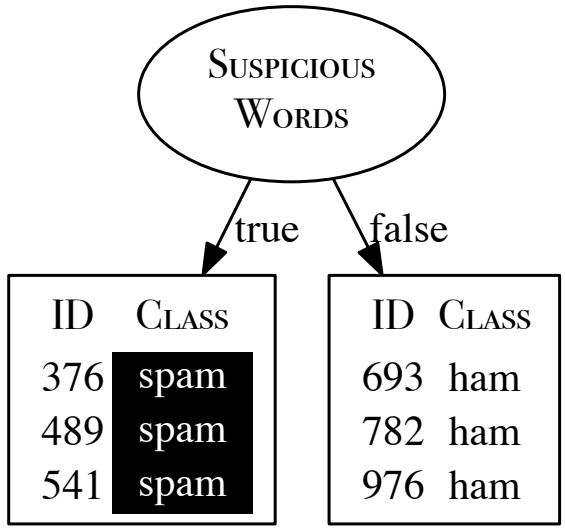
$$H(t, \mathcal{D}) = -\sum_{l \in levels(t)} (P(t = l) \times log_2(P(t = l)))$$

ID	Suspicious Words	Unknown Sender	Contains Images	Class
376	true	false	true	spam
489	true	true	false	spam
541	true	true	false	spam
693	false	true	true	ham
782	false	false	false	ham
976	false	false	false	ham

Step 1: Entropy for the target feature

$$
\begin{aligned}
H(t, \mathcal{D}) = & - \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} (P(t = l) \times log_2(P(t = l))) \\
= & - ((P(t = \text{'spam'}) \times log_2(P(t = \text{'spam'})) \\
& + (P(t = \text{'ham'}) \times log_2(P(t = \text{'ham'}))) \\
= & -\left(\left({}^{3}/_{6} \times log_2({}^{3}/_{6})\right) + \left({}^{3}/_{6} \times log_2({}^{3}/_{6})\right)\right) \\
= & \quad 1 \ bit
\end{aligned}
$$

Step 2: Partitions for the SUSPICIOUS W. feature

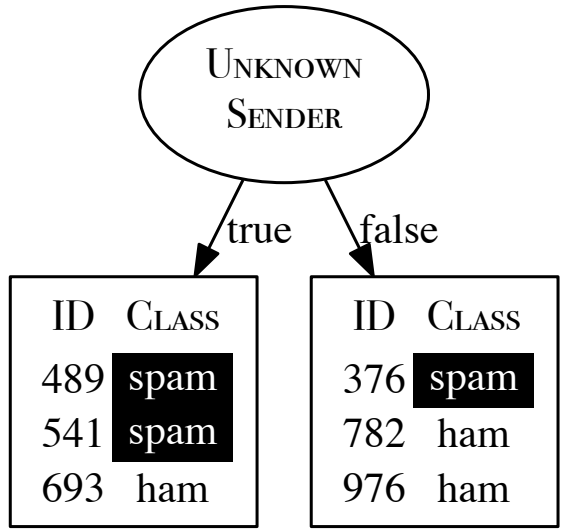


Partitions for the SUSPICIOUS W. feature

Step 2: Remainder for the SUSPICIOUS W. feature

$$
\begin{aligned}
&rem\,(\text{WORDS}, \mathcal{D}) \\
&= \left(\frac{|\mathcal{D}_{\text{WORDS}=T}|}{|\mathcal{D}|} \times H(t, \mathcal{D}_{\text{WORDS}=T}) \right) + \left(\frac{|\mathcal{D}_{\text{WORDS}=F}|}{|\mathcal{D}|} \times H(t, \mathcal{D}_{\text{WORDS}=F}) \right) \\
&= \left({}^{3}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&+ \left({}^{3}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&= \left({}^{3}/_{6} \times \left(- \left(\left({}^{3}/_{3} \times log_2({}^{3}/_{3}) \right) + \left({}^{0}/_{3} \times log_2({}^{0}/_{3}) \right) \right) \right) \right) \\
&+ \left({}^{3}/_{6} \times \left(- \left(\left({}^{0}/_{3} \times log_2({}^{0}/_{3}) \right) + \left({}^{3}/_{3} \times log_2({}^{3}/_{3}) \right) \right) \right) \right) = 0 \; bits
\end{aligned}
$$

Step 2: Partitions for the UNKNOWN SENDER feature

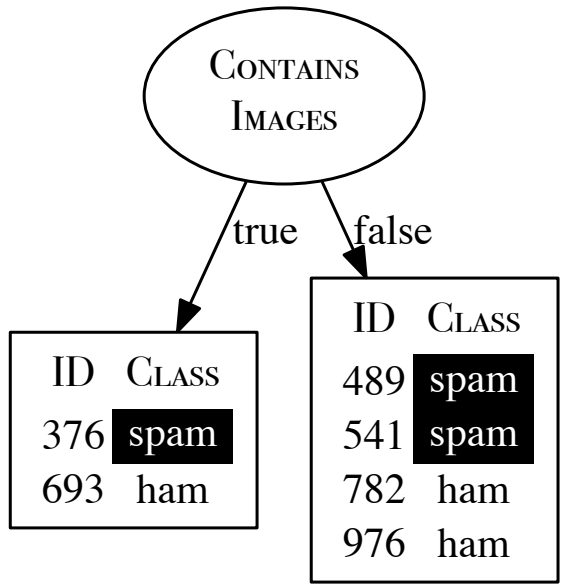


Partitions for the UNKNOWN SENDER feature

Step 2: Remainder for the UNKNOWN SENDER feature

$$
\begin{aligned}
&rem\,(\text{SENDER}, \mathcal{D}) \\
&= \left(\frac{|\mathcal{D}_{\text{SENDER}=T}|}{|\mathcal{D}|} \times H\,(t, \mathcal{D}_{\text{SENDER}=T}) \right) + \left(\frac{|\mathcal{D}_{\text{SENDER}=F}|}{|\mathcal{D}|} \times H\,(t, \mathcal{D}_{\text{SENDER}=F}) \right) \\
&= \left({}^{3}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&+ \left({}^{3}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&= \left({}^{3}/_{6} \times \left(- \left(\left({}^{2}/_{3} \times log_2({}^{2}/_{3}) \right) + \left({}^{1}/_{3} \times log_2({}^{1}/_{3}) \right) \right) \right) \right) \\
&+ \left({}^{3}/_{6} \times \left(- \left(\left({}^{1}/_{3} \times log_2({}^{1}/_{3}) \right) + \left({}^{2}/_{3} \times log_2({}^{2}/_{3}) \right) \right) \right) \right) = 0.9183\ bits
\end{aligned}
$$

Step 2: Partitions for the CONTAINS IMAGES feature



Partitions for the CONTAINS IMAGES feature

Step 2: Remainder for the CONTAINS IMAGES feature

$$
\begin{aligned}
&rem\,(\text{IMAGES}, \mathcal{D}) \\
&= \left(\frac{|\mathcal{D}_{\text{IMAGES}=T}|}{|\mathcal{D}|} \times H\,(t, \mathcal{D}_{\text{IMAGES}=T}) \right) + \left(\frac{|\mathcal{D}_{\text{IMAGES}=F}|}{|\mathcal{D}|} \times H\,(t, \mathcal{D}_{\text{IMAGES}=F}) \right) \\
&= \left({}^{2}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&+ \left({}^{4}/_{6} \times \left(- \sum_{l \in \{\text{'spam'}, \text{'ham'}\}} P(t = l) \times log_2(P(t = l)) \right) \right) \\
&= \left({}^{2}/_{6} \times \left(- \left(\left({}^{1}/_{2} \times log_2({}^{1}/_{2}) \right) + \left({}^{1}/_{2} \times log_2({}^{1}/_{2}) \right) \right) \right) \right) \\
&+ \left({}^{4}/_{6} \times \left(- \left(\left({}^{2}/_{4} \times log_2({}^{2}/_{4}) \right) + \left({}^{2}/_{4} \times log_2({}^{2}/_{4}) \right) \right) \right) \right) = 1\ bit
\end{aligned}
$$

Step 3: Compute the Information Gain

- Calculate the information gain for the three descriptive feature in the dataset.

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

Step 3: Compute the Information Gain

$$
\begin{aligned}
IG(\textsc{Suspicious Words}, \mathcal{D}) &= H(\textsc{Class}, \mathcal{D}) - rem(\textsc{Suspicious Words}, \mathcal{D}) \\
&= 1 - 0 = 1 \textit{ bit} \\
IG(\textsc{Unknown Sender}, \mathcal{D}) &= H(\textsc{Class}, \mathcal{D}) - rem(\textsc{Unknown Sender}, \mathcal{D}) \\
&= 1 - 0.9183 = 0.0817 \textit{ bits} \\
IG(\textsc{Contains Images}, \mathcal{D}) &= H(\textsc{Class}, \mathcal{D}) - rem(\textsc{Contains Images}, \mathcal{D}) \\
&= 1 - 1 = 0 \textit{ bits}
\end{aligned}
$$

Contents

The ID3 Algorithm

- ID3 Algorithm (Iterative Dichotomizer 3)
- Attempts to create the shallowest tree that is consistent with the data that it is given.
- The ID3 algorithm builds the tree in a recursive, depth-first manner, beginning at the root node and working down to the leaf nodes.

How the different nodes are treated?

1. The algorithm begins by choosing the best descriptive feature to test (i.e., the best question to ask first) using **information gain**.

2. A root node is then added to the tree and labelled with the selected test feature.

3. The training dataset is then partitioned using the test.

4. For each partition a branch is grown from the node.

5. The process is then repeated for each of these branches using the relevant partition of the training set in place of the full training set and with the selected test feature excluded from further testing.

When does the recursion stop?

The algorithm defines three situations where the recursion stops and a leaf node is constructed:

1. All of the instances in the dataset have the same classification (target feature value) then return a leaf node tree with that classification as its label.

2. The set of features left to test is empty then return a leaf node tree with the majority class of the dataset as its classification.

3. If the dataset is empty return a leaf node with the majority class of the dataset at the parent node that made the recursive call.

Contents

The vegetation classification dataset

(a) chaparral veg.

(b) riparian veg.

(c) conifer veg.

The vegetation classification dataset.

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

Entropy of the target feature for the whole dataset

$$
\begin{aligned}
& H(\text{VEGETATION}, \mathcal{D}) \\
&= - \sum_{I \in \left\{ \begin{array}{l} \text{'chaparral'}, \\ \text{'riparian'}, \\ \text{'conifer'} \end{array} \right\}} P(\text{VEGETATION} = I) \times log_2 \left(P(\text{VEGETATION} = I) \right) \\
&= - \left(\left({}^3/_7 \times log_2({}^3/_7) \right) + \left({}^2/_7 \times log_2({}^2/_7) \right) + \left({}^2/_7 \times log_2({}^2/_7) \right) \right) \\
&= 1.5567 \; bits
\end{aligned}
$$

Information gain for each feature

Partition sets (Part.), entropy, remainder (Rem.) and information gain (Info. Gain) by feature for the dataset

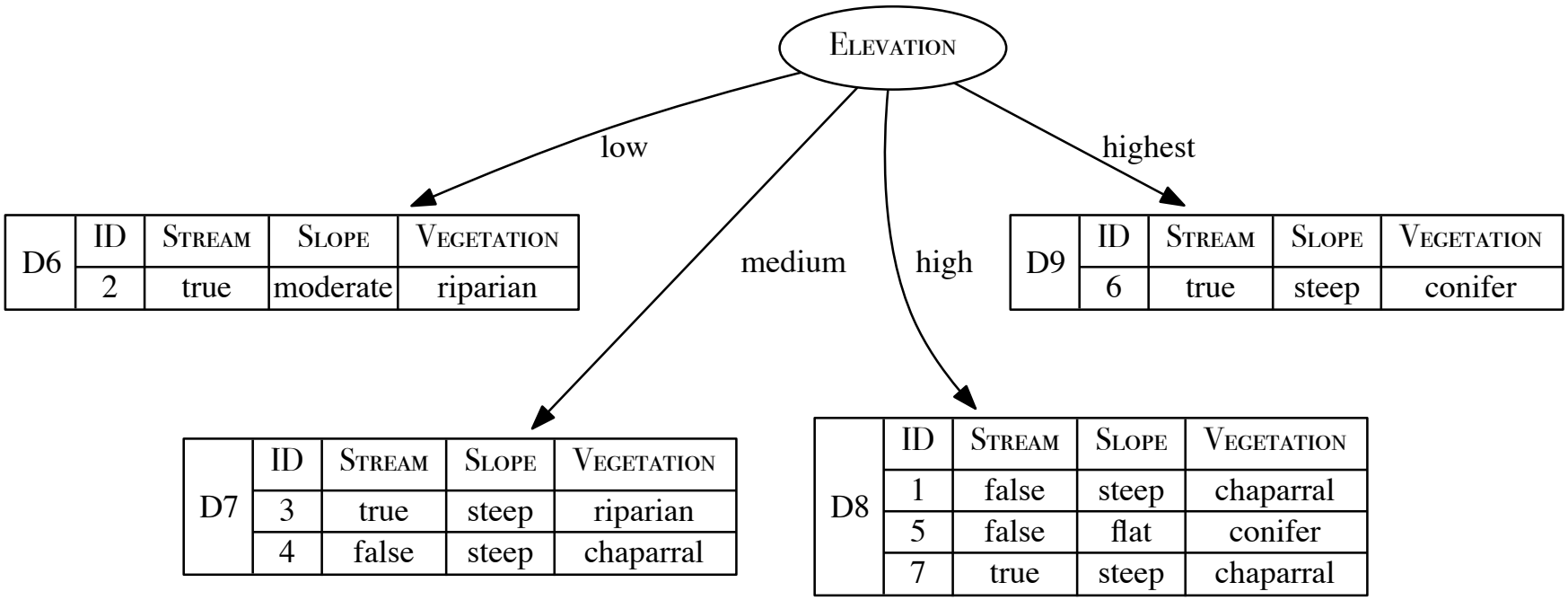
Split By Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*'true'*	$\mathcal{D}_1$	$\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_6, \mathbf{d}_7$	1.5	1.2507	0.3060
	'false'	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_4, \mathbf{d}_5$	0.9183		
SLOPE	*'flat'*	$\mathcal{D}_3$	?	?	?	?
	'moderate'	$\mathcal{D}_4$	?	?		
	'steep'	$\mathcal{D}_5$	?	?		
ELEVATION	*'low'*	$\mathcal{D}_6$	?	?	?	?
	'medium'	$\mathcal{D}_7$	?	?		
	'high'	$\mathcal{D}_8$	?	?		
	'highest'	$\mathcal{D}_9$	?	?		

Information gain for each feature

Partition sets (Part.), entropy, remainder (Rem.) and information gain (Info. Gain) by feature for the dataset

Split By Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*'true'*	$\mathcal{D}_1$	$\mathbf{d}_2, \mathbf{d}_3, \mathbf{d}_6, \mathbf{d}_7$	1.5	1.2507	0.3060
	'false'	$\mathcal{D}_2$	$\mathbf{d}_1, \mathbf{d}_4, \mathbf{d}_5$	0.9183		
SLOPE	*'flat'*	$\mathcal{D}_3$	$\mathbf{d}_5$	0	0.9793	0.5774
	'moderate'	$\mathcal{D}_4$	$\mathbf{d}_2$	0		
	'steep'	$\mathcal{D}_5$	$\mathbf{d}_1, \mathbf{d}_3, \mathbf{d}_4, \mathbf{d}_6, \mathbf{d}_7$	1.3710		
ELEVATION	*'low'*	$\mathcal{D}_6$	$\mathbf{d}_2$	0	0.6793	0.8774
	'medium'	$\mathcal{D}_7$	$\mathbf{d}_3, \mathbf{d}_4$	1.0		
	'high'	$\mathcal{D}_8$	$\mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_7$	0.9183		
	'highest'	$\mathcal{D}_9$	$\mathbf{d}_6$	0		

The decision tree after using ELEVATION



D6

ID	STREAM	SLOPE	VEGETATION
2	true	moderate	riparian

D7

ID	STREAM	SLOPE	VEGETATION
3	true	steep	riparian
4	false	steep	chaparral

D8

ID	STREAM	SLOPE	VEGETATION
1	false	steep	chaparral
5	false	flat	conifer
7	true	steep	chaparral

D9

ID	STREAM	SLOPE	VEGETATION
6	true	steep	conifer

The decision tree after the data has been split using ELEVATION.

Entropy of the target feature for the group $\mathcal{D}_7$

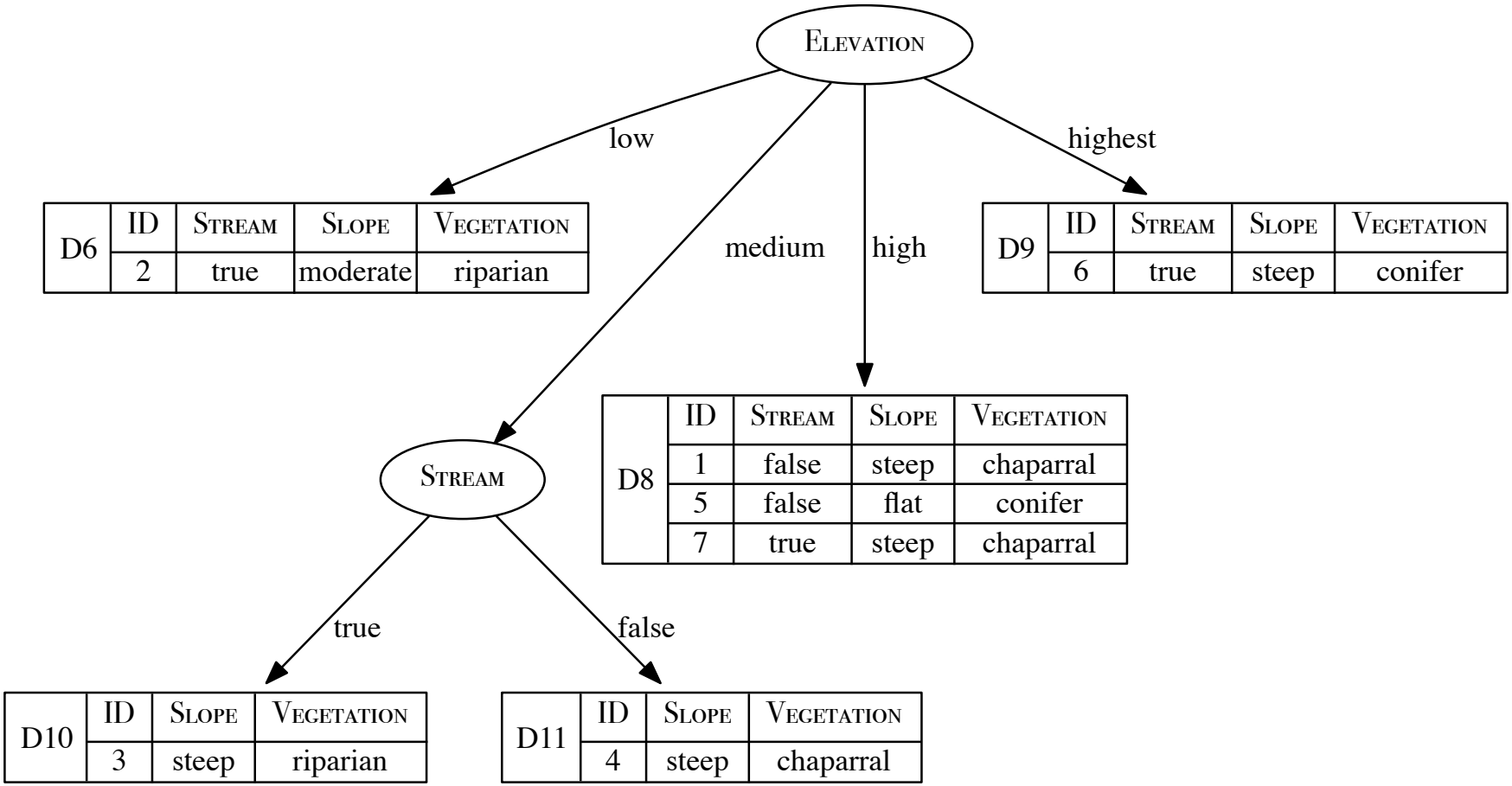
$$
\begin{aligned}
& H\left(\text{VEGETATION}, \mathcal{D}_7\right) \\
&= - \sum_{I \in \left\{ \begin{array}{l} \text{'chaparral'}, \\ \text{'riparian'}, \\ \text{'conifer'} \end{array} \right\}} P(\text{VEGETATION} = I) \times log_2 \left(P(\text{VEGETATION} = I) \right) \\
&= - \left(\left({}^1/_2 \times log_2({}^1/_2) \right) + \left({}^1/_2 \times log_2({}^1/_2) \right) + \left({}^0/_2 \times log_2({}^0/_2) \right) \right) \\
&= 1.0 \; \textit{bits}
\end{aligned}
$$

Information gain for the remaining features of $\mathcal{D}_7$

Partition sets (Part.), entropy, remainder (Rem.) and information gain (Info. Gain) by feature for the dataset $\mathcal{D}_7$ in the previous Figure

Split By Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*'true'*	$\mathcal{D}_{10}$	$\mathbf{d}_3$	0	0	1.0
	'false'	$\mathcal{D}_{11}$	$\mathbf{d}_4$	0		
SLOPE	*'flat'*	$\mathcal{D}_{12}$		0	1.0	0
	'moderate'	$\mathcal{D}_{13}$		0		
	'steep'	$\mathcal{D}_{14}$	$\mathbf{d}_3, \mathbf{d}_4$	1.0		

Decision tree after splitting $\mathcal{D}_7$ using STREAM



The state of the decision tree after the $\mathcal{D}_7$ partition has been split using STREAM.

Entropy of the target feature for the group $\mathcal{D}_8$

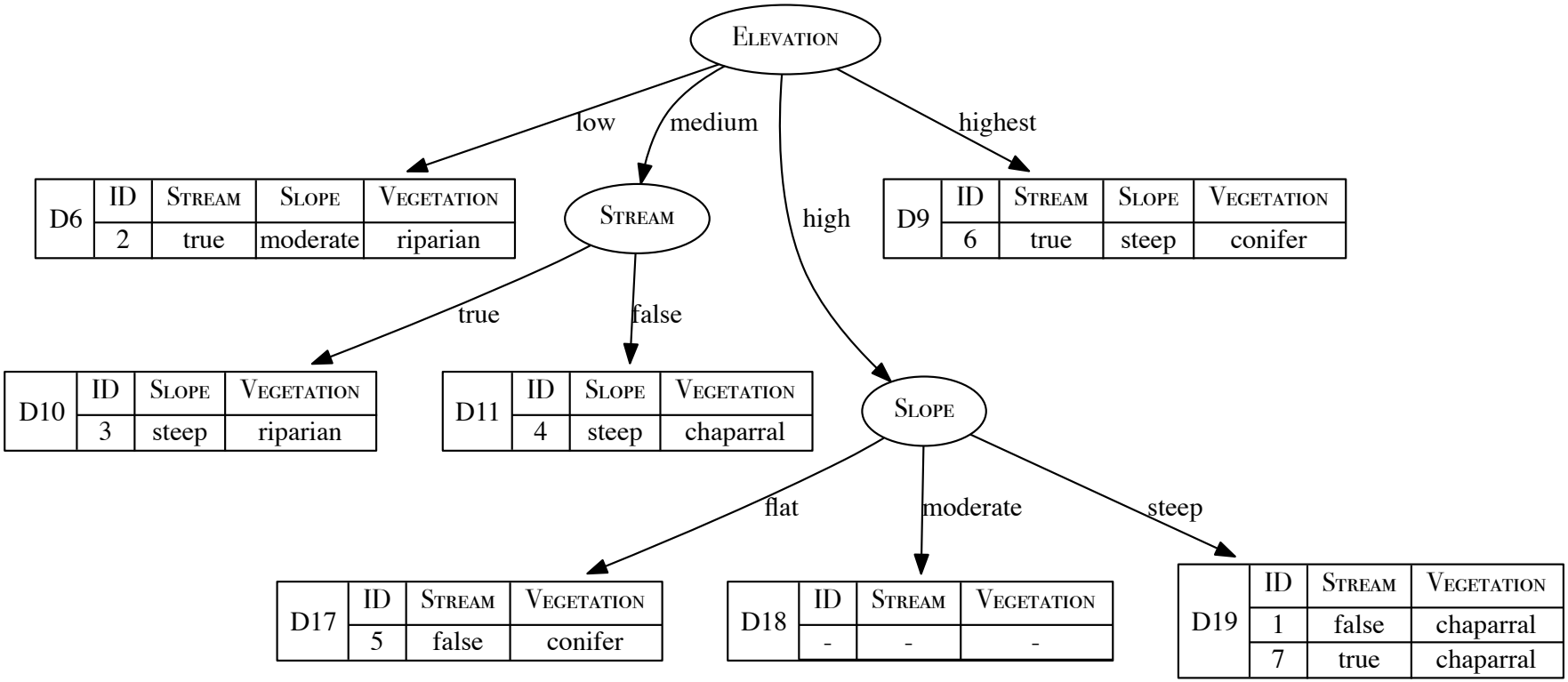
$$
\begin{aligned}
& H\left(\text{VEGETATION}, \mathcal{D}_8\right) \\
&= - \sum_{I \in \left\{ \begin{array}{l} \text{'chaparral'}, \\ \text{'riparian'}, \\ \text{'conifer'} \end{array} \right\}} P(\text{VEGETATION} = I) \times log_2 \left(P(\text{VEGETATION} = I) \right) \\
&= - \left(\left({}^2/_3 \times log_2({}^2/_3) \right) + \left({}^0/_3 \times log_2({}^0/_3) \right) + \left({}^1/_3 \times log_2({}^1/_3) \right) \right) \\
&= 0.9183 \; bits
\end{aligned}
$$

Information gain for the remaining features of $\mathcal{D}_8$

Partition sets (Part.), entropy, remainder (Rem.) and information gain (Info. Gain) by by feature for the dataset $\mathcal{D}_8$.

Split By Feature	Level	Part.	Instances	Partition Entropy	Rem.	Info. Gain
STREAM	*'true'*	$\mathcal{D}_{15}$	$\mathbf{d}_7$	0	0.6666	0.2517
	'false'	$\mathcal{D}_{16}$	$\mathbf{d}_1, \mathbf{d}_5$	1.0		
SLOPE	*'flat'*	$\mathcal{D}_{17}$	$\mathbf{d}_5$	0	0	0.9183
	'moderate'	$\mathcal{D}_{18}$		0		
	'steep'	$\mathcal{D}_{19}$	$\mathbf{d}_1, \mathbf{d}_7$	0		

Decision tree after splitting $\mathcal{D}_8$ using SLOPE



The state of the decision tree after the $\mathcal{D}_8$ partition has been split using SLOPE.

What to do with $\mathcal{D}_{18}$?

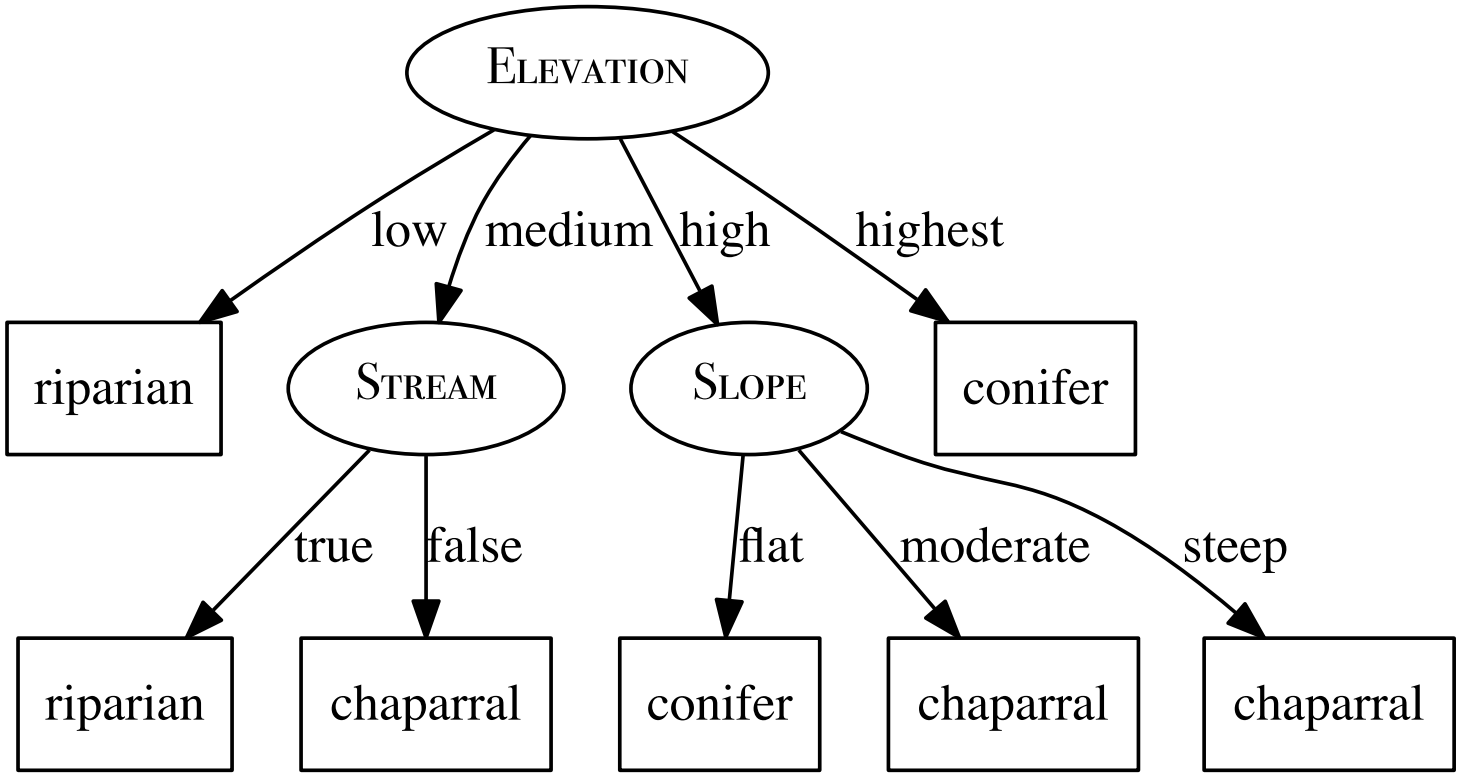
We use the third situation for deciding on the leaf:

1. All of the instances in the dataset have the same classification (target feature value) then return a leaf node tree with that classification as its label.

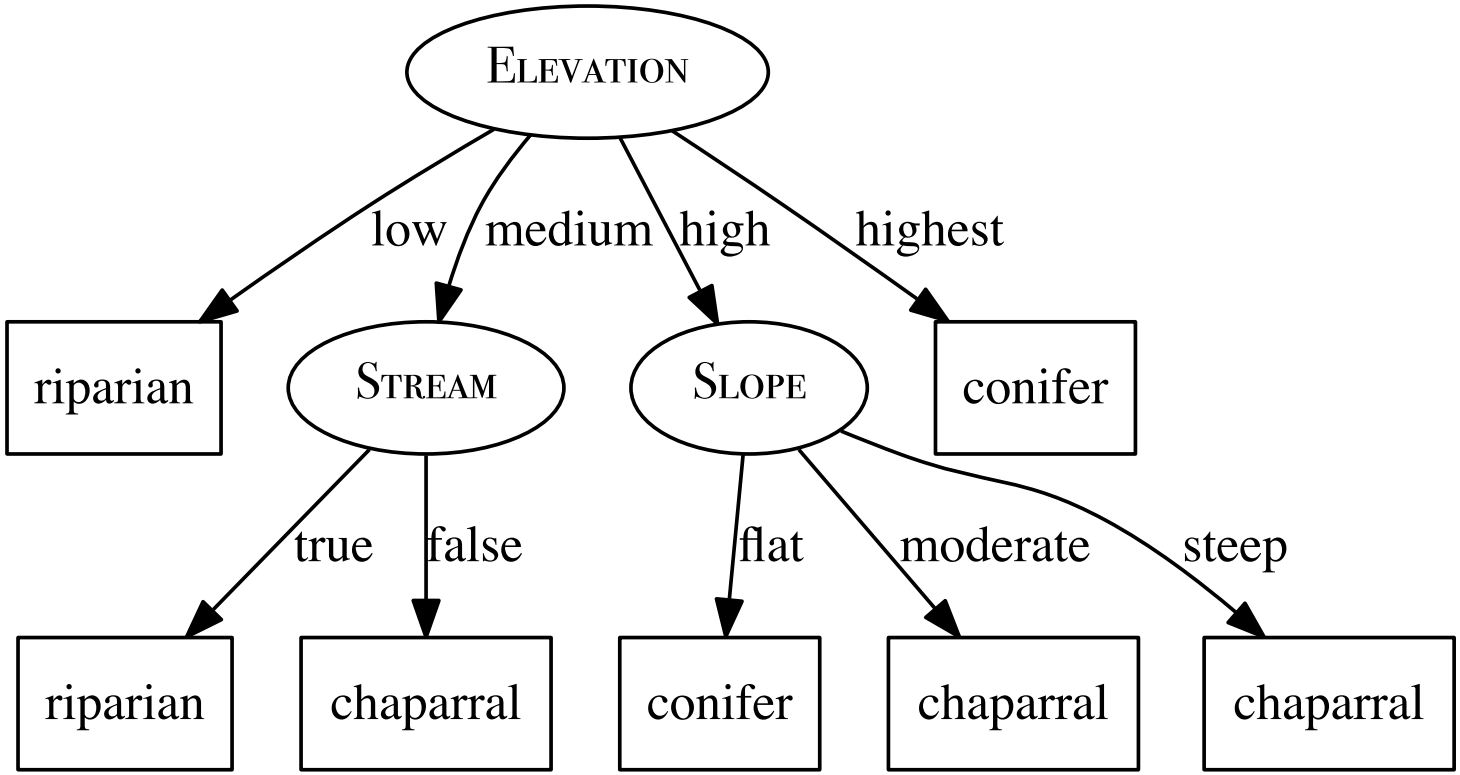
2. The set of features left to test is empty then return a leaf node tree with the majority class of the dataset as its classification.

3. **If the dataset is empty return a leaf node with the majority class of the dataset at the parent node that made the recursive call**.

The final vegetation classification decision tree



Prediction for a new query



- What prediction will this decision tree model return for the following query?

STREAM = *'true'*, SLOPE=*'Moderate'*, ELEVATION=*'High'*

Contents

Regression trees

- Regression trees are constructed so as to reduce the variance in the set of training examples at each of the leaf nodes in the tree

- We can do this by adapting the ID3 algorithm to use a measure of variance rather than a measure of classification impurity (entropy) when selecting the best attribute

Impurity for regression

- The impurity (variance) at a node can be calculated using the following equation:

$$var\left(t, \mathcal{D}\right) = \frac{\sum_{i=1}^{n}\left(t_i - \bar{t}\right)^2}{n-1}$$

- We select the feature that minimizes the weighted variance across the resulting partitions:

$$\mathbf{d}[best] = \underset{d \in \mathbf{d}}{\text{argmin}} \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} \times var(t, \mathcal{D}_{d=l})$$

A dataset listing the number of bike rentals per day

ID	Season	Work Day	Rentals
1	winter	false	800
2	winter	false	826
3	winter	true	900
4	spring	false	2 100
5	spring	true	4 740
6	spring	true	4 900
7	summer	false	3 000
8	summer	true	5 800
9	summer	true	6 200
10	autumn	false	2 910
11	autumn	false	2 880
12	autumn	true	2 820

Weighted variance according to the partitions

Partitioning of the dataset for bike rentals based on SEASON and WORK DAY features and the computation of the weighted variance for each partitioning.

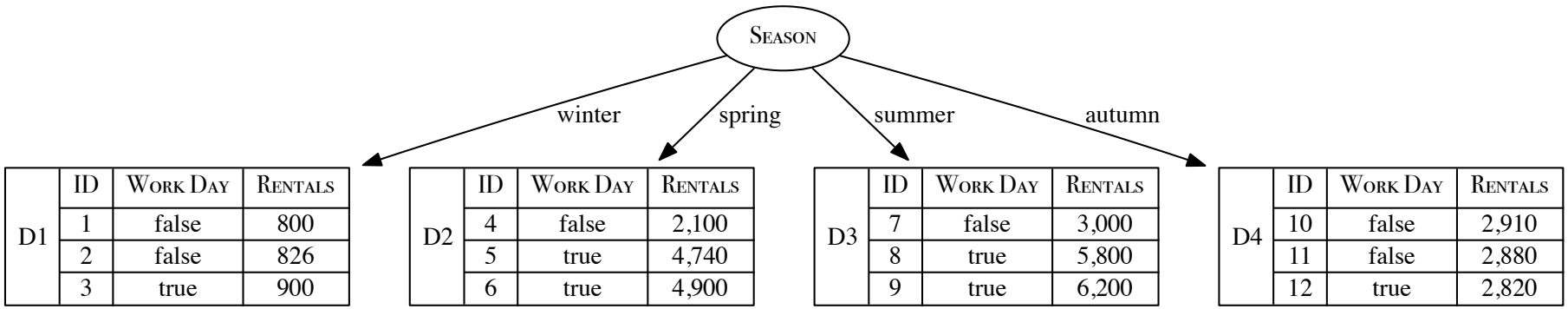
Split by Feature	Level	Part.	Instances	$\frac{\lvert\mathcal{D}_{d=l}\rvert}{\lvert\mathcal{D}\rvert}$	$var\,(t,\mathcal{D})$	Weighted Variance
SEASON	*'winter'*	$\mathcal{D}_1$	?	?	?	?
	'spring'	$\mathcal{D}_2$	?	?	?	
	'summer'	$\mathcal{D}_3$	?	?	?	
	'autumn'	$\mathcal{D}_4$	?	?	?	
WORK DAY	*'true'*	$\mathcal{D}_5$	$\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_8, \mathbf{d}_9, \mathbf{d}_{12}$	0.50	$4\,026\,346\frac{1}{3}$	$2\,551\,813\frac{1}{3}$
	'false'	$\mathcal{D}_6$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_7, \mathbf{d}_{10}, \mathbf{d}_{11}$	0.50	$1\,077\,280$	

Weighted variance according to the partitions

Partitioning of the dataset for bike rentals based on SEASON and WORK DAY features and the computation of the weighted variance for each partitioning.

Split by Feature	Level	Part.	Instances	$\frac{\lvert\mathcal{D}_{d=l}\rvert}{\lvert\mathcal{D}\rvert}$	$var\,(t,\mathcal{D})$	Weighted Variance
SEASON	*'winter'*	$\mathcal{D}_1$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_3$	0.25	2 692	$1\,379\,331\frac{1}{3}$
	'spring'	$\mathcal{D}_2$	$\mathbf{d}_4, \mathbf{d}_5, \mathbf{d}_6$	0.25	$2\,472\,533\frac{1}{3}$	
	'summer'	$\mathcal{D}_3$	$\mathbf{d}_7, \mathbf{d}_8, \mathbf{d}_9$	0.25	3 040 000	
	'autumn'	$\mathcal{D}_4$	$\mathbf{d}_{10}, \mathbf{d}_{11}, \mathbf{d}_{12}$	0.25	2 100	
WORK DAY	*'true'*	$\mathcal{D}_5$	$\mathbf{d}_3, \mathbf{d}_5, \mathbf{d}_6, \mathbf{d}_8, \mathbf{d}_9, \mathbf{d}_{12}$	0.50	$4\,026\,346\frac{1}{3}$	$2\,551\,813\frac{1}{3}$
	'false'	$\mathcal{D}_6$	$\mathbf{d}_1, \mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_7, \mathbf{d}_{10}, \mathbf{d}_{11}$	0.50	1 077 280	

Resulting decision tree



	ID	WORK DAY	RENTALS
D1	1	false	800
	2	false	826
	3	true	900

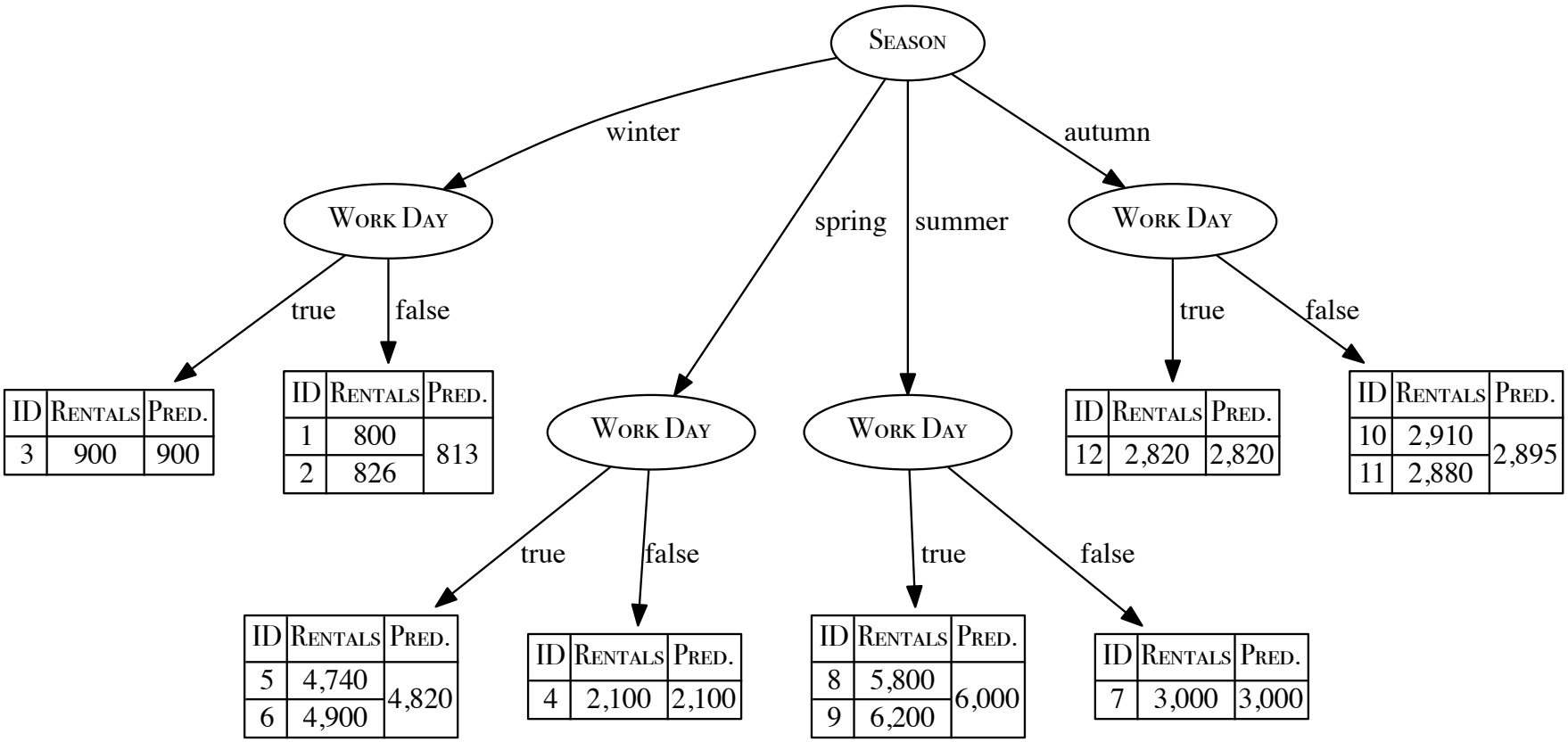
	ID	WORK DAY	RENTALS
D2	4	false	2,100
	5	true	4,740
	6	true	4,900

	ID	WORK DAY	RENTALS
D3	7	false	3,000
	8	true	5,800
	9	true	6,200

	ID	WORK DAY	RENTALS
D4	10	false	2,910
	11	false	2,880
	12	true	2,820

The decision tree resulting from splitting the data using the feature SEASON.

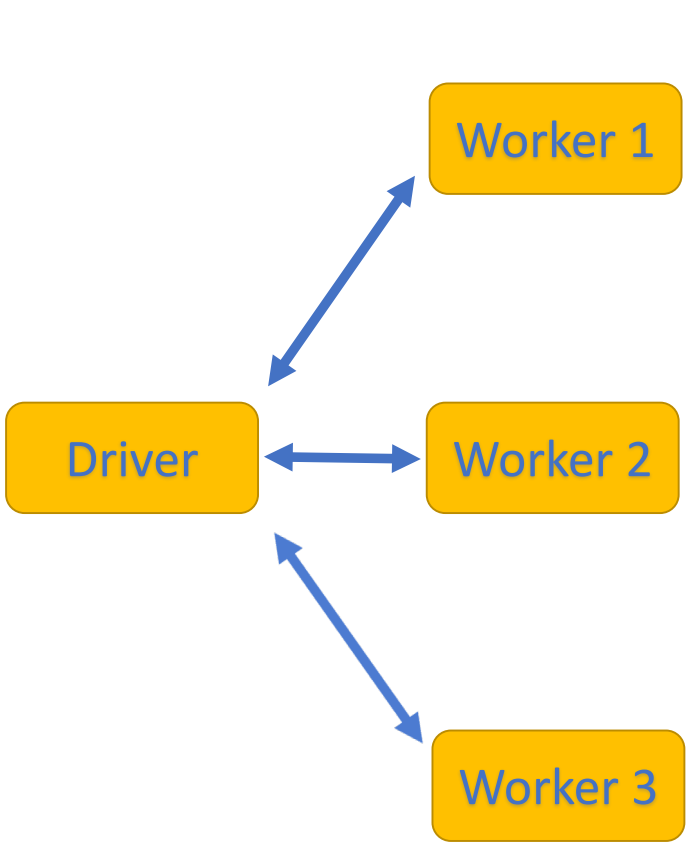
Final decision tree



The final decision tree induced from the dataset. To illustrate how the tree generates predictions, this tree lists the instances that ended up at each leaf node and the prediction (PRED.) made by each leaf node.

Contents

The distributed dataset



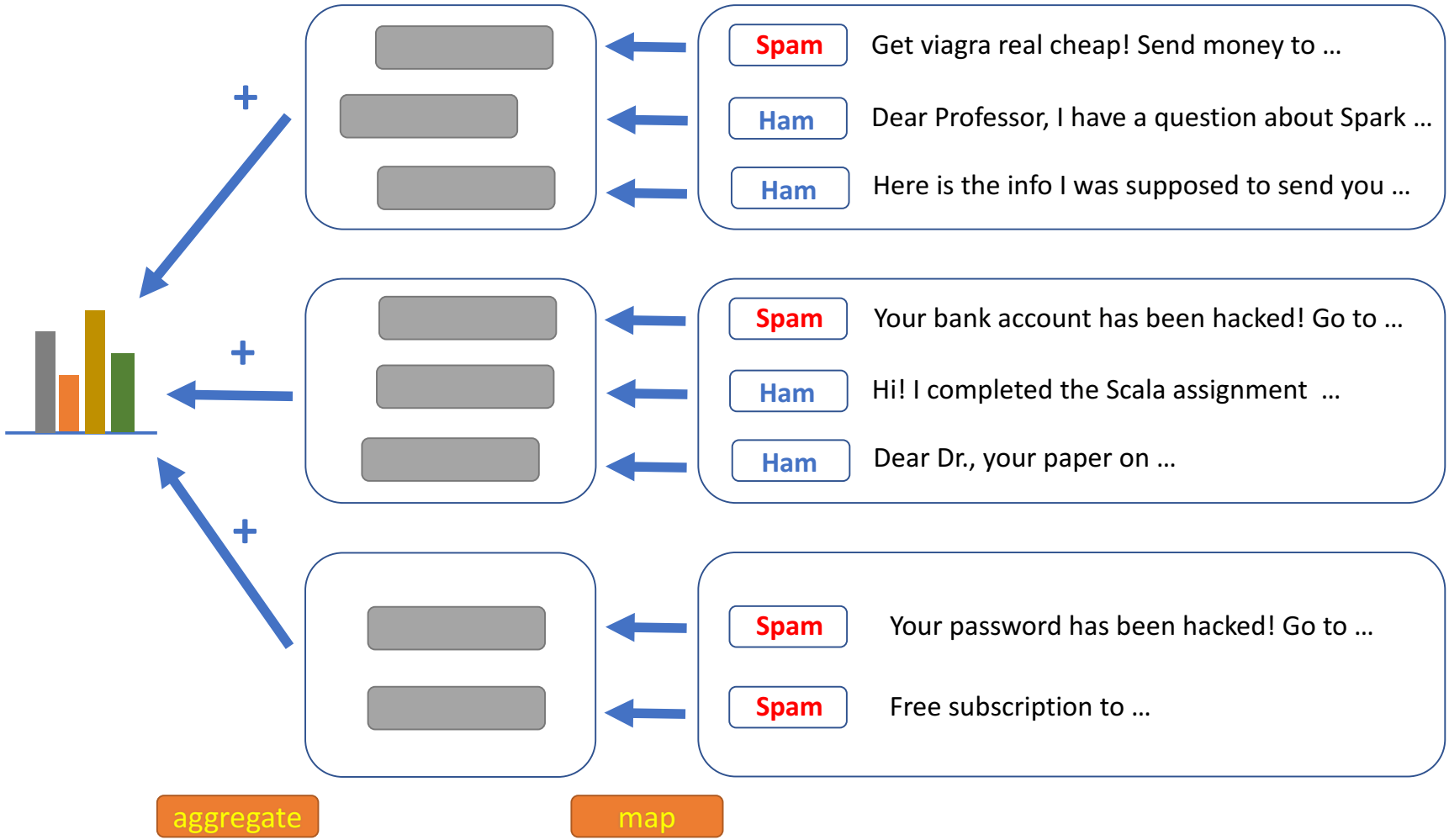
Spam	Get viagra real cheap! Send money to …
Ham	Dear Professor, I have a question about Spark …
Ham	Here is the info I was supposed to send you …

Spam	Your bank account has been hacked! Go to …
Ham	Hi! I completed the Scala assignment …
Ham	Dear Dr., your paper on …

Spam	Your password has been hacked! Go to …
Spam	Free subscription to …

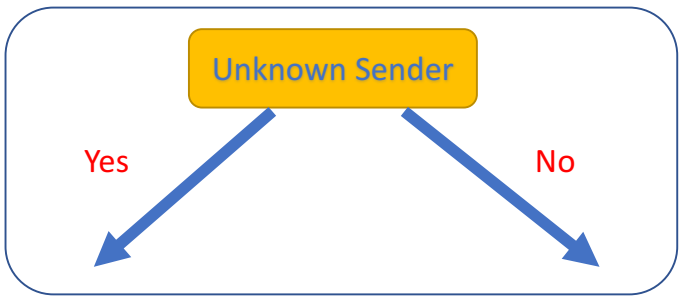
A distributed dataset

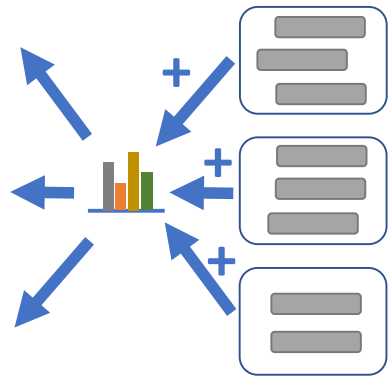
The distributed dataset



A distributed dataset

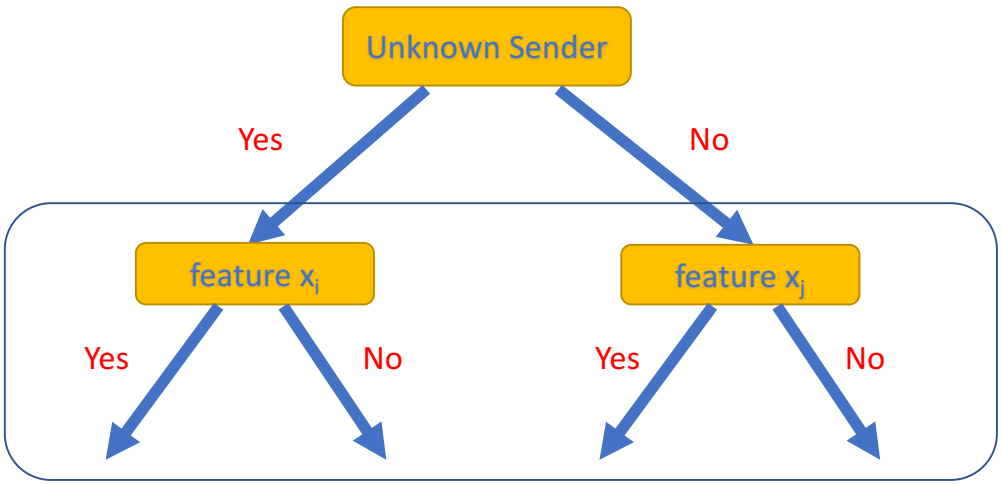
Building a decision tree

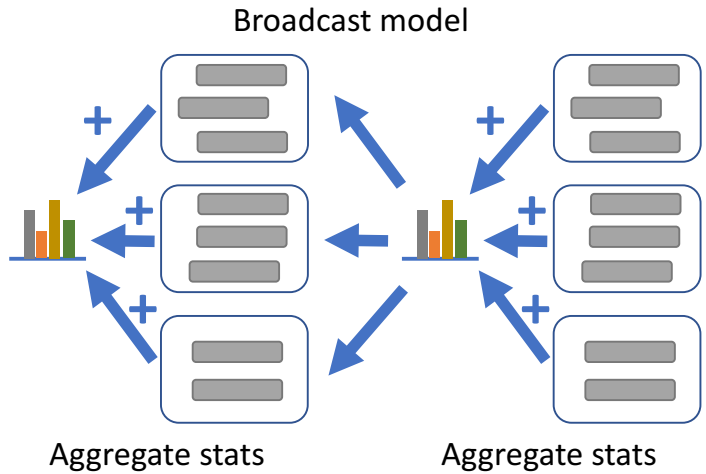




Building the tree in Spark

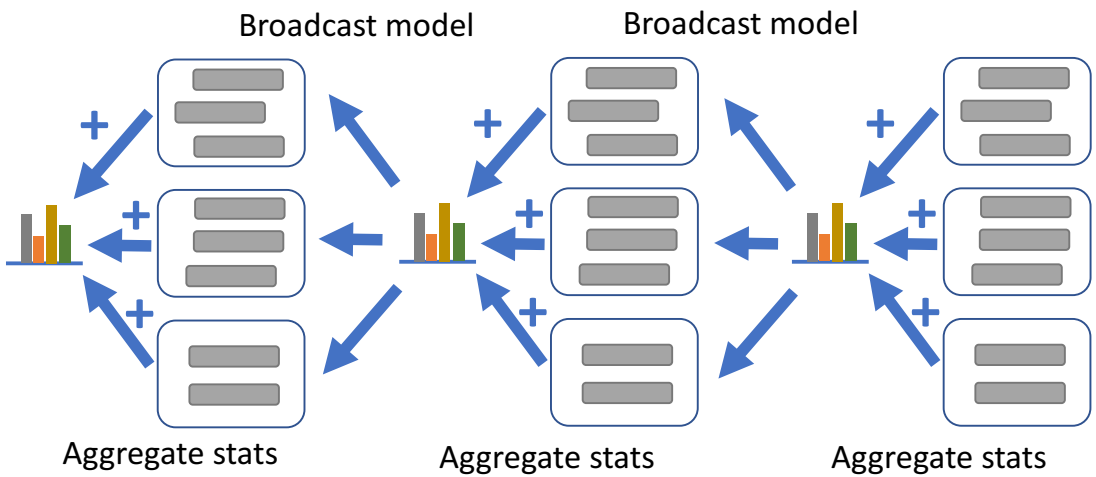
Building a decision tree

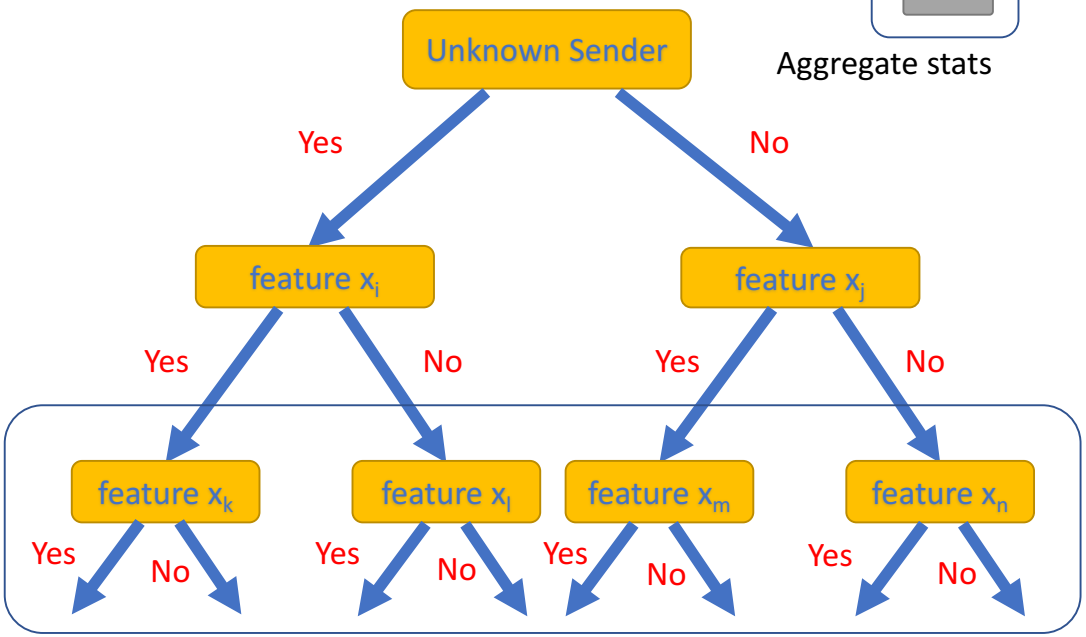




Building the tree in Spark

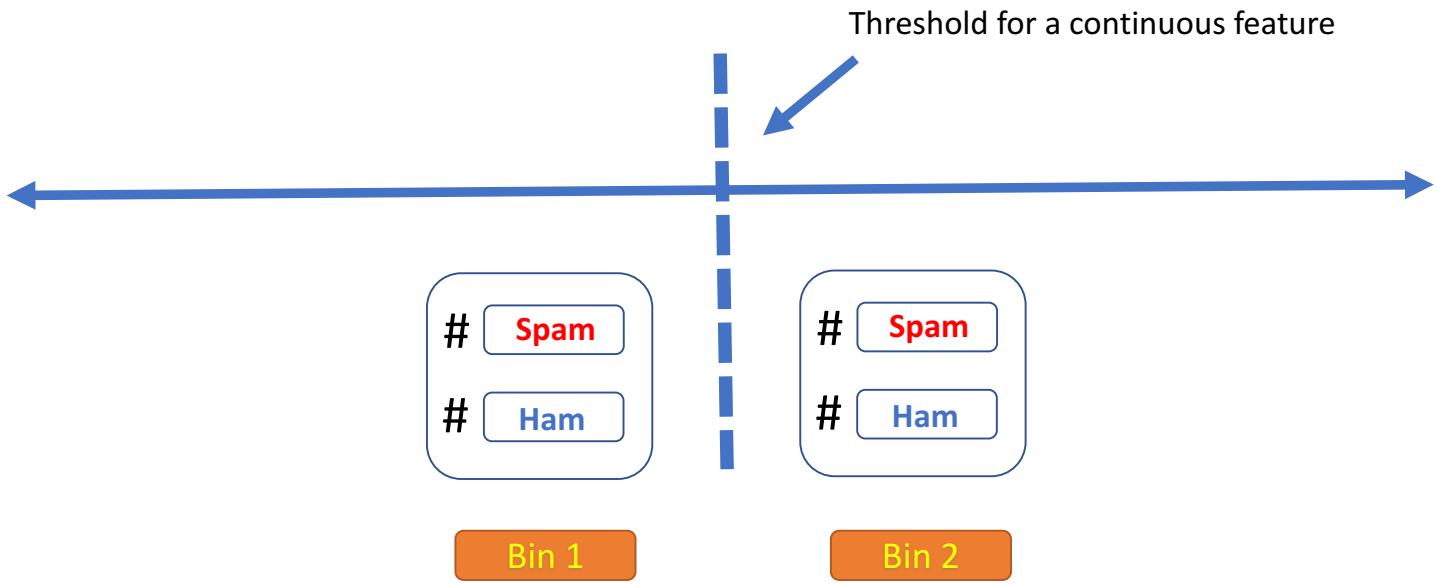
Building a decision tree





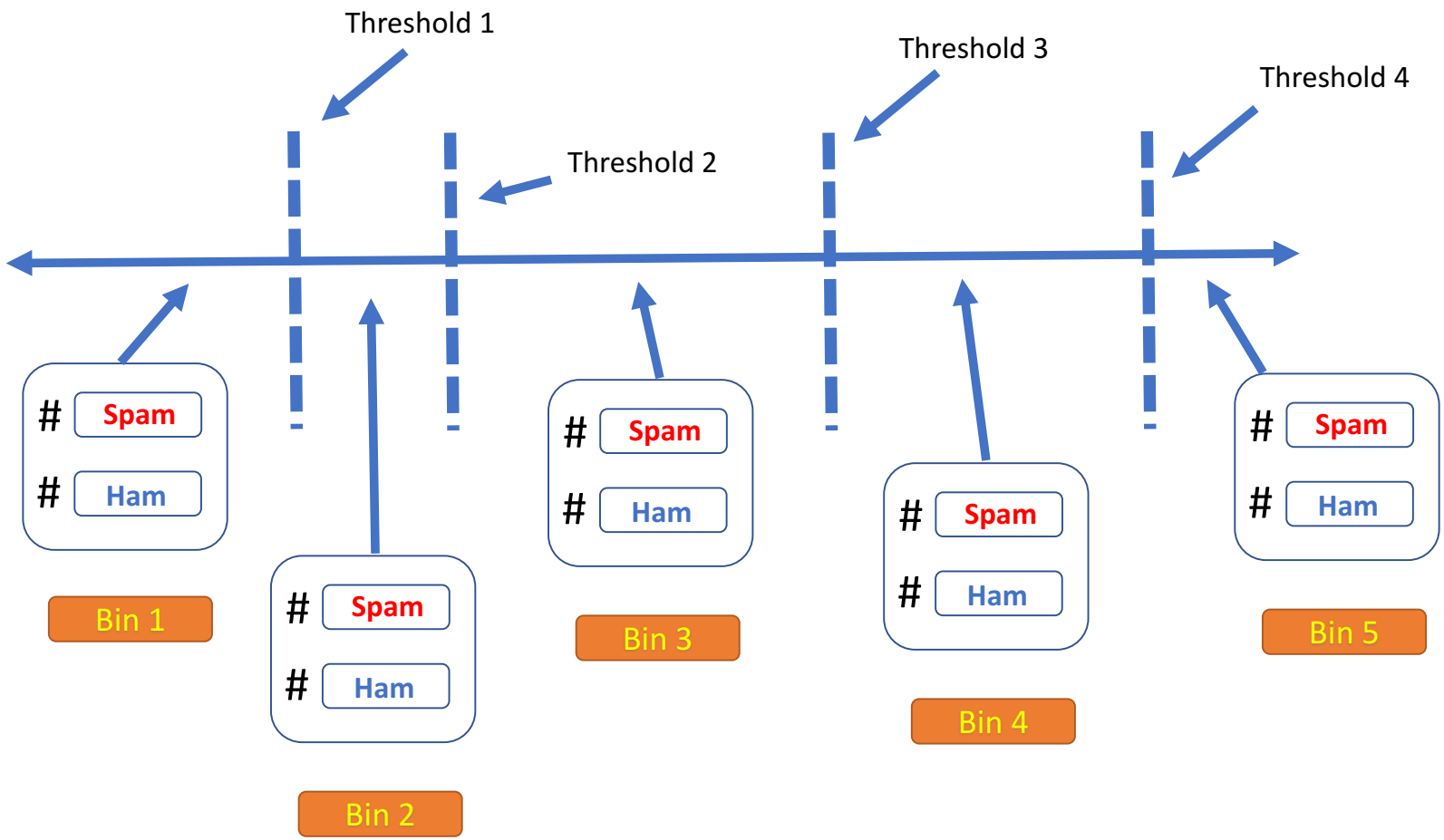
Building the tree in Spark

Bins for continuous features



Bins in continuous features

Bins for continuous features



Bins in continuous features

Contents

Contents

Gini index

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

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

- 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.

The vegetation classification dataset

(a) chaparral veg.

(b) riparian veg.

(c) conifer veg.

The vegetation classification dataset.

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

Gini index for the target feature

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

Information gain using the Gini index

Partition sets (Part.), entropy, Gini index, remainder (Rem.), and information gain (Info. Gain) by feature

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		

Information gain ratio

- Entropy based information gain has a preference for 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:

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

Contents

Why overfitting?

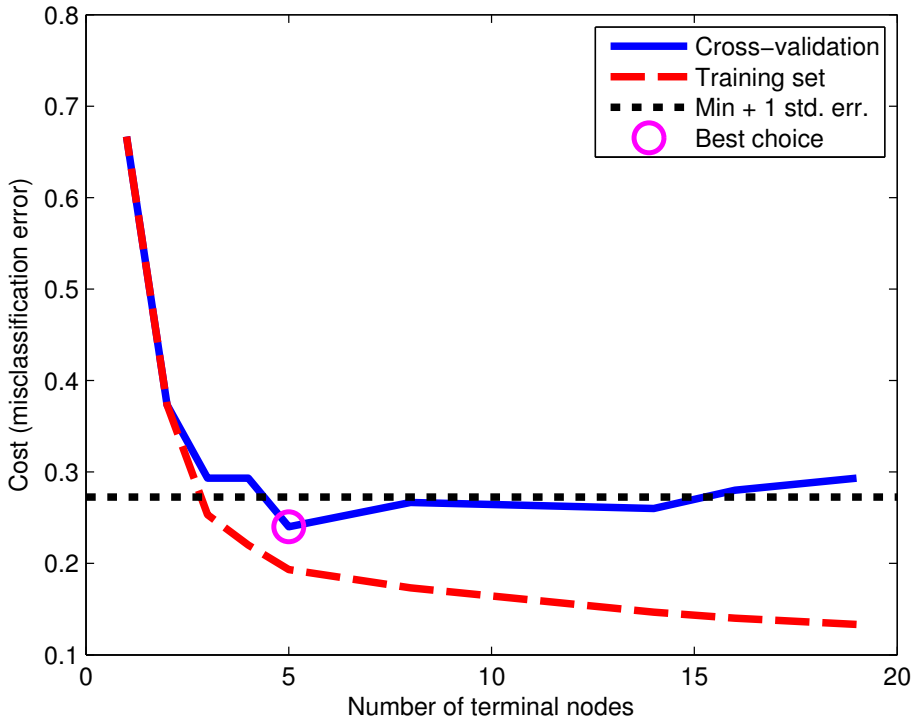
The likelihood of over-fitting occurring increases as a tree gets deeper because the resulting classifications are based on smaller and smaller subsets as the dataset is partitioned after each feature test in the path.

Pruning strategies

- **Pre-pruning**: stop the recursive partitioning early. Pre-pruning is also known as **forward pruning**.
- We can stop growing the tree if the decrease in the error is not sufficient to justify the extra complexity of adding an extra subtree.
- **Post-pruning**: allow the algorithm to grow the tree as much as it likes and then prune the tree of the branches that cause over-fitting.

Common Post-pruning Approach

- Using the validation set evaluate the prediction accuracy achieved by both the fully grown tree and the pruned copy of the tree.
- If the pruned copy of the tree performs no worse than the fully grown tree the node is a candidate for pruning.



Contents

Definition

- Rather than creating a single model they generate a set of models and then make predictions by aggregating the outputs of these models.
- A prediction model that is composed of a set of models is called a **model ensemble**.
- In order for this approach to work the models that are in the ensemble must be different from each other.

Approaches

There are two standard approaches to creating ensembles:

1. **boosting**
2. **bagging**.

Boosting: How does it work?

- 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**

Boosting: 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.

Algorithm

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:

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

3. Decreases the weights for the instances correctly classified:

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

4. Calculate a **confidence factor**, α, for the model such that α increases as ϵ decreases:

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

Predictions

- 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.

Bagging: Definition

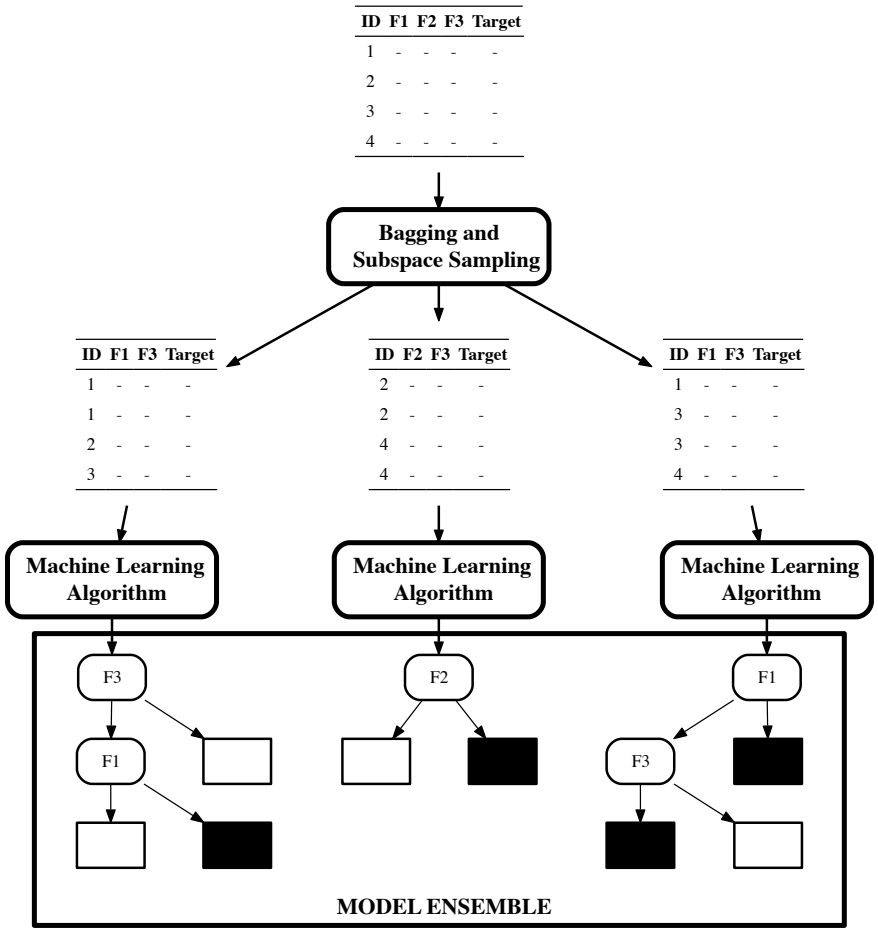
- When we use **bagging** (or **bootstrap aggregating**) each model in the ensemble is trained on a random sample of the dataset known as **bootstrap samples**.
- Each random sample is the same size as the dataset and **sampling with replacement** is used.
- Consequently, every bootstrap sample will 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

Bagging: Random forest

- 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.

Example of using bagging



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

Empirical results of boosting and bagging

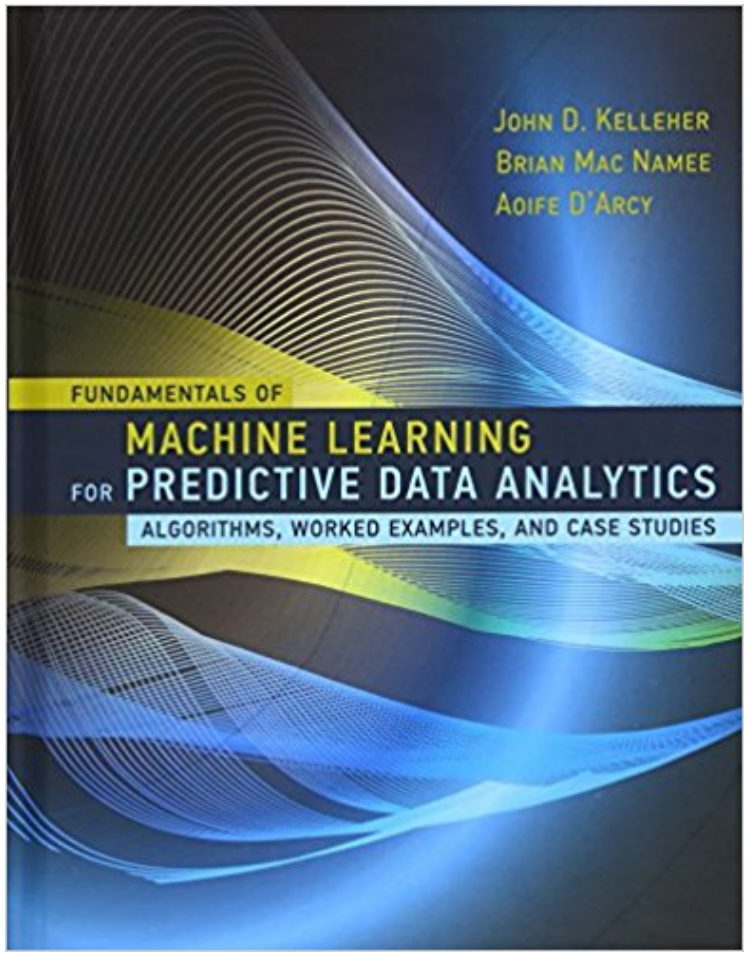
- Which approach should we use? Bagging is simpler to implement and parallelize than boosting and, so, may be better with respect to ease of use and training time.

- 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.

Contents

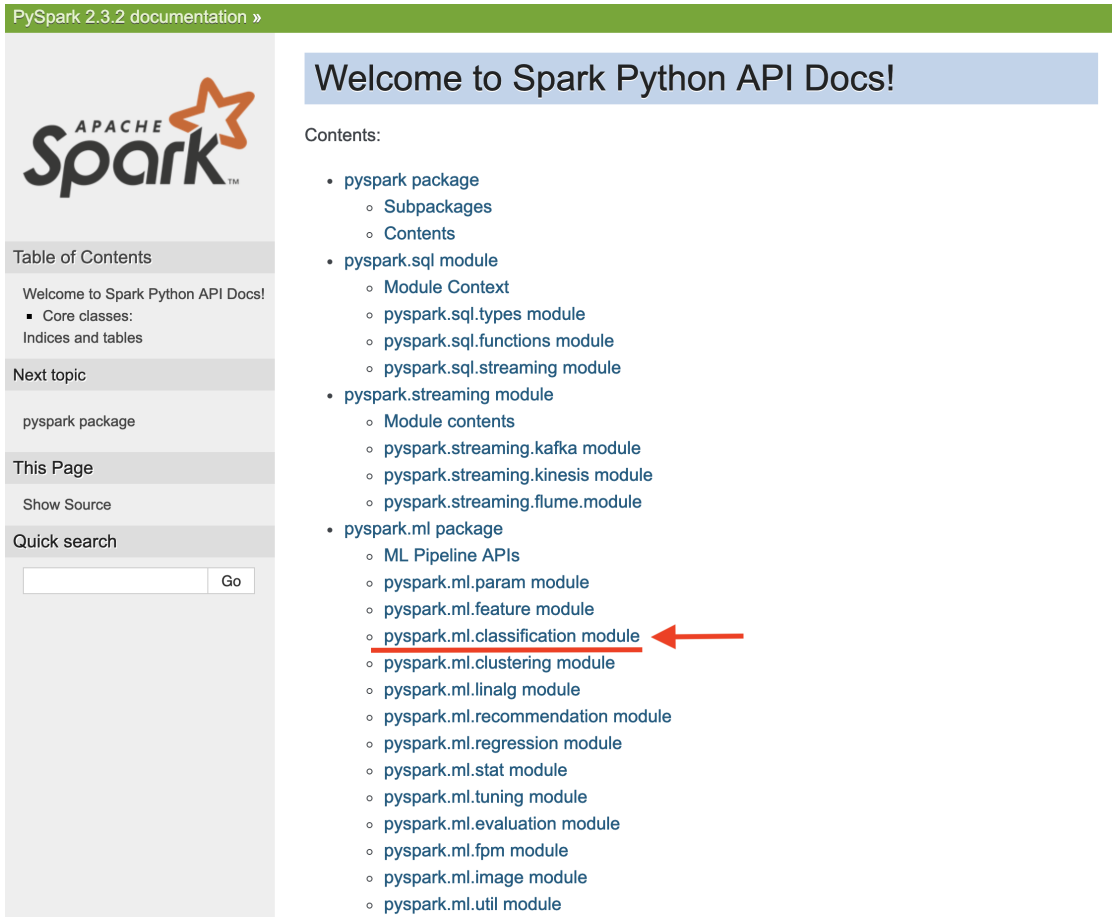
References used in this lecture

Book: *Fundamentals of Machine Learning for Predictive Analytics* by Kelleher, Mac Namee and D'Arcy, 2015.



References used in this lecture

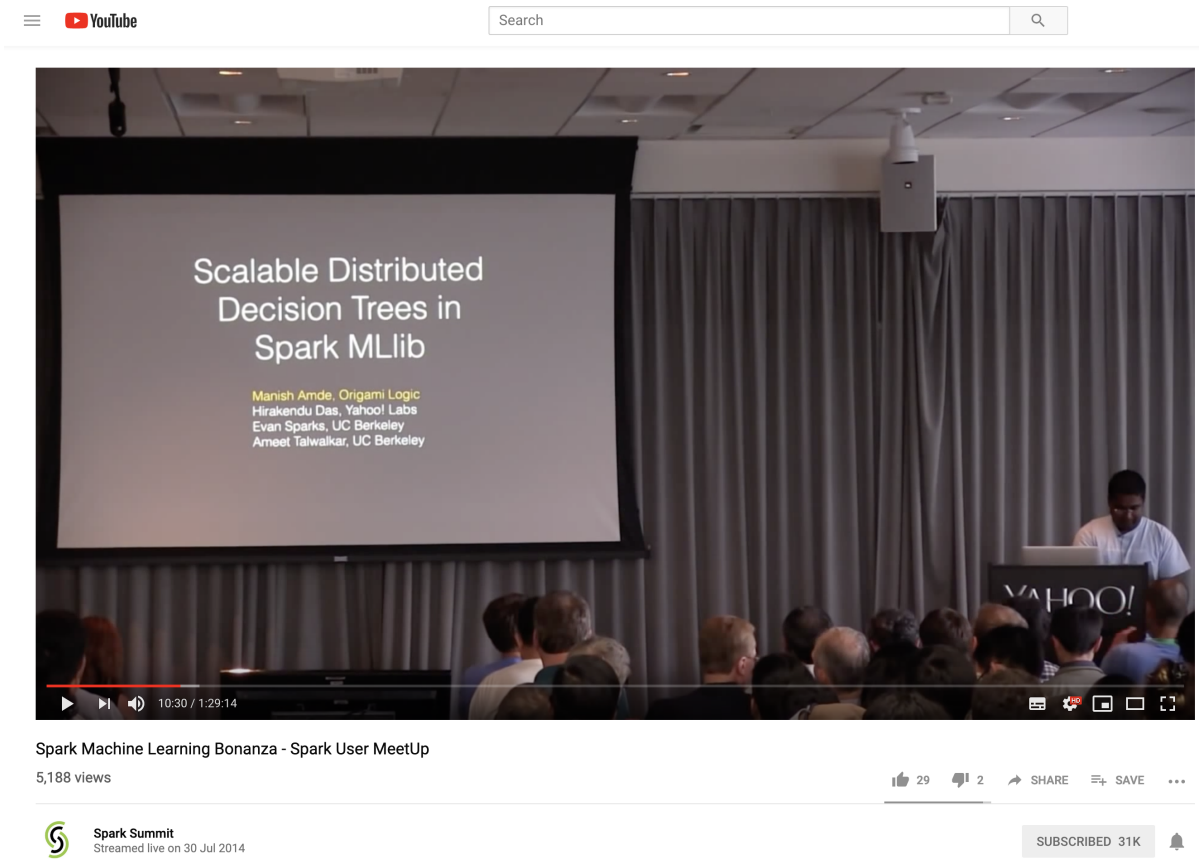
Website: *Spark Python API Documentation*.



`https://spark.apache.org/docs/2.3.2/api/python/index.html`

References used in this lecture

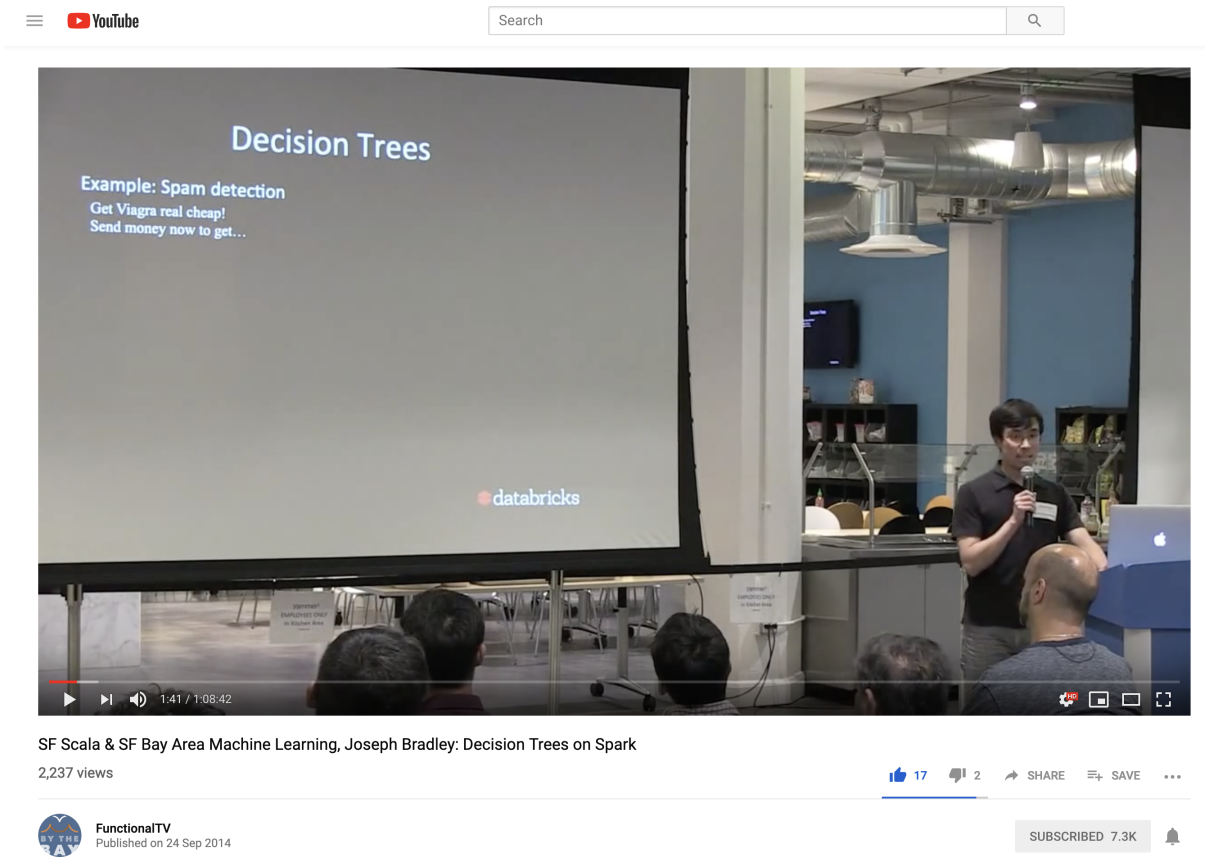
Youtube video: *Scalable Decision Trees in Spark MLlib* by Manish Amde (contributor to the MLlib implementation of Decision Trees in Spark).



`https://www.youtube.com/watch?v=N453EV5gHRA&t=10m30s`

References used in this lecture

Youtube video: *Decision Trees on Spark* by Joseph Bradley (from databricks).



`https://www.youtube.com/watch?v=3WS9OK3EXVA`

References used in this lecture

Paper: *PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce* by B. Panda et al. (from Google).

PLANET: Massively Parallel Learning of Tree Ensembles with MapReduce

Biswanath Panda, Joshua S. Herbach, Sugato Basu, Roberto J. Bayardo
Google, Inc.
[bpanda, jsherbach, sugato]@google.com, bayardo@alum.mit.edu

ABSTRACT

Classification and regression tree learning on massive datasets is a common data mining task at Google, yet many state of the art tree learning algorithms require training data to reside in memory on a single machine. While more scalable implementations of tree learning have been proposed, they typically require specialized parallel computing architectures. In contrast, the majority of Google's computing infrastructure is based on commodity hardware.

In this paper, we describe PLANET: a scalable distributed framework for learning tree models over large datasets. PLANET defines tree learning as a series of distributed computations, and implements each one using the *MapReduce* model of distributed computation. We show how this framework supports scalable construction of classification and regression trees, as well as ensembles of such models. We discuss the benefits and challenges of using a MapReduce compute cluster for tree learning, and demonstrate the scalability of this approach by applying it to a real world learning task from the domain of computational advertising.

plexities such as data partitioning, scheduling tasks across many machines, handling machine failures, and performing inter-machine communication. These properties have motivated many technology companies to run MapReduce frameworks on their compute clusters for data analysis and other data management tasks. MapReduce has become in some sense an industry standard. For example, there are open source implementations such as Hadoop that can be run either in-house or on cloud computing services such as Amazon EC2.[1] Startups like Cloudera[2] offer software and services to simplify Hadoop deployment, and companies including Google, IBM and Yahoo! have granted several universities access to Hadoop clusters to further cluster computing research.[3]

Despite the growing popularity of MapReduce [12], its application to certain standard data mining and machine learning tasks remains poorly understood. In this paper we focus on one such task: tree learning. We believe that a tree learner capable of exploiting a MapReduce cluster can effectively address many scalability issues that arise in building tree models on massive datasets. Our choice of focusing