Fundamentals of Machine Learning

Chapter 4: Information-based Learning

Sections 4.4, 4.5

Alternative Feature Selection Metrics

- 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 denominator above is simply H(d, D), that is, entropy of feature d in the D data partition.

$$
\begin{aligned}
IG(\text{STREAM}, \mathcal{D}) &= 0.3060 \\
IG(\text{SLOPE}, \mathcal{D}) &= 0.5774 \\
IG(\text{ELEVATION}, \mathcal{D}) &= 0.8774
\end{aligned}
$$

$$
\begin{aligned}
&H(\textsc{Stream}, \mathcal{D}) \\
&= - \sum_{I \in \left\{ \substack{\text{'true'}, \\ \text{'false'}} \right\}} P(\textsc{Stream} = I) \times log_2 \left(P(\textsc{Stream} = I) \right) \\
&= - \left(\left({}^4/_7 \times log_2({}^4/_7) \right) + \left({}^3/_7 \times log_2({}^3/_7) \right) \right) \\
&= 0.9852 \textit{ bits}
\end{aligned}
$$

$$
\begin{aligned}
&H(\textsc{Slope}, \mathcal{D}) \\
&= - \sum_{I \in \left\{ \substack{\text{'flat'}, \\ \text{'moderate'}, \\ \text{'steep'}} \right\}} P(\textsc{Slope} = I) \times log_2 \left(P(\textsc{Slope} = I) \right) \\
&= - \left(\left({}^1/_7 \times log_2({}^1/_7) \right) + \left({}^1/_7 \times log_2({}^1/_7) \right) + \left({}^5/_7 \times log_2({}^5/_7) \right) \right) \\
&= 1.1488 \textit{ bits}
\end{aligned}
$$

$$
\begin{aligned}
&H(\textsc{Elevation}, \mathcal{D}) \\
&= - \sum_{I \in \left\{ \substack{\text{'low'}, \\ \text{'medium'}, \\ \text{'high'}, \\ \text{'highest'}} \right\}} P(\textsc{Elevation} = I) \times log_2 \left(P(\textsc{Elevation} = I) \right) \\
&= - \left(\left({}^1/_7 \times log_2({}^1/_7) \right) + \left({}^2/_7 \times log_2({}^2/_7) \right) + \left({}^3/_7 \times log_2({}^3/_7) \right) + \left({}^1/_7 \times log_2({}^1/_7) \right) \right) \\
&= 1.8424 \textit{ bits}
\end{aligned}
$$

$$GR(\text{STREAM}, \mathcal{D}) = \frac{0.3060}{0.9852} = 0.3106$$

$$GR(\text{SLOPE}, \mathcal{D}) = \frac{0.5774}{1.1488} = 0.5026$$

$$GR(\text{ELEVATION}, \mathcal{D}) = \frac{0.8774}{1.8424} = 0.4762$$

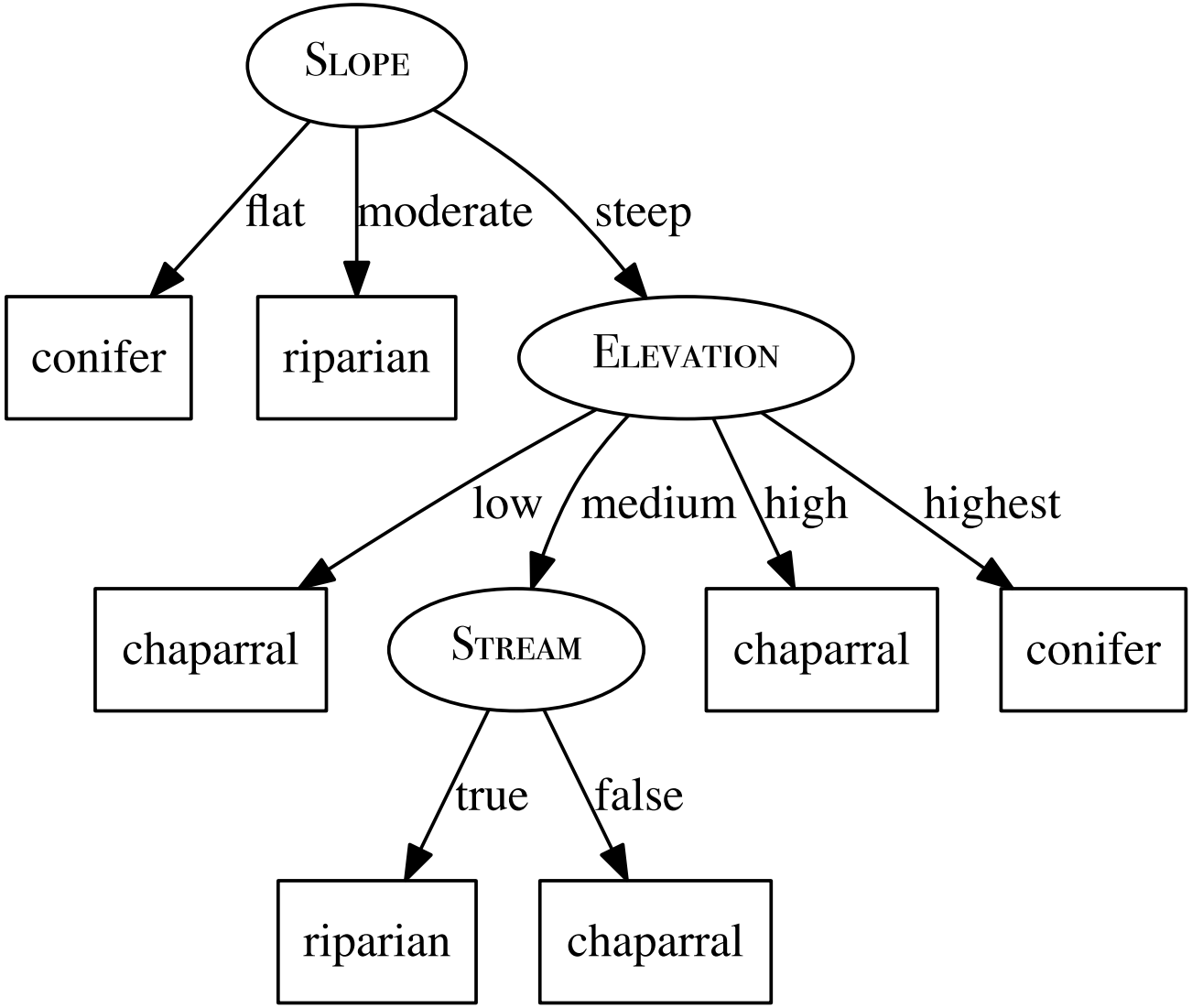


Figure: The vegetation classification decision tree generated using information gain ratio.

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

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

- 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.

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

Table: 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		

Handling Continuous Descriptive Features

- The easiest way to handle continuous valued descriptive features is to turn them into boolean features by defining a threshold and using this threshold to partition the instances based their value of the continuous descriptive feature.
- How do we set the threshold?

1. The instances in the dataset are sorted according to the continuous feature values.
2. The adjacent instances in the ordering that have different classifications are then selected as possible threshold points.
3. The optimal threshold is found by computing the information gain for each of these classification transition boundaries and selecting the boundary with the highest information gain as the threshold.

- Once a threshold has been set the dynamically created new boolean feature can compete with the other categorical features for selection as the splitting feature at that node.
- This process can be repeated at each node as the tree grows.

Table: Dataset for predicting the vegetation in an area with a continuous ELEVATION feature (measured in feet).

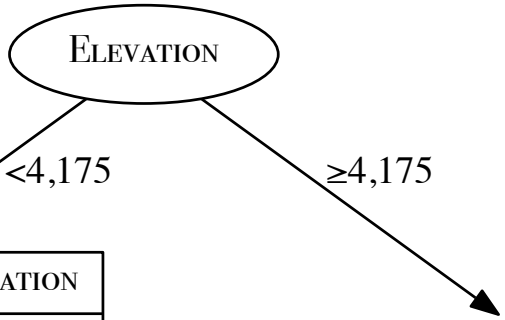
ID	STREAM	SLOPE	ELEVATION	VEGETATION
1	false	steep	3 900	chapparal
2	true	moderate	300	riparian
3	true	steep	1 500	riparian
4	false	steep	1 200	chapparal
5	false	flat	4 450	conifer
6	true	steep	5 000	conifer
7	true	steep	3 000	chapparal

Table: Dataset for predicting the vegetation in an area sorted by the continuous ELEVATION feature.

ID	STREAM	SLOPE	ELEVATION	VEGETATION
2	true	moderate	300	riparian
4	false	steep	1 200	chapparal
3	true	steep	1 500	riparian
7	true	steep	3 000	chapparal
1	false	steep	3 900	chapparal
5	false	flat	4 450	conifer
6	true	steep	5 000	conifer

Table: Partition sets (Part.), entropy, remainder (Rem.), and information gain (Info. Gain) for the candidate ELEVATION thresholds: $\geq$750, $\geq$1 350, $\geq$2 250 and $\geq$4 175.

Split by Threshold	Part.	Instances	Partition Entropy	Rem.	Info. Gain
$\geq$750	$\mathcal{D}_1$	$\mathbf{d}_2$	0.0	1.2507	0.3060
	$\mathcal{D}_2$	$\mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.4591		
$\geq$1 350	$\mathcal{D}_3$	$\mathbf{d}_2, \mathbf{d}_4$	1.0	1.3728	0.1839
	$\mathcal{D}_4$	$\mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.5219		
$\geq$2 250	$\mathcal{D}_5$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3$	0.9183	0.9650	0.5917
	$\mathcal{D}_6$	$\mathbf{d}_7, \mathbf{d}_1, \mathbf{d}_5, \mathbf{d}_6$	1.0		
$\geq$4 175	$\mathcal{D}_7$	$\mathbf{d}_2, \mathbf{d}_4, \mathbf{d}_3, \mathbf{d}_7, \mathbf{d}_1$	0.9710	0.6935	0.8631
	$\mathcal{D}_8$	$\mathbf{d}_5, \mathbf{d}_6$	0.0		



	ID	Stream	Slope	Elevation	Vegetation
D7	2	true	moderate	300	riparian
	4	false	steep	1,200	chaparral
	3	true	steep	1,500	riparian
	7	true	steep	3,000	chaparral
	1	false	steep	3,900	chaparral

	ID	Stream	Slope	Elevation	Vegetation
D8	5	false	flat	4,450	conifer
	6	true	steep	5,000	conifer

Figure: The vegetation classification decision tree after the dataset has been split using ELEVATION $\geq$ 4 175.

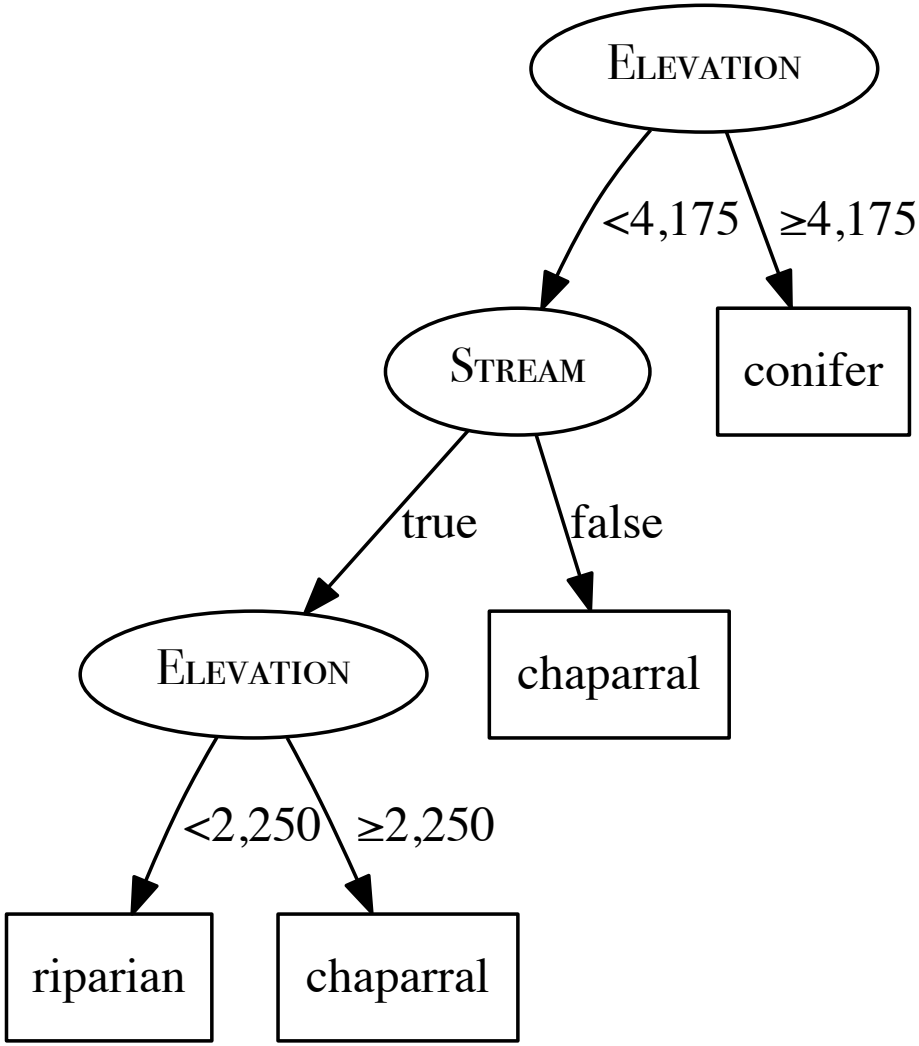


Figure: The decision tree that would be generated for the vegetation classification dataset listed in Table 3 [17] using information gain.

Predicting Continuous Targets

- 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

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

$$var\left(t, \mathcal{D}\right) = \frac{\sum_{i=1}^{n}\left(t_i - \bar{t}\right)^2}{n-1} \tag{3}$$

- We select the feature to split on at a node by selecting the feature that minimizes the weighted variance across the resulting partitions:

$$\mathbf{d}[best] = \underset{d \in \mathbf{d}}{\text{argmin}} \sum_{l \in levels(d)} \frac{|\mathcal{D}_{d=l}|}{|\mathcal{D}|} \times var(t, \mathcal{D}_{d=l}) \tag{4}$$

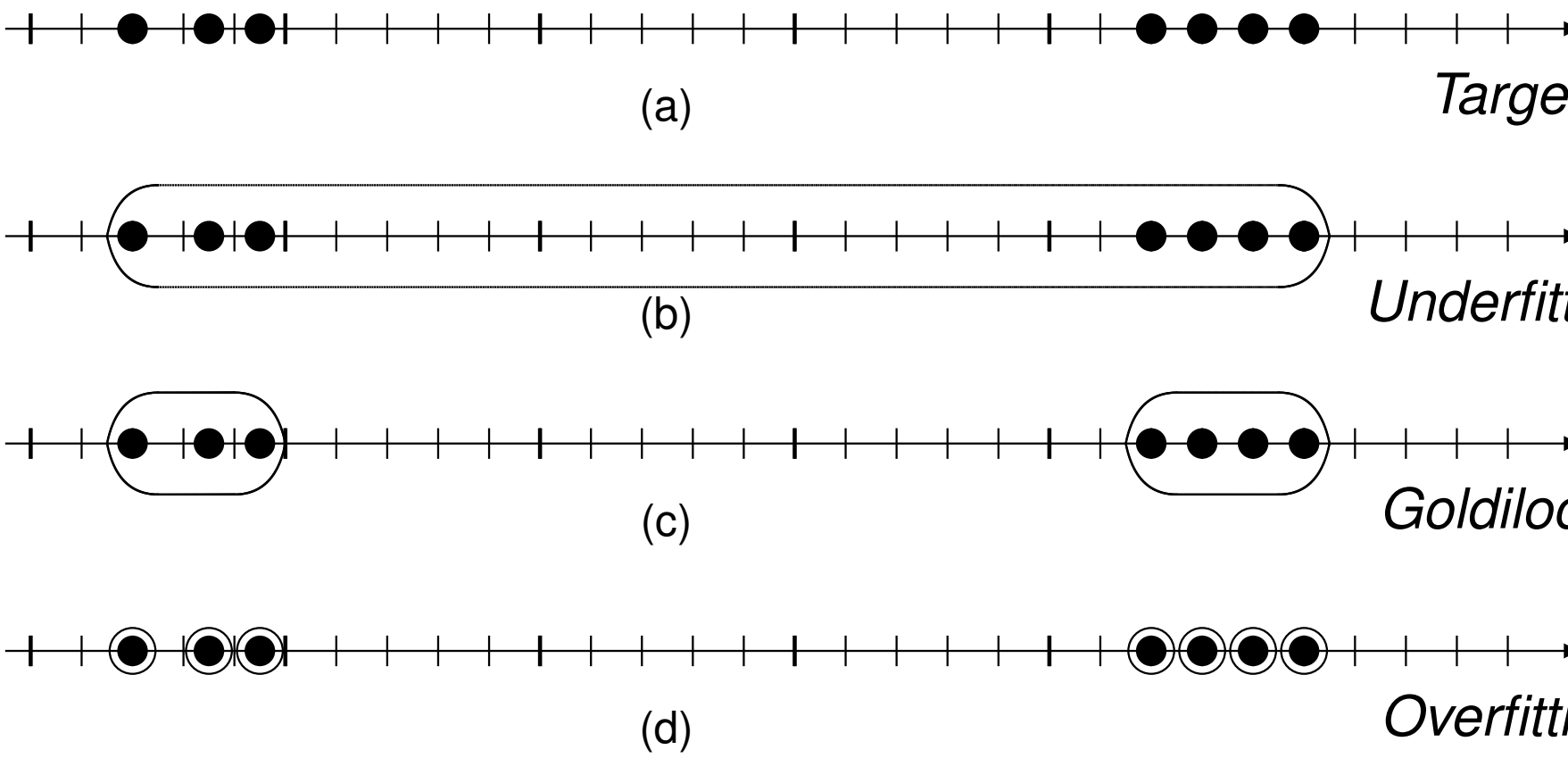


Figure: (a) A set of instances on a continuous number line; (b), (c), and (d) depict some of the potential groupings that could be applied to these instances.

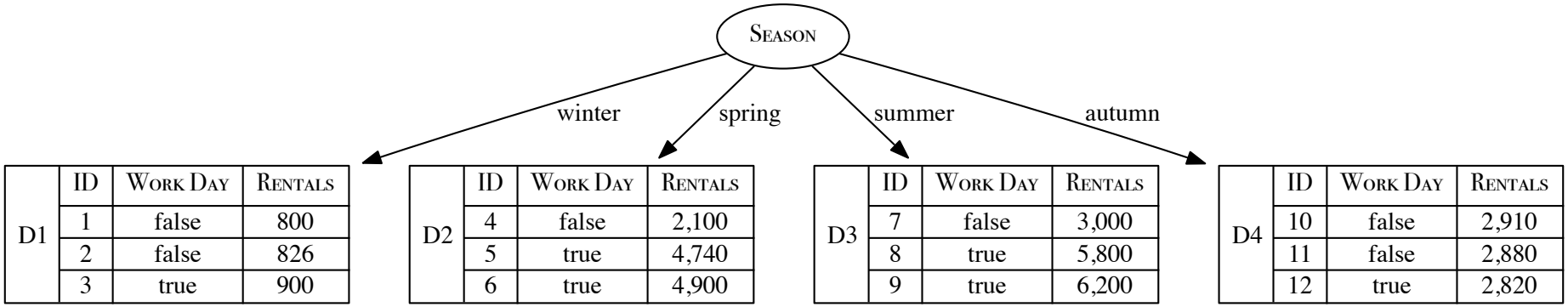
Table: A dataset listing the number of bike rentals per day.

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

Table: The partitioning of the dataset in Table 5 [25] 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	

How to calculate the first variance value 2692 in Python:
numpy.array([800, 826, 900]).var(ddof=1).round(0)



	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

Figure: The decision tree resulting from splitting the data in Table 5 [25] using the feature SEASON.

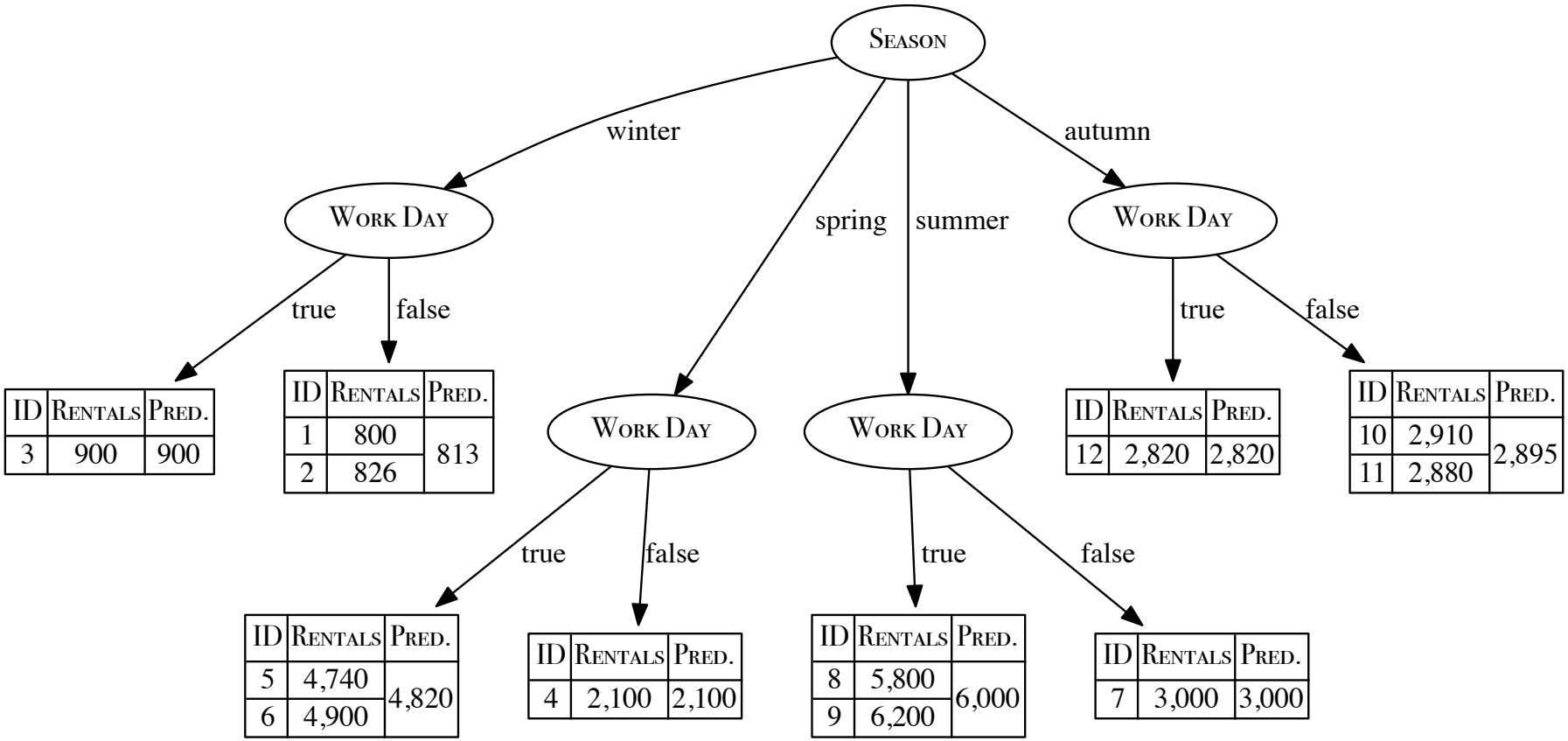


Figure: The final decision tree induced from the dataset in Table 5 [25]. To illustrate how the tree generates predictions, this tree lists the instances that ended up at each leaf node and the prediction (PRED.) made by each leaf node.

Noisy Data, Overfitting and Tree Pruning

- In the case of a decision tree, over-fitting involves splitting the data on an irrelevant feature.

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.

- **Pre-pruning**: stop the recursive partitioning early. Pre-pruning is also known as **forward pruning**.

Common Pre-pruning Approaches

1. **early stopping**
2. **χ^2 pruning**

- **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.

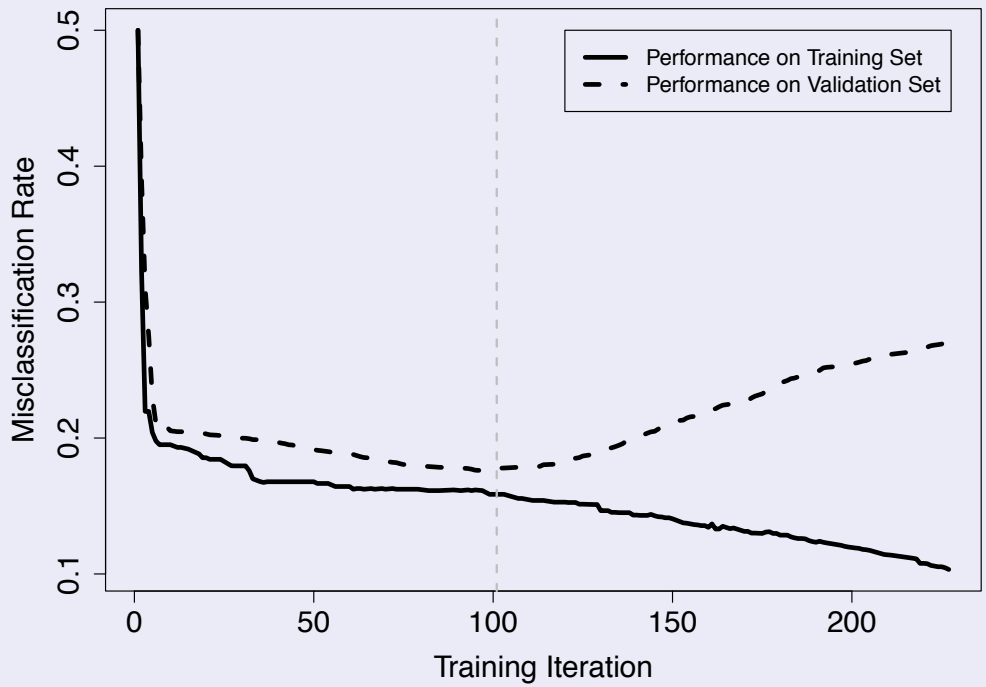


Table: An example validation set for the post-operative patient routing task.

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

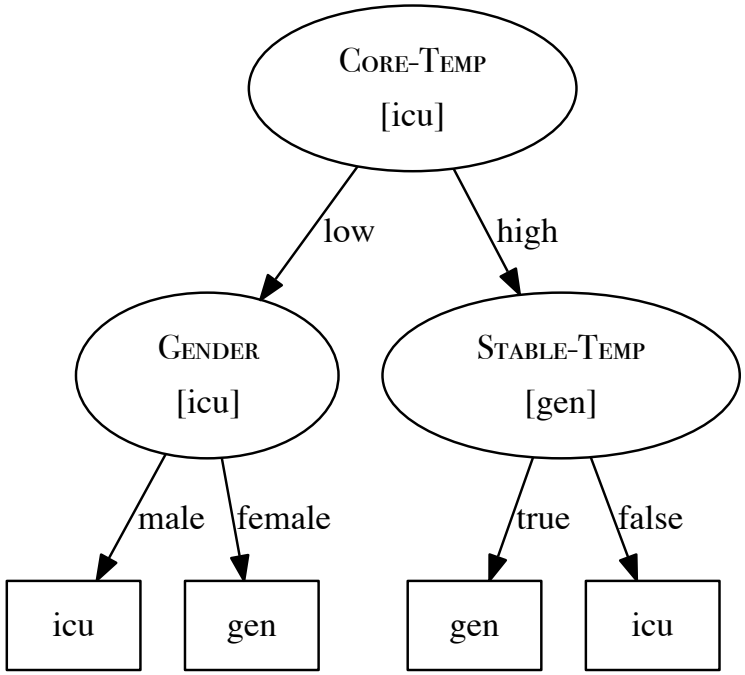


Figure: The decision tree for the post-operative patient routing task.

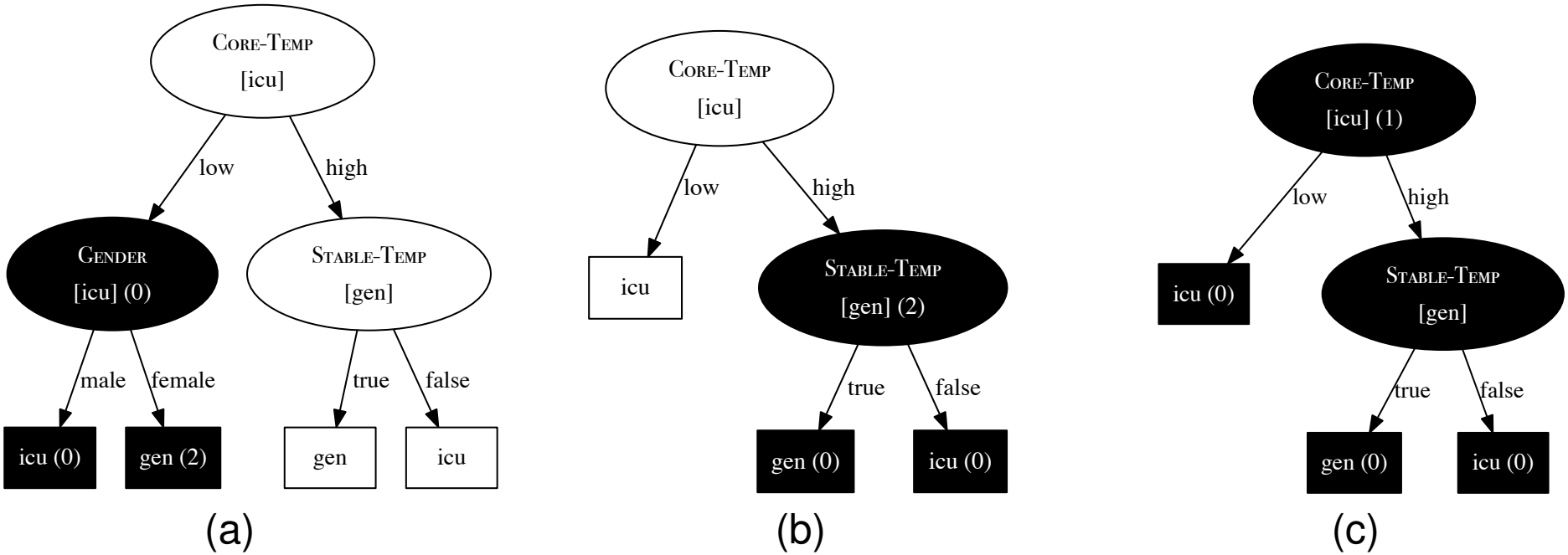


Figure: The iterations of reduced error pruning for the decision tree in Figure 7 [34] using the validation set in Table 7 [33]. The subtree that is being considered for pruning in each iteration is highlighted in black. The prediction returned by each non-leaf node is listed in square brackets. The error rate for each node is given in round brackets.

Advantages of pruning:

- Smaller trees are easier to interpret
- Increased generalization accuracy when there is noise in the training data (**noise dampening**).

Model Ensembles

- 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.

How to calculate the probability of a model ensemble that uses simple majority vote making an incorrect prediction in the following two scenarios using the binomial distribution:

The ensemble contains 3 independent models, all of which have an error rate of 40%.

$$\Pr(\text{incorrect prediction}) = \binom{3}{2}(0.4)^2(0.6)^1 + \binom{3}{3}(0.4)^3(0.6)^0$$
$$= 3*0.16*0.6 + 1*0.064$$
$$= 0.0288 + 0.064$$
$$= \mathbf{0.35}$$

Alternatively, compute directly: (A: Accurate prediction, N: Inaccurate prediction)

$$\Pr(\text{incorrect prediction}) = \Pr(\text{NNA}) + \Pr(\text{ANN}) + \Pr(\text{NAN}) + \Pr(\text{NNN})$$
$$= 3*0.16*0.6 + 0.064$$
$$= 0.35$$

The ensemble contains 5 independent models, all of which have an error rate of 40%.

$$\Pr(\text{incorrect prediction}) = \binom{5}{3}(0.4)^3(0.6)^2 + \binom{5}{4}(0.4)^4(0.6)^1 + \binom{5}{5}(0.4)^5(0.6)^0$$
$$= 10*0.064*0.36 + 5*0.0256*0.6 + 1*0.01$$
$$= 0.02304 + 0.0768 + 0.01024$$
$$= \mathbf{0.32}$$

Based on the above calculations, the ensemble with 5 independent models would be selected because it gives a lower probability for an incorrect prediction.

- There are two standard approaches to creating ensembles:
  1. **boosting**
  2. **bagging**.

- 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:

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

3. Decreases the weights for the instances correctly classified:

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

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

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

- 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.

- 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.

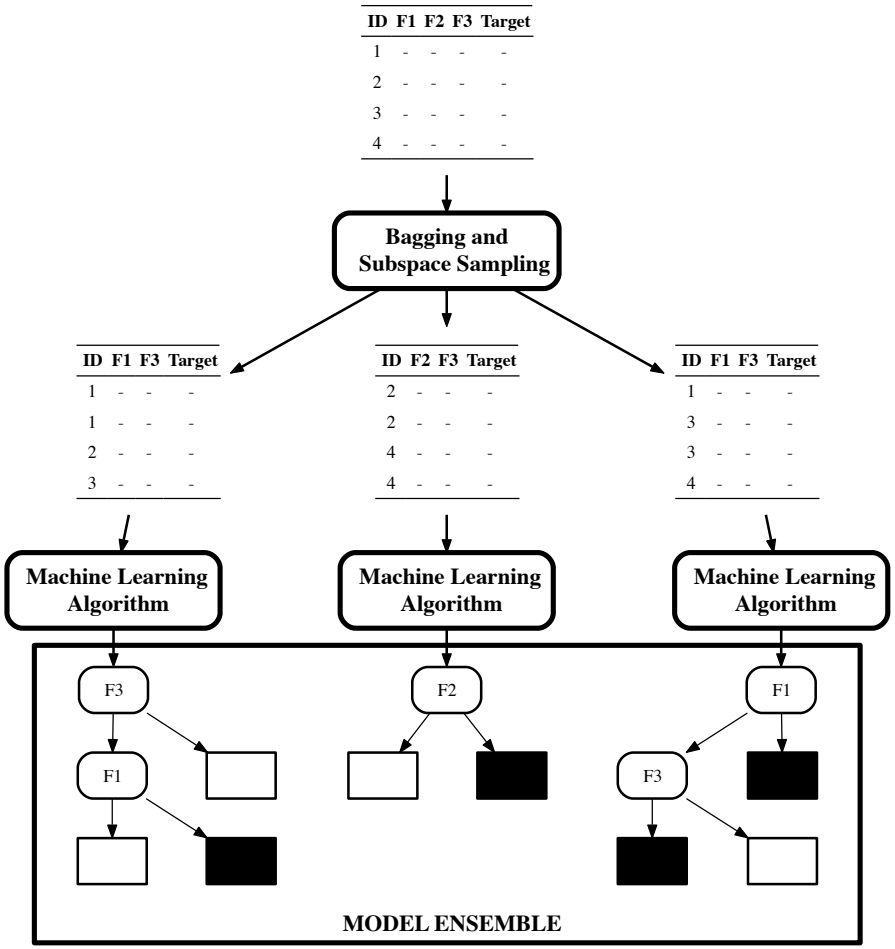


Figure: The process of creating a model ensemble using bagging and subspace sampling.

- 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.

Summary

- The **decision tree** model makes predictions based on sequences of tests on the descriptive feature values of a query
- The **ID3** algorithm as a standard algorithm for inducing decision trees from a dataset.

Decision Trees: Advantages

- interpretable.
- handle both categorical and continuous descriptive features.
- has the ability to model the interactions between descriptive features (diminished if **pre-pruning** is employed)
- relatively, robust to the **curse of dimensionality**.
- relatively, robust to noise in the dataset if **pruning** is used.

Decision Tress: Potential Disadvantages

- trees become large when dealing with continuous features.
- decision trees are very expressive and sensitive to the dataset, as a result they can overfit the data if there are a lot of features (curse of dimensionality)
- eager learner (concept drift).

1. Alternative Feature Selection Metrics
2. Handling Continuous Descriptive Features
3. Predicting Continuous Targets
4. Noisy Data, Overfitting and Tree Pruning
5. Model Ensembles
   - Boosting
   - Bagging
6. Summary