Chapter 5: Decision Tree Learning

Reza Rezazadegan

Sharif University of Technology

Fall 2022

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$.

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$. As usual $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ are the features of the data.

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$. As usual $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ are the features of the data.
- At the root of the tree is the whole training dataset . At each other node v we have a subset $S_v \subset D$.

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$. As usual $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ are the features of the data.
- At the root of the tree is the whole training dataset . At each other node v we have a subset $S_v \subset D$.
- Each edge represents a "decision" or "rule" on the value of a feature e.g. "$x_k > 0.24$".

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$. As usual $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ are the features of the data.
- At the root of the tree is the whole training dataset . At each other node v we have a subset $S_v \subset D$.
- Each edge represents a "decision" or "rule" on the value of a feature e.g. "$x_k > 0.24$".
- If $u \to v$ is an edge (i.e. u is the parent of v) with a decision such as $x_k < c$ then $S_v = \{\mathbf{x} \in S_u : x_k < c\}$.

Introducing Decision Tree Learning

- In Decision Tree Learning, a decision tree is learned which gives a classification of our dataset $D = \{(\mathbf{x}_i, y_i)\}$. As usual $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ are the features of the data.
- At the root of the tree is the whole training dataset . At each other node v we have a subset $S_v \subset D$.
- Each edge represents a "decision" or "rule" on the value of a feature e.g. "$x_k > 0.24$".
- If $u \to v$ is an edge (i.e. u is the parent of v) with a decision such as $x_k < c$ then $S_v = \{\mathbf{x} \in S_u : x_k < c\}$.
- Decision Tree Learning algorithms such as ID3 or CART use a measure of impurity (such as *entropy* or *Gini impurity*) and at each node, try to find a feature-condition pair that reduces impurity most.
- Such algorithms try to find a tree whose leaves S_v are pure, i.e. contain only one class.

Example of a Decision Tree

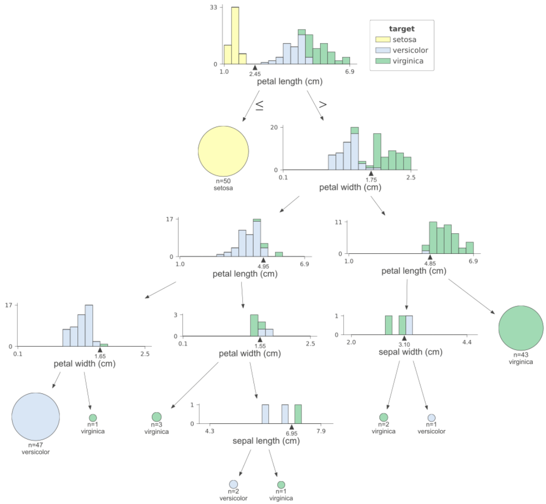


Figure: A decision tree for the iris dataset depicted using `dtreeviz` library.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.
- $H(P) = 0$ if and only if $p(x_0) = 1$ for some x_0.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.
- $H(P) = 0$ if and only if $p(x_0) = 1$ for some x_0.
- Theorem: Among distributions on the same sample space, maximum entropy belongs to the uniform distribution.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.
- $H(P) = 0$ if and only if $p(x_0) = 1$ for some x_0.
- Theorem: Among distributions on the same sample space, maximum entropy belongs to the uniform distribution.
- Thus entropy is a measure of uncertainty of a probability distribution or random variable.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.
- $H(P) = 0$ if and only if $p(x_0) = 1$ for some x_0.
- Theorem: Among distributions on the same sample space, maximum entropy belongs to the uniform distribution.
- Thus entropy is a measure of uncertainty of a probability distribution or random variable.
- Shannon's source coding theorem: a random variable X with probability distribution p can not be compressed into more than $H(p)$ bits of information.

Measures of Impurity: Entropy

- Remember that the **Shannon Entropy** of a probability distribution p is given by $H(p) = -E[\log_2 p] = -\sum p(x) \log_2 p(x)$.
- Idea: when $p(x)$ is close to 1 then the surprise of the event x is low; if its close to 0 then the surprise is high. $-\log(p(x)) = \log(1/p(x))$.
- $H(P) = 0$ if and only if $p(x_0) = 1$ for some x_0.
- Theorem: Among distributions on the same sample space, maximum entropy belongs to the uniform distribution.
- Thus entropy is a measure of uncertainty of a probability distribution or random variable.
- Shannon's source coding theorem: a random variable X with probability distribution p can not be compressed into more than $H(p)$ bits of information.
- In case we have a set S containing elements belonging to m different classes, and $p(k)$ is the probability of belonging to class k then $H(p)$ is a measure of "mixedness" (or impurity) of S in terms of classes.

Entropy cont.

- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.

Entropy cont.

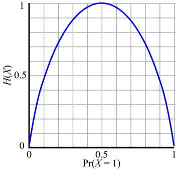
- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.

Entropy cont.

- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.

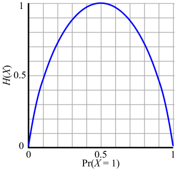
Entropy cont.

- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.



Entropy cont.

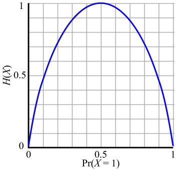
- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.



- A fair coin is more surprising than a biased coin!

Entropy cont.

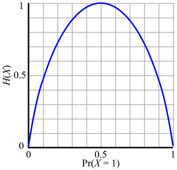
- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.



- A fair coin is more surprising than a biased coin!
- We can interpret this probability distribution as coming from a mix of two classes $y=0, y=1$.

Entropy cont.

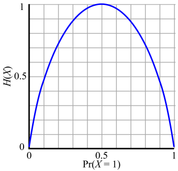
- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.



- A fair coin is more surprising than a biased coin!
- We can interpret this probability distribution as coming from a mix of two classes $y=0, y=1$.
- More generally if S is a set of N elements, partitioned into classes $C_1, C_2 \ldots C_m$ then it gives us the probability distribution $p(C_k)=|C_k|/|S|$ and we can compute its entropy:

Entropy cont.

- For example consider the Bernoulli distribution: $p(y=1)=p, p(y=0)=1-p$. Its entropy is given by $-p\log(p)-(1-p)\log(1-p)$.
- If $p=0$ or $p=1$ then $H=0$.If $p=1/2$ then $H=1$.



- A fair coin is more surprising than a biased coin!
- We can interpret this probability distribution as coming from a mix of two classes $y=0, y=1$.
- More generally if S is a set of N elements, partitioned into classes $C_1, C_2 \dots C_m$ then it gives us the probability distribution $p(C_k)=|C_k|/|S|$ and we can compute its entropy: $H(S)=\sum_i \frac{|C_i|}{N}(\log(N)-\log(|C_i|)$.

Information Gain and the ID3 (Iterative Dichotomiser 3) algorithm

- At each node v of a decision tree, we want to split S_v in such a way that the impurity (entropy in this case) of the children is much smaller than that of v.

Information Gain and the ID3 (Iterative Dichotomiser 3) algorithm

- At each node v of a decision tree, we want to split S_v in such a way that the impurity (entropy in this case) of the children is much smaller than that of v.
- To this end, we use the **Information Gain (IG)** of parent and children.

Information Gain and the ID3 (Iterative Dichotomiser 3) algorithm

- At each node v of a decision tree, we want to split S_v in such a way that the impurity (entropy in this case) of the children is much smaller than that of v.
- To this end, we use the **Information Gain (IG)** of parent and children. If $u_1, u_2, \ldots, u_k$ are the children of v and we assign subsets $S_{u_i} \subset S_v$ to them which make a partition of S_v then

$$IG(v, \{u_1, \ldots, u_k\}) = H(S_v) - \sum_i p(S_{u_i}) H(S_{u_i}). \tag{1}$$

  Here $p(S_{u_i}) = |S_{u_i}|/|S_v|$.

Information Gain and the ID3 (Iterative Dichotomiser 3) algorithm

- At each node v of a decision tree, we want to split S_v in such a way that the impurity (entropy in this case) of the children is much smaller than that of v.
- To this end, we use the **Information Gain (IG)** of parent and children. If $u_1, u_2, \ldots, u_k$ are the children of v and we assign subsets $S_{u_i} \subset S_v$ to them which make a partition of S_v then

$$IG(v, \{u_1, \ldots, u_k\}) = H(S_v) - \sum_i p(S_{u_i}) H(S_{u_i}). \tag{1}$$

  Here $p(S_{u_i}) = |S_{u_i}|/|S_v|$.
- Note that the base of logarithm is unimportant for computing IG, as long as it is the same throughout.

Information Gain and the ID3 (Iterative Dichotomiser 3) algorithm

- At each node v of a decision tree, we want to split S_v in such a way that the impurity (entropy in this case) of the children is much smaller than that of v.
- To this end, we use the **Information Gain (IG)** of parent and children. If $u_1, u_2, \ldots, u_k$ are the children of v and we assign subsets $S_{u_i} \subset S_v$ to them which make a partition of S_v then

$$IG(v, \{u_1, \ldots, u_k\}) = H(S_v) - \sum_i p(S_{u_i}) H(S_{u_i}). \tag{1}$$

  Here $p(S_{u_i}) = |S_{u_i}|/|S_v|$.
- Note that the base of logarithm is unimportant for computing IG, as long as it is the same throughout.
- At each node v, the algorithm finds the feature x_j such that splitting S_v according to the values of x_j gives the highest information gain.

Example of Information Gain

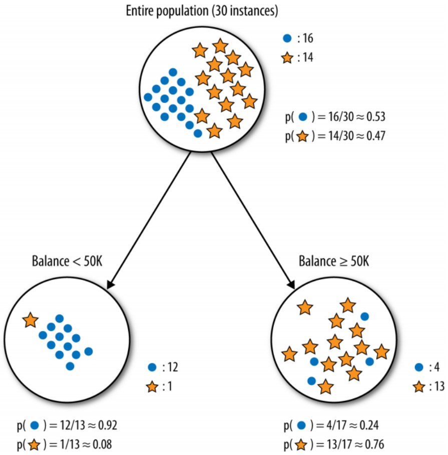


Figure: Compute the information gain for this splitting. Credit: Foster provost and Tom Fawcett

Another example of Information Gain

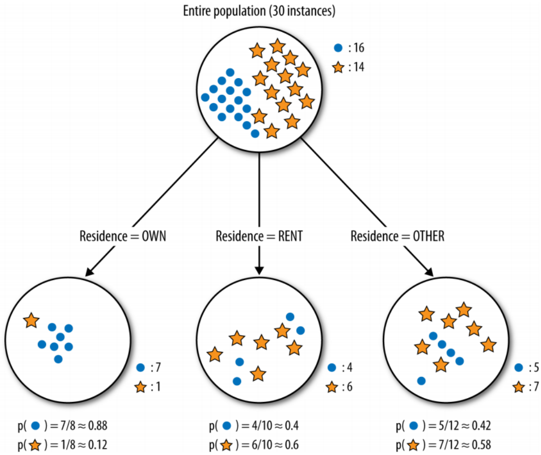


Figure: Another splitting of the same set. Credit: Foster Provost and Tom Fawcett

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \ldots k\}$

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \ldots k\}$ i.e. the children being $\{\mathbf{x} \in S_v : x_j = 1\}$, $\{\mathbf{x} \in S_v : x_j = 2\}$, etc.

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \dots k\}$ i.e. the children being $\{\mathbf{x} \in S_v : x_j = 1\}$, $\{\mathbf{x} \in S_v : x_j = 2\}$, etc.
- The original ID3 algorithm can only handle discrete values.

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \dots k\}$ i.e. the children being $\{\mathbf{x} \in S_v : x_j = 1\}$, $\{\mathbf{x} \in S_v : x_j = 2\}$, etc.
- The original ID3 algorithm can only handle discrete values. However the successor C4.5 Algorithm converts continuous features into discrete ones.

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \ldots k\}$ i.e. the children being $\{\mathbf{x} \in S_v : x_j = 1\}$, $\{\mathbf{x} \in S_v : x_j = 2\}$, etc.
- The original ID3 algorithm can only handle discrete values. However the successor C4.5 Algorithm converts continuous features into discrete ones.
- A continuous feature x_k can be discretized by dividing its range into intervals $[c_1, c_2), [c_2, c_3), \ldots, [c_{n-1}, c_n]$ and treating these intervals as discrete values.

ID3 and C4.5 Algorithms

- For a discrete-valued feature x_j, the splitting is done according to its possible values $\{1, 2, \dots k\}$ i.e. the children being $\{\mathbf{x} \in S_v : x_j = 1\}$, $\{\mathbf{x} \in S_v : x_j = 2\}$, etc.
- The original ID3 algorithm can only handle discrete values. However the successor C4.5 Algorithm converts continuous features into discrete ones.
- A continuous feature x_k can be discretized by dividing its range into intervals $[c_1, c_2), [c_2, c_3), \dots, [c_{n-1}, c_n]$ and treating these intervals as discrete values.
- This way, starting from the root, the algorithm grows the decision tree. A node is taken as a leaf if its set is pure or if the set cannot be split by features anymore, or if a preset maximum depth is reached.

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance $\mathbf{x}$, starting from the root, the tree is traversed to find the node to which $\mathbf{x}$ belongs.

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance $\mathbf{x}$, starting from the root, the tree is traversed to find the node to which $\mathbf{x}$ belongs. The most frequent class in the node is assigned to $\mathbf{x}$.

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance $\mathbf{x}$, starting from the root, the tree is traversed to find the node to which $\mathbf{x}$ belongs. The most frequent class in the node is assigned to $\mathbf{x}$.
- ID3, C4.5 and CART algorithms are *greedy algorithms* that may not reach the global optimum.

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance $\mathbf{x}$, starting from the root, the tree is traversed to find the node to which $\mathbf{x}$ belongs. The most frequent class in the node is assigned to $\mathbf{x}$.
- ID3, C4.5 and CART algorithms are *greedy algorithms* that may not reach the global optimum.
- The global Decision Tree optimization problem for a dataset D takes place on the space of all trees T whose nodes v are labeled with subsets $S_v \subset D$.

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance **x**, starting from the root, the tree is traversed to find the node to which **x** belongs. The most frequent class in the node is assigned to **x**.
- ID3, C4.5 and CART algorithms are *greedy algorithms* that may not reach the global optimum.
- The global Decision Tree optimization problem for a dataset D takes place on the space of all trees T whose nodes v are labeled with subsets $S_v \subset D$. The loss (error) function for such a tree is defined as:

$$L(T) = \sum_{v \in T} \frac{|S_v|}{|D|} H(S_v). \tag{2}$$

ID3 and C4.5 Algorithms cont.

- After the decision tree is learned, for inference (prediction) for a new instance **x**, starting from the root, the tree is traversed to find the node to which **x** belongs. The most frequent class in the node is assigned to **x**.
- ID3, C4.5 and CART algorithms are *greedy algorithms* that may not reach the global optimum.
- The global Decision Tree optimization problem for a dataset D takes place on the space of all trees T whose nodes v are labeled with subsets $S_v \subset D$. The loss (error) function for such a tree is defined as:

$$L(T) = \sum_{v \in T} \frac{|S_v|}{|D|} H(S_v). \tag{2}$$

- However finding the optimal tree w.r.t. L is NP-Complete and thus we use greedy algorithms to find a near-optimum.

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

- Gini impurity is slightly faster to compute than entropy.

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

- Gini impurity is slightly faster to compute than entropy.
- The CART (Classification and Regression Trees) algorithm is another decision tree learning algorithm that produces only binary trees.

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

- Gini impurity is slightly faster to compute than entropy.
- The CART (Classification and Regression Trees) algorithm is another decision tree learning algorithm that produces only binary trees.
- At each node v, feature-threshold pairs (x_j, c) are examined, and for each, two children are considered: $S_{u_1} = \{\mathbf{x} \in S_v : x_j \leq c\}$ and $S_{u_2} = \{\mathbf{x} \in S_v : x_j > c\}$.

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

- Gini impurity is slightly faster to compute than entropy.
- The CART (Classification and Regression Trees) algorithm is another decision tree learning algorithm that produces only binary trees.
- At each node v, feature-threshold pairs (x_j, c) are examined, and for each, two children are considered: $S_{u_1} = \{\mathbf{x} \in S_v : x_j \leq c\}$ and $S_{u_2} = \{\mathbf{x} \in S_v : x_j > c\}$.
- The pair that minimizes $\frac{|S_{u_1}|}{|S_v|} G(S_{u_1}) + \frac{|S_{u_2}|}{|S_v|} G(S_{u_2})$ is chosen to split the tree.

Gini Impurity and the CART algorithm

- Gini impurity is another measure of mixedness and is given by

$$G(P) = 1 - \sum_i p_i^2. \tag{3}$$

- Gini impurity is slightly faster to compute than entropy.
- The CART (Classification and Regression Trees) algorithm is another decision tree learning algorithm that produces only binary trees.
- At each node v, feature-threshold pairs (x_j, c) are examined, and for each, two children are considered: $S_{u_1} = \{\mathbf{x} \in S_v : x_j \leq c\}$ and $S_{u_2} = \{\mathbf{x} \in S_v : x_j > c\}$.
- The pair that minimizes $\frac{|S_{u_1}|}{|S_v|} G(S_{u_1}) + \frac{|S_{u_2}|}{|S_v|} G(S_{u_2})$ is chosen to split the tree.
- In Scikit-Learn, decision tree classifier is provided by the class `tree.DecisionTreeClassifier` and uses the CART algorithm and currently does not support categorical features.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data. Thus it has more degrees of freedom than a parametric model such as Naive Bayes or Linear Regression.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data. Thus it has more degrees of freedom than a parametric model such as Naive Bayes or Linear Regression.
- A parametric model is fully determined by its parameters e.g. mean and standard deviation in Gaussian Naive Bayes.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data. Thus it has more degrees of freedom than a parametric model such as Naive Bayes or Linear Regression.
- A parametric model is fully determined by its parameters e.g. mean and standard deviation in Gaussian Naive Bayes.
- To restrict the freedom of DTL, we can use a few parameters in `DecisionTreeClassifier` such as:
  - `max_depth`: maximum depth of the tree (maximum number of edges between the root and leaves).

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data. Thus it has more degrees of freedom than a parametric model such as Naive Bayes or Linear Regression.
- A parametric model is fully determined by its parameters e.g. mean and standard deviation in Gaussian Naive Bayes.
- To restrict the freedom of DTL, we can use a few parameters in `DecisionTreeClassifier` such as:
  - `max_depth`: maximum depth of the tree (maximum number of edges between the root and leaves).
  - `min_samples_leaf`: the minimum number of samples a leaf is allowed to have.

Overfitting and regularization in Decision Tree Learning (DTL)

- DT learning algorithms can produce complicated decision trees for larger datasets, specially if there are many features.
- This is because DTL is a *nonparametric model* which does not make any assumptions on the distribution of data. Thus it has more degrees of freedom than a parametric model such as Naive Bayes or Linear Regression.
- A parametric model is fully determined by its parameters e.g. mean and standard deviation in Gaussian Naive Bayes.
- To restrict the freedom of DTL, we can use a few parameters in `DecisionTreeClassifier` such as:
  - `max_depth`: maximum depth of the tree (maximum number of edges between the root and leaves).
  - `min_samples_leaf`: the minimum number of samples a leaf is allowed to have.
  - `min_samples_split`: minimum number of samples a node must have to be allowed to split.

Overfitting and regularization in Decision Tree Learning (DTL) cont.

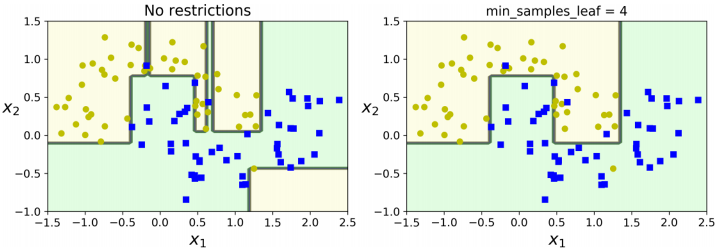


Figure: An example of Decision Tree Learning with and without regularization. Credit: Aurelien Geron

Overfitting and regularization in Decision Tree Learning (DTL) cont.

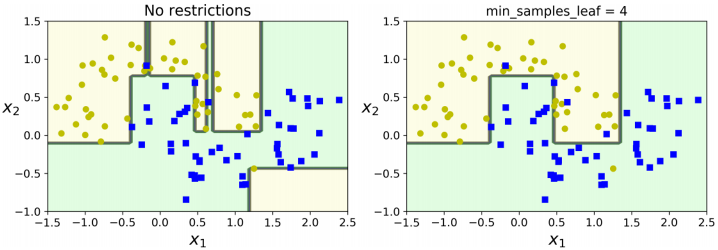


Figure: An example of Decision Tree Learning with and without regularization. Credit: Aurelien Geron

- In Scikit, decision boundaries for a classifier can be drawn using the `inspection.DecisionBoundaryDisplay.from_estimator` function.

Decision Tree Regression

- The CART algorithm can be used for regression as well.

Decision Tree Regression

- The CART algorithm can be used for regression as well. For a continuous target variable y, the predicted value $\hat{y}$ of each node of the tree is the average of the y of its datapoints.

Decision Tree Regression

- The CART algorithm can be used for regression as well. For a continuous target variable y, the predicted value $\hat{y}$ of each node of the tree is the average of the y of its datapoints.
- The measure of impurity is now, the Means Square Error between the predicted and actual values of elements:

$$G(S_v) = \frac{1}{|S_v|} \sum_{(\mathbf{x}_i, y_i) \in S_v} (y_i - \hat{y}_i)^2. \tag{4}$$

Decision Tree Regression

- The CART algorithm can be used for regression as well. For a continuous target variable y, the predicted value $\hat{y}$ of each node of the tree is the average of the y of its datapoints.
- The measure of impurity is now, the Means Square Error between the predicted and actual values of elements:

$$G(S_v) = \frac{1}{|S_v|} \sum_{(\mathbf{x}_i, y_i) \in S_v} (y_i - \hat{y}_i)^2. \tag{4}$$

- Similar to the case of classification, the CART algorithm tries to split each node in such a way that minimizes this impurity.

Decision Tree Regression

- The CART algorithm can be used for regression as well. For a continuous target variable y, the predicted value $\hat{y}$ of each node of the tree is the average of the y of its datapoints.
- The measure of impurity is now, the Means Square Error between the predicted and actual values of elements:

$$G(S_v) = \frac{1}{|S_v|} \sum_{(\mathbf{x}_i, y_i) \in S_v} (y_i - \hat{y}_i)^2. \tag{4}$$

- Similar to the case of classification, the CART algorithm tries to split each node in such a way that minimizes this impurity.
- In Scikit, decision tree regression is provided by `tree.DecisionTreeRegressor`.

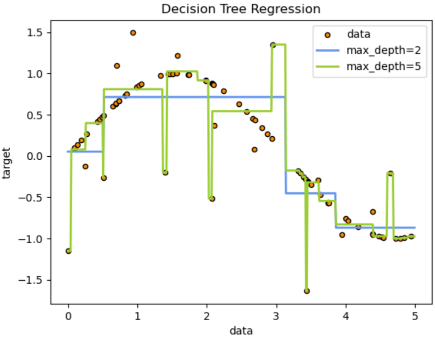


Figure: An example of Decision Tree Regression on sinusoidal data. We have only one feature which is plotted on the x-axis.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.
- Computational complexity of training a decision tree on data with n samples and d features is $O(d \cdot n \cdot \log_2(n))$, because at each node we have to consider all the features.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.
- Computational complexity of training a decision tree on data with n samples and d features is $O(d \cdot n \cdot \log_2(n))$, because at each node we have to consider all the features.
- Decision Trees are very sensitive small changes in the data.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.
- Computational complexity of training a decision tree on data with n samples and d features is $O(d \cdot n \cdot \log_2(n))$, because at each node we have to consider all the features.
- Decision Trees are very sensitive small changes in the data.
- They are also sensitive to rotating the data.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.
- Computational complexity of training a decision tree on data with n samples and d features is $O(d \cdot n \cdot \log_2(n))$, because at each node we have to consider all the features.
- Decision Trees are very sensitive small changes in the data.
- They are also sensitive to rotating the data. This is because DTs only use vertical decision boundaries such as $x_j = c$; they doesn't use combinations of features.

Pros and Cons of Decision Tree Learning (DTL)

- Decision Trees are highly explainable.
- Decision Trees work natively as multi-class classifiers.
- DTL does not make any assumptions on the distribution of data and does not need data normalization.
- Assuming the tree is binary and fairly balanced, the prediction complexity is $\log_2(n)$, equal to the average length of a path from the root to a leaf.
- Computational complexity of training a decision tree on data with n samples and d features is $O(d \cdot n \cdot \log_2(n))$, because at each node we have to consider all the features.
- Decision Trees are very sensitive small changes in the data.
- They are also sensitive to rotating the data. This is because DTs only use vertical decision boundaries such as $x_j = c$; they doesn't use combinations of features.
- Decision Trees can be imbalanced if the classed in data are imbalanced.