# Image Annotation with Decision Tree

# 10

*We may be different, but we all share a common ancestor.*

## 10.1 Introduction

The machine learning methods we discussed so far are typically black box type of classifiers, in the sense that the decisions they make are not transparent to users. In other words, these models are neither interpretable nor comprehensible to users. Another issue with these methods is that their decision-making process is one path or non-conditional process, which means that there are no alternatives when the original decision was not appropriate.

Human beings, however, tend to make decisions in a step-by-step and hierarchical way. For example, when we look at an image with complex patterns, we tend to first organize the different patterns into groups using the most prominent attribute or feature, then go further to identify the objects we are interested, and analyze them in detail using other types of attributes or features. This kind of hierarchical and step-by-step analysis is repeated until we are satisfied.

In machine learning, this kind of intuitive, hierarchical, and step-by-step analysis can be modeled using a *decision tree* or DT. DT is a "divide-and-conquer" approach to learn classification from a set of training samples. A DT is built from a training dataset by recursively dividing the dataset into several subsets based on the possible values of a selected attribute. The procedure starts at the root node and continues until all the instances of a subset have the same class label or there is no other attribute left to divide them.

A DT is typically built upside down with its root at the top. Figure 10.1 shows an example of a DT on image classification [1, 2]. On the DT, an *internal node* (with outgoing branches) is labeled with an input feature or a selected attribute. The branches coming from a node are labeled with each of the possible values of the selected attribute. Each *leaf node* (without outgoing branch) of the tree is labeled with a class or a probability distribution over the classes.



D. Zhang, *Fundamentals of Image Data Mining*, Texts in Computer Science, https://doi.org/10.1007/978-3-030-17989-2_10

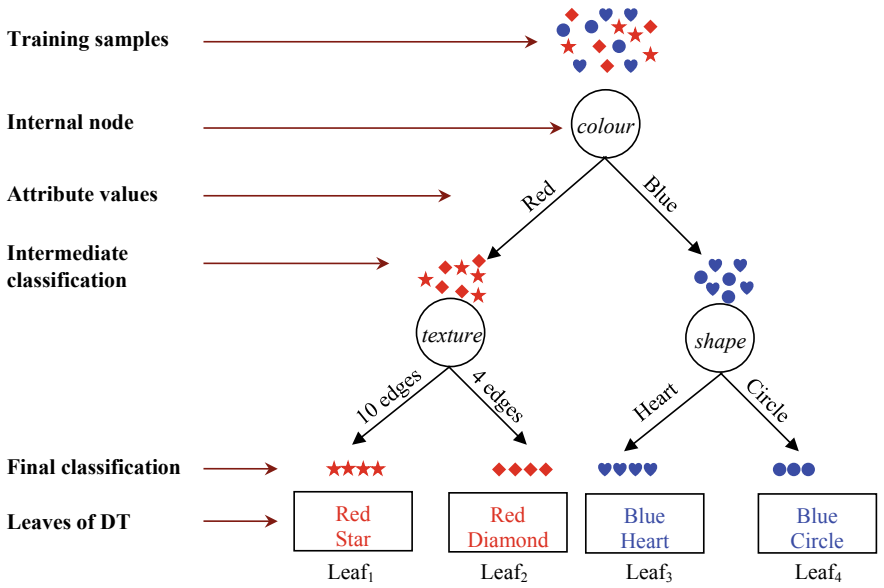


**Fig. 10.1** A DT for image classification

The following is a list of terminologies associated with a DT:

1. *Root Node*. It represents the entire dataset and it is to be divided into two or more homogeneous sets.
2. *Internal Node/Decision Node*. It is a node which can be split into two or more sub-nodes.
3. *Leaf Node/Terminal Node*. It is a node which cannot be split into sub-nodes.
4. *Branch/Sub-tree*. An edge coming out of a node is called a branch and the section under a branch is called a sub-tree.
5. *Parent and Child Node*. A node which is divided into sub-nodes is called parent node and a node under a parent node is called a child node.
6. *Splitting*. It is a process of dividing a node into two or more sub-nodes.
7. *Pruning*. It is a process of removing unwanted sub-nodes and branches. It is the opposite process of splitting.

Depending on the type of attribute values, a DT can be either a *classification tree* or a *regression tree*. A classification tree takes a *discrete* set of attribute values and the predicted outcomes are the class labels to which the data belong, while a regression tree takes *continuous* attribute values and the predicted outcomes are real numbers.

Quinlan [3] first formulated a DT algorithm called ID3 (Iterative Dichotomiser 3) which only accepts discrete features. ID3 is later extended to C4.5 [4] which accepts both discrete and continuous features. In the following, we describe the characteristics of different types of DTs.

## 10.2 ID3

The ID3 algorithm begins with a training dataset $T$ and an attribute set $A$ as the root node. It then checks every unused attribute of the attribute set $A$ and calculates the entropy *info*($T$) (or information gain $IG(T)$) of that attribute. It then selects the attribute which has the smallest entropy (or largest $IG$) value. The set $T$ is then split into subsets by the selected attribute. The above procedure is repeated on each subset until there is no unused attribute or the subset is homogeneous (all instances in the subset are from the same class). ID3 only accepts data with discrete or nominal values. The algorithm of the ID3 can be summarized as follows:

ID3 ($T$, $A$) {

1. Create a root node *Root*
2. If all instances in *Root* belong to the same class $C$ or $A$ is empty

   2.1. Stop and return *Root* with label = $C$

3. Else

   3.1. Select an attribute $A_i$ with possible values $A_i^1, A_i^2, \ldots, A_i^n$
   3.2. Partition $T$ into subsets $T_1$, $T_2$, …, $T_n$ according to the values of $A_i$
   3.3. For each $T_j$

      3.3.1. Add a new branch under *Root* to connect $T_j$ with *Root*
      3.3.1. If $T_j$ is homogeneous

         3.3.1.1. Then, below this new branch add a leaf node with label = $A_i^j$

      3.3.1. Else, below this new branch add the sub-tree ID3 ($T_j$, A − $\{A_i\}$)

}

This algorithm, in fact, is a *general DT algorithm*. Central to a DT algorithm is step 3.1, which requires an attribute selection criterion or a splitting criterion. This splitting criterion determines how the tree looks like and the performance of a DT as well.

### 10.2.1 ID3 Splitting Criterion

Because the split criterion is critical to the success of a DT, variety of criteria have been proposed. The rule is to select an attribute which reduces the maximum amount of uncertainty in data, because the higher the uncertainty in data is, the

more difficult it is to predict the class of an instance. Intuitively, this is equivalent to selecting the most useful or most telling attribute to make a decision. Information gain is a statistical measurement of reducing the uncertainty in data. Therefore, in ID3, the attribute which gives the highest information gain is selected as the test attribute.

Information gain of an attribute measures how much information we can save or gain if it is selected to split the training set. Mathematically, it is measured as the *difference* between information needed to classify an instance before and after the attribute splits the training dataset.

*Information before the splitting*:

- Given a training set $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i$ is the data or sample, $y_i$ is the class label for $\mathbf{x}_i$, and $y_i \in \{C_1, C_2, \ldots, C_m\}$.
- Instances in $T$ are characterized with the set of attributes $A = \{A_1, A_2, \ldots, A_n\}$.
- Each attribute $A_i$ has possible values $A_i^1, A_i^2, \ldots, A_i^{n_i}$.
- The probability that an instance of $T$ belongs to class $C_j$ is given as

$$P_j = \frac{\left\|C_j\right\|}{\|T\|} \tag{10.1}$$

where $\left\|C_j\right\|$ is the number of instances in $C_j$.

To classify an instance in $T$, the information needed (or the *entropy*) is given as,

$$info(T) = -\sum_{j=1}^{j=m} P_j \times \log P_j \tag{10.2}$$

The negative sign before the sum is to make the information a positive value; this is because $P_j \leq 1$ and $\log P_j \leq 0$. Generally, *entropy* refers to *disorder* or *uncertainty* in a dataset; a smaller *info*($T$) means a more predictable class. This is consistent with our understanding of information. An event with higher chance of occurrence carries little to zero information such as sunrise/sunset, while an event with less chance of occurrence carries more information such as rain/sunshine. In English language, frequent words such as "a", "the", and "this" carry almost zero information, while rare words such as "Delphi", "nirvana", and "dialectics" carry a lot of information. In terms of a dataset, a data source (or a class) with higher probability value carries less information than a data source with lower probability value. Therefore, a DT learning algorithm attempts to split $T$ into subsets so that the expected information needed is minimized after a split.

*Information after the splitting*:

Suppose, an attribute $A_i$ has $n_i$ nominal values such as $A_i^1, A_i^2, \ldots, A_i^{n_i}$. If attribute $A_i$ is selected at the current node, it splits the training set $T$ into $T_i^1, T_i^2, \ldots, T_i^{n_i}$. After the splitting, the *expected information* is calculated as

$$E(A_i) = \sum_{j=1}^{j=n_i} \frac{\left\|T_i^j\right\|}{\|T\|} \times info(T_i^j) \tag{10.3}$$

*The information gain is the difference between info(T) and* $E(A_i)$:

$$IG(A_i) = info(T) - E(A_i) \tag{10.4}$$

The attribute which gives the highest *IG* is chosen for splitting the training set. Because *info*(*T*) in (10.2) is the same for all attributes, the attribute $A_i$ which gives the highest gain has lowest expected information $E(A_i)$. Therefore, the attribute which leads to the least expected information is selected.

## 10.3 C4.5

C4.5 build a DT from a training dataset in the same way as ID3, except C4.5 has made a number of improvements to ID3:

- Use *gain ratio* (*GR*) instead of *IG* to build a better DT.
- Accept both *continuous* and *discrete* attributes. For continuous attributes, C4.5 creates a threshold and then splits the list into those whose attribute value is above the threshold and those that are less than or equal to it.
- Handle *incomplete data* points. C4.5 allows attribute values to be marked as "?" for missing data.
- Apply different *weights* to the attributes.
- Overcome *over-fitting* problem by a bottom-up *pruning*. C4.5 goes back through the tree once it has been created and removes branches that are deemed unnecessary by replacing them with leaf nodes.

### 10.3.1 C4.5 Splitting Criterion

In ID3, *IG* is used as splitting criterion. However, the disadvantage of using *IG* as splitting criterion in ID3 is that it favors the highly branching attributes, that is, the attributes which have a large number of possible values. Let us think about the extreme case where the instance ID is used as an attribute. Say, it is denoted as $A_{ID}$, and it has a distinct value for each instance. If $A_{ID}$ is used to split the dataset *T*, each subsequent subset will have only one instance. According to (10.3), $E(A_{ID})$ is zero because each subset has zero entropy (information). Therefore, $IG(A_{ID})$ will be the highest and $A_{ID}$ will be selected. However, such a selection tells nothing about the nature of the decision and leads to no classification at all. To reduce the effect of

high branching factor on information gain, a modified measure called *gain ratio* (*GR*) or *normalized IG* is proposed by Quinlan in C4.5. The gain ratio is defined as *IG* normalized by split information [4]

$$GR(A_i) = \frac{IG(A_i)}{splitInfo(A_i)} \tag{10.5}$$

where $splitInfo(A_i)$ is calculated based on the proportion of each subset resulted from the split using attribute $A_i$, regardless of the class information inside each subset. Specifically, it is defined as

$$splitInfo(A_i) = -\sum_{j=1}^{j=n_i} \left( \frac{\|T_i^j\|}{\|T\|} \times \log\left( \frac{\|T_i^j\|}{\|T\|} \right) \right) \tag{10.6}$$

It can be observed that the higher the number of attribute values of $A_i$, the larger the magnitude of $splitInfo(A_i)$ and the lower its gain ratio. Therefore, using gain ratio, the high branching behavior is penalized.

## 10.4 CART

In machine learning, a DT can be either *classification tree* or a *regression tree*. For a classification tree, the predicted outcome is a class such as tree, tiger, water, etc., while for a regression tree, the predicted outcome is a real number, such as stock price, queueing time, etc.

CART stands for Classification And Regression Tree; it is an umbrella term used to cover both classification DT and regression DT. It was first introduced by Breiman et al. in 1984 [5]. A CART tree is a *binary* DT that is constructed by splitting a node into two child nodes repeatedly, beginning with the root node that contains the entire training data. The splits are done using the twoing criteria and the obtained tree is pruned by cost–complexity technique. CART can handle both numeric and nominal attributes, and it can also handle outliers.

### 10.4.1 Classification Tree Splitting Criterion

A CART uses splitting criteria called the *twoing criteria* for a classification tree, which is defined as (10.7)

$$\Delta i(t) = \frac{P_L P_R}{4} \left( \sum_{j=1}^{m} |p(j|t_L) - p(j|t_R)| \right)^2 \tag{10.7}$$

where

- $t$ is the node to be split,
- $\Delta i(t)$ indicates the *impurity* of node $t$,
- $P_L$ is the proportion of data split into the left node $t_L$ and similar for $P_R$,
- $P_L = ||t_L||/||t||$ and $P_R = ||t_R||/||t||$, where $t$ is the parent node of $t_L$ and $t_R$, $\|t\|$ is the total number of data in node $t$,
- $p(j|t_L)$ is the proportion of data belonging to class $j$ at the left node $t_L$, and
- $m$ is the number of classes in the training set.

The twoing criteria measure the difference between the two split nodes, and a split is achieved by maximizing the difference or $\Delta i(t)$.

*Gini impurity* can also be used to define a splitting criterion for a classification tree. First, the Gini index (*GI*) or *GI*s are computed for both the left split node $t_L$ and right split node $t_R$, which are given in (10.8) and (10.9), respectively.

$$GI(t_L) = 1 - \sum_{j=1}^{m} [p(j|t_L)]^2 \tag{10.8}$$

$$GI(t_R) = 1 - \sum_{j=1}^{m} [p(j|t_R)]^2 \tag{10.9}$$

It can be observed from (10.8) and (10.9) that

- A *GI* is maximum or $GI = (1 - 1/m)$ when records in a node are equally distributed among all classes, indicating maximum uncertainty.
- A *GI* is minimum or $GI = 0$ when all records in a node belong to one class, indicating minimum uncertainty.

A split is achieved by minimizing the *Gini impurity* which is defined as (10.10)

$$iG(t) = [||t_L|| \cdot GI(t_L) + ||t_R|| \cdot GI(t_R)]/||t|| \tag{10.10}$$

The Gini impurity splitting algorithm works faster than twoing splitting algorithm; however, the twoing splitting criterion builds a more balanced DT and offers a superior performance on complex classification such as multi-class and noisy data.

### 10.4.2 Regression Tree Splitting Criterion

A regression tree is also called a prediction tree. Instead of identifying the class label for a training or unknown data, a regression tree predicts the likely target value

of the data. For regression tree, the splitting criterion is typically given by the mean squared error (*MSE*). Given a training set $T = \{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \ldots, (\mathbf{x}_N, y_N)\}$, where $\mathbf{x}_i$ is the data or sample and $y_i$ is the target value for data $\mathbf{x}_i$. The splitting is determined by the two nodes which give the smallest *MSE*:

$$MSE = \frac{1}{N_L}\sum_{i=1}^{N_L}(y_i - \hat{y}_L)^2 + \frac{1}{N_R}\sum_{j=1}^{N_R}(y_j - \hat{y}_R)^2 \tag{10.11}$$

where

- $N_L$ and $N_R$ are the total number of samples falling into the left split node and the right split node, respectively,
- $\hat{y}_L$ and $\hat{y}_R$ are the prediction values for the left and right split nodes, respectively,
- $\hat{y}_L$ is typically given as the result of a *regression* from the data falling into the left node: $\hat{y}_L = \hat{f}_L(\mathbf{x}_i) = \boldsymbol{\beta}^T \cdot \mathbf{x}_i + b$.

A *regression* tree is basically a *piecewise linear* approximation of a dataset in space. Figure 10.2 demonstrates a contrast between a global linear regression and a regression tree approximation on one variable dataset. In the figure, the green line shows an approximation from a global linear regression, while the red lines represent a regression tree approximation of the same data. It can be observed that the regression tree gives a much closer approximation than the global linear regression model.

### 10.4.3 Application of Regression Tree

The prediction value $\hat{y}_L$ in (10.11) can also be estimated by the *mean* of the left node $\hat{y}_L = \frac{1}{N_L}\sum_{i=1}^{N_L} y_i$ (similar for $\hat{y}_R$). The tree built in this way provides a *piecewise constant* approximation of the data. The mean prediction model is a much faster method to build a regression tree.

Figure 10.3 shows a mean prediction tree for predicting median house price of California based on the two variables: latitude and longitude. The actual data map and the tree partitions are shown in Fig. 10.4. It can be seen that the finer partitions are concentrated at the darker areas.

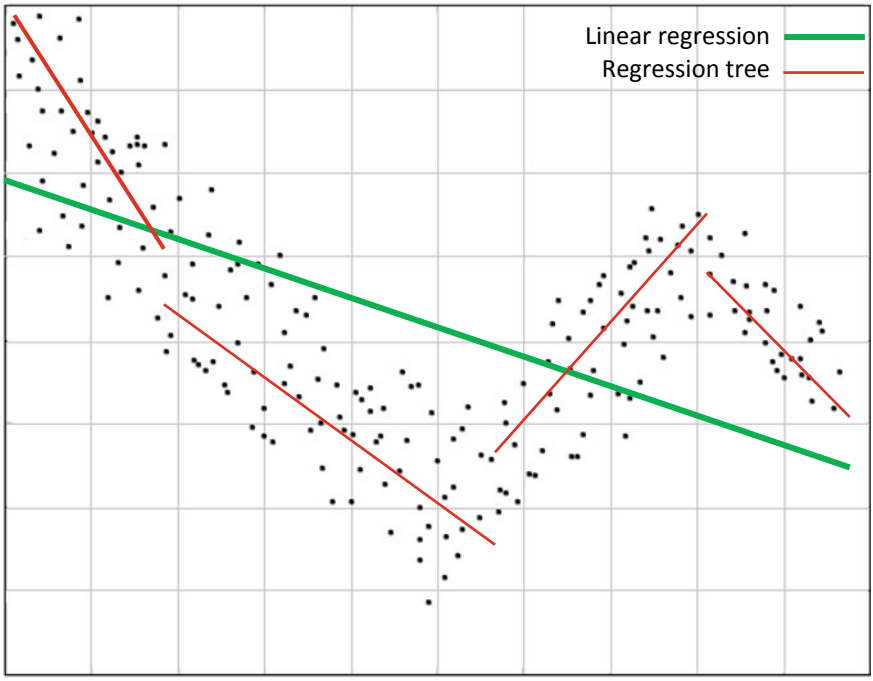


**Fig. 10.2** Contrast between linear regression and regression tree

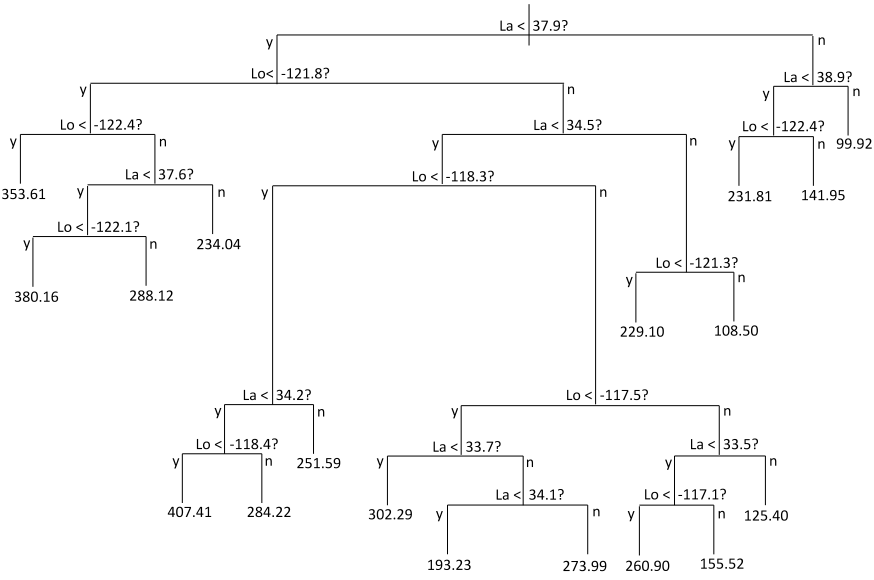


**Fig. 10.3** A regression tree for predicting median house price ('000) in California from the geographic coordinates of Fig. 10.4. Legend: La = latitude, Lo = longitude, y = yes, n = no

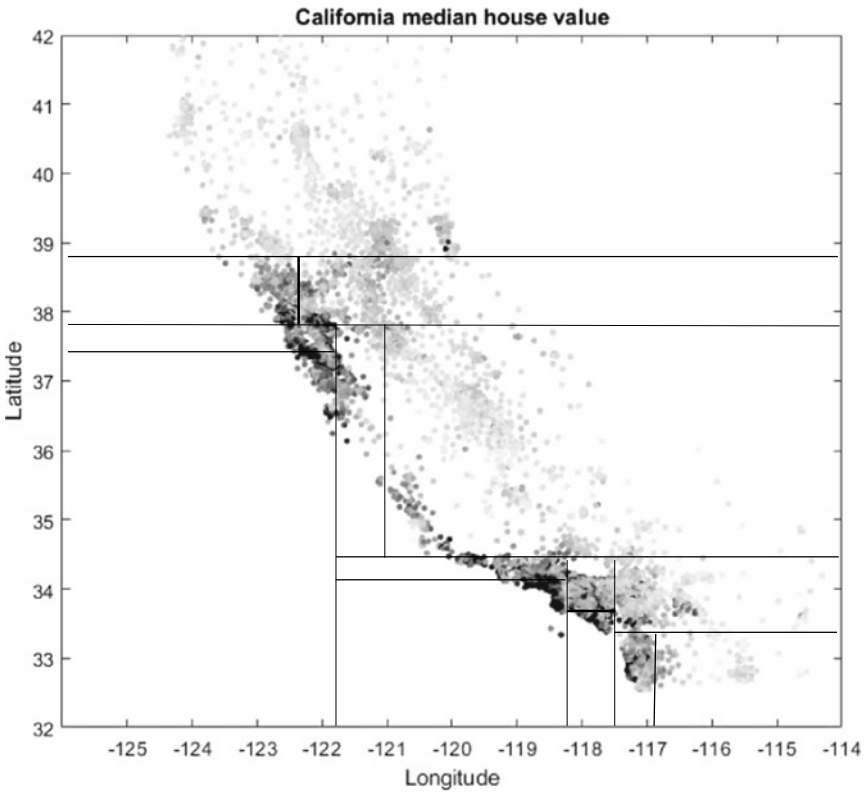


**Fig. 10.4** Data map of actual median house prices in California and the tree partition of the data map, the darker the color, the higher the house value

## 10.5 DT for Image Classification

Images are complex data. A typical image usually has multiple regions/objects and has multiple interpretations. Therefore, the first step is to segment an image into individual regions and represent each region with an $n$-dimensional feature vector: $\mathbf{x} = (x_1, x_2, \ldots, x_n)$. For color images, each image region is represented with a color feature vector $\mathbf{x}_C$ and a texture feature vector $\mathbf{x}_T$. Because certain types of image regions can be best described by both color and texture, the third feature is also created by combining both color and texture into a single feature vector $\mathbf{x}_{CT}$.

### 10.5.1 Feature Discretization

The three types of features $\mathbf{x}_C$, $\mathbf{x}_T$, and $\mathbf{x}_{CT}$ are all continuous features; in order to classify these regions using a classification DT, the features need to be quantized

into discrete values using a vector quantization (VQ) technique. The idea of a VQ technique is to cluster similar image regions into clusters which are then assigned with nominal values such as 0, 1, 2, …, $K$. These nominal values correspond to class labels such as sky, water, grass, firework, tiger, etc., which are used for DT classification. Common VQ algorithm is the LBG algorithm [6] which is given as follows:

LBG ($T$, $K$, $\varepsilon$) {
*Input:* $T = \{\mathbf{x}_i \in R^n, i = 1, 2, \ldots, N\}$
*Output:* $C = \{c_j \in R^n, j = 1, 2, \ldots, K\}$

1. Initiate a codebook $C = \{c_j \in R^n, j = 1, 2, \ldots, K\}$
2. Set $D_0 = 0$ and $k = 0$
3. Classify the $N$ training vectors into $K$ clusters $T_q (q = 1, 2, \ldots, K)$ and classify $\mathbf{x}_i$ to $T_q$ if the distance $d(\mathbf{x}_i - c_q) < d(\mathbf{x}_i - c_j)$ for all $j \neq q$
4. Update cluster centroids $c_j$ by $c_j = \frac{1}{|T_j|} \sum_{\mathbf{x}_i \in T_j} \mathbf{x}_i (j = 1, 2, \ldots, K)$
5. Set $k \leftarrow k + 1$ and compute the distortion $D_k = \sum_{j=1}^{K} \sum_{\mathbf{x}_i \in T_j} d(\mathbf{x}_i - c_j)$
6. If $(D_{k-1} - D_k)/D_k > \varepsilon$ (*a small positive number*), repeat steps 3–5
7. Return the codebook $C = \{c_j \in R^n, j = 1, 2, \ldots, K\}$

}

By applying the LBG VQ algorithm on $\mathbf{x}_C$, $\mathbf{x}_T$, and $\mathbf{x}_{CT}$, three codebooks or visual dictionaries $V_i (i = 1, 2, 3)$ are created:

$$V_i = \left\{ v_1^i, v_2^i, \ldots, v_{n_i}^i \right\}, \quad i = 1, 2, 3 \tag{10.12}$$

where $v_j^i$ represents a code word of $V_i$ and $n_i$ is the total number of code words in $V_i$.

For each image region $R$ in the training dataset, it is represented as three discrete attribute values as follows:

$$R = (ind_1, ind_2, ind_3) \tag{10.13}$$

and

$$ind_i = \arg\min_j (dist(v^i, v_j^i)) \tag{10.14}$$

where

- $dist(v^i, v_j^i)$ is the distance between feature $v^i$ and code word $v_j^i$,
- $v^i$ is one of the feature vectors ($\mathbf{x}_C$, $\mathbf{x}_T$ or $\mathbf{x}_{CT}$) of $R$,

- $v_j^i$ is the $j$th value of attribute $V_i$, and
- $ind_i$ is an index value of attribute $V_i$ and it is an integer between 0 and $n_i$.

Therefore, each image region in the training set is now associated with three discrete attributes $A_i = V_i(i = 1, 2, 3)$ which are ready for building a DT.

### 10.5.2 Building the DT

With a training set of image regions $T$ and a set of visual attributes $A$, an image classification DT can be built using the following algorithm [1]:

1. If all training regions of $T$ belong to the same class $C$,

   1.1. The tree is a leaf node with the outcome $C$.
   1.2. Stop.

2. If the regions of $T$ belong to more than one class but there is no attribute to separate them,

   2.1. The tree is a leaf node. The outcome is determined as follows.

      2.1.1. If there is a single majority class in $T$, the outcome is that class.
      2.1.2. Else, the outcome is the majority class of the parent node.

   2.2. Stop.

3. If the regions of $T$ belong to more than one class and there are one or more attributes to separate them

   3.1. Create an internal node.
   3.2. Calculate the *IG* or *gain ratio* for each attribute.
   3.3. Select the attribute $A_i$ with the highest gain ratio: $A_i = \{v_1^i, v_2^i, v_3^i, \ldots, v_{n_i}^i\}$
   3.4. Use $A_i$ as the test attribute for the internal node.
   3.5. Split the training set $T$ into subsets: $T_i^0$, $T_i^1$, $T_i^2$, $T_i^3$, …, $T_i^{n_i}$, where image regions in $T_i^j$ have attribute value $v_i^j$.
   3.6. Remove attribute $A_i$ from the attribute list.
   3.7 Repeat from Step 1 for each $T_i^j$.

A DT generated from the above algorithm can have nodes with isolated or noisy samples. A common practice is to include a pre-pruning procedure after each splitting to remove those nodes with noisy samples. The following pre-pruning procedure can be included as step 3.7 in the above algorithm, and name the original step 3.7 as 3.8.

3.7 Pre-pruning [7]:

3.7.1 Calculate the probability $P_i'$ for every class in subset $T_i^j$ ($j = 0, 1, 2, \ldots, n_i$) as

$$P_i' = \frac{||p_j||}{||p_i||} \tag{10.15}$$

where $p_i$ and $p_j$ are the number of instances of class $C_i$ in $T$ and $T_i^j$, respectively.

3.7.2 Remove those samples from subset $T_i^j$ whose class probability $P_i'$ is less than a threshold $k$.

3.7.3 Remove $T_i^j$ if it is an empty subset after sample removal.

Figure 10.5 shows a DT learnt from a dataset of 570 image regions which have been quantized into 19 classes (30 images/class), and the pre-pruning threshold is $k = 0.1$ [8]. The meanings of the leave labels are as follows: A = *Grass*, B = *Forest*, C = *Sky*, D = *Sea*, E = *Flower*, F = *Sunset*, G = *Beach*, H = *Firework*, I = *Tiger*, J = *Fur*, K = *Eagle*, L = *Building*, M = *Snow*, N = *Rock*, O = *Bear*, P = *Night*, Q = *Crowd*, R = *Butterfly*, S = *Mountain*, and U = *Unknown*.

A DT generated from the above algorithm can still be very complex and imbalanced. A post-pruning procedure is usually applied after the initial tree has been generated; this can be added as step 4 in the above DT algorithm. The post-pruning is a procedure to *remove those isolated branches* and merge them with neighboring nodes.

1. Post-pruning [7]:

   a. If for more than one value of the attribute $A_i$, the outcome class labels are identical, i.e., $C_i$, then all the leaf nodes corresponding to these attribute values are merged as a single leaf node labeled with class $C_i$.
   b. If the outcomes for all the possible values of attribute $A_i$ are identical, i.e., $C_i$, then the *sub-tree* rooted at $A_i$ is replaced with a single leaf node with $C_i$ as an outcome.

Figure 10.6 shows the DT from Fig. 10.5 after post-pruning; it is a much simpler and more robust DT [7, 8].

### 10.5.3 Image Classification and Annotation with DT

Once the DT is generated from a training dataset, a set of rules or a *DT model* can be formulated from the DT by traversing the tree from the root to each of the leaf

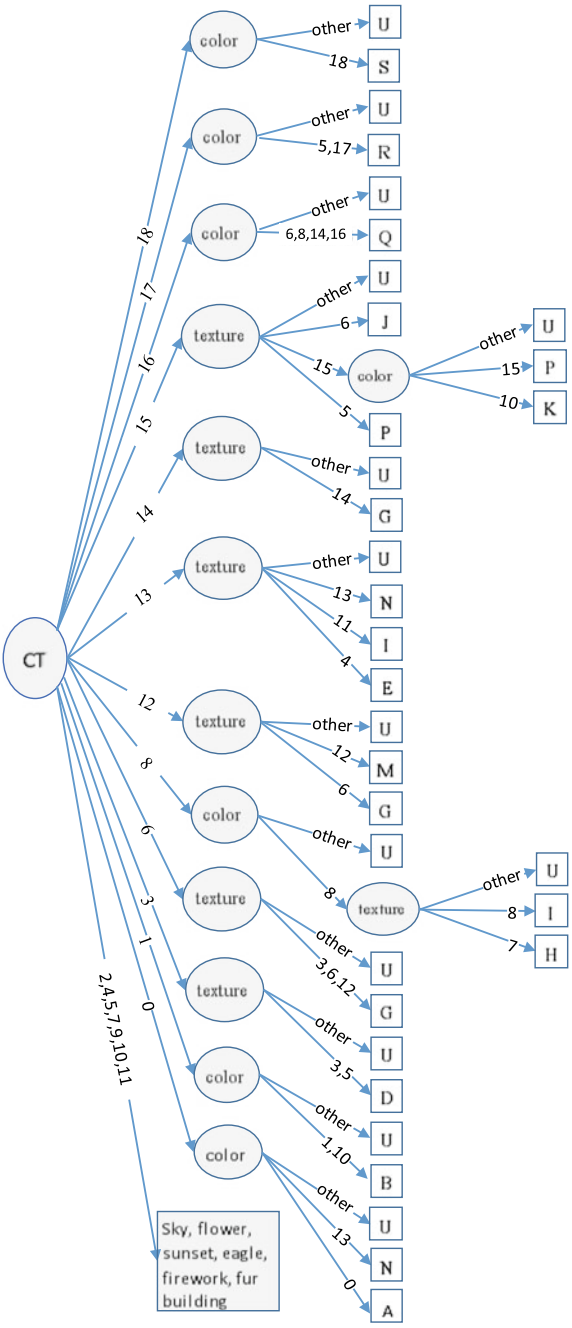


**Fig. 10.5** A DT for image classification without post-pruning, CT = color and texture

nodes. This DT model is to be used as an image classifier. The following is the DT model formulated from the DT shown in Fig. 10.6.

```
IF CT is 1 (or 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 14, 15, 16, 17) THEN
{
        Region is Forest (or Sky, Sea, Flower, Sunset, Beach, Firework, Fur, Eagle, Building,
        Snow, Bear, Butterfly, Crowd, Mountain respectively)
}
ELSE IF CT is 0 AND
{
        IF color is 13 THEN Region is Rock
        ELSE Region is Grass
}
ELSE IF CT is 8 AND
{
        IF color is 8 AND
                IF texture is 7 THEN Region is Firework
                ELSE Region is Tiger
        ELSE Region is Tiger
}
ELSE IF CT is 13 AND
{
        IF texture is 4 THEN Region is Flower
        ELSE IF texture is 11 THEN Region is Tiger
        ELSE Region is Rock
}
ELSE IF CT is 15 AND
{
        IF texture is 9 THEN Region is Fur
        ELSE IF TEXTURE IS 15 AND
                IF color is 10 THEN Region is Eagle
                ELSE Region is Night
        ELSE Region is Night
}
END
```

Given a new or unknown image, it is also segmented into regions using the same algorithm as that used by the training dataset. Each region is then represented as three discrete attribute values: $R = (ind_1, ind_2, ind_3)$, using the learnt visual dictionaries. Each of the regions $R$ is then analyzed using the DT model as shown in the above and is classified into a class.

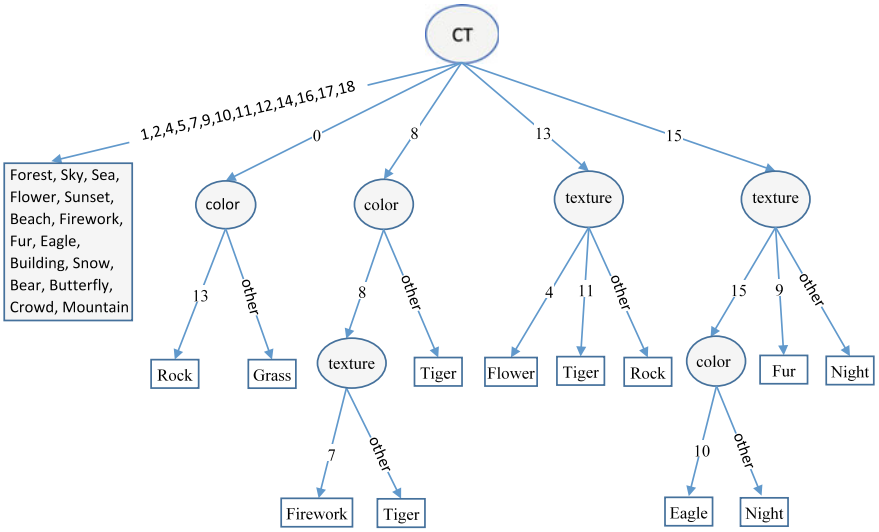


**Fig. 10.6** The DT from Fig. 10.5 after pruning

As can be observed, a DT model is completely transparent and comprehensible to a human user. A classifier working in this way can be easily modified and fine-tuned to adapt to the data. The other advantage of the DT model is that issues can be found and corrected at learning stage without much difficulty.

## 10.6 Summary

DT is a powerful image classification tool. Due to its hierarchical nature and piecewise approximation of data, it offers a middle ground between *generative* and *discriminative* approaches. Compared with other classification tools, DT has a number of advantages.

- DT is a tool known for its simplicity, transparency, and comprehensibility.
- DT can handle both numeric and categorical attributes.
- DT can handle both noisy and missing data.
- DT offers an intuitive and step-by-step analysis based on selected attributes.
- DT does not require complex computation.
- DT generates rules which are easy to interpret.

A DT may grow too complex and imbalanced due to noise, fragmentation, and missing data. Therefore, a pruning mechanism is essential to a DT algorithm. Common practice is to apply a post-pruning technique after the tree has been generated. However, many misplaced instances would have a better chance to be