

## DECISION TREES

A **decision tree** is a type of flowchart that shows a clear pathway to a decision. It starts at a single point (or **root node**) which then branches (or **splits**) in two or more directions. Each branch offers different possible outcomes until a final outcome is achieved. When plotted as a graph, it resembles a tree. The nodes at the end of the decision path are called **leaf nodes** (or **terminal nodes**). Between the root node and the leaf nodes, there are a number of **internal nodes** (or **decision nodes**).

Decision trees work for both categorical and numerical variables. Their objective is to partition the population into homogeneous non-overlapping sets, based on the most significant **input** (or **explanatory** or **predictor**) **variables**. The following two types of trees are commonly used in practice: **regression tree** (for a continuous **target variable**), and **classification tree**, for the categorical target variable.

### Classification and Regression Trees

Different rules apply to "grow" regression and classification trees. For regression trees, residual sum of squares (RSS) and chi-squared automated interaction detection (CHAID) splitting criteria are used. For classification trees, entropy, Gini, and CHAID criteria are used.

Sometimes created regression and classification trees are too complex, with redundant splits, and a very small number of cases in leaf nodes. In this case **pruning** of the tree is advisable, which results in a smaller number of leaves with more substantial splitting and easier interpretation. The cost-complexity pruning technique is commonly used.

To compare the performance of fitted regression trees and choose the best-performing tree, it is customary to split randomly the original data set into a training set (containing typically 70% or 80% of the data rows) and a testing set (containing the remaining 30% or 20% of the rows). A decision tree is developed on the training set, and the goodness-of-fit is verified on the testing set. For regression trees (when the response is continuous), the response is predicted on the testing data set, and proportions of predicted values that are within, say, 10%, 15%, and 20% of the true values are computed and compared for different models. The model for which these proportions are the highest is the best predicting model. For classification trees, prediction is in the form of the probability of being in each category. We can assume that the category with the highest probability is the one that is being predicted, and the measure of performance is the proportion of predictions that coincide with the true observations in the testing data set. The model with the highest proportion of correctly predicted categories should be chosen, or, equivalently, the model with the smallest **misclassification rate** should be chosen.

## Regression Tree

### The RSS Splitting Criterion

For Residual-Sum-of-Squares (RSS) splitting criterion, the predictor space is partitioned into multi-dimensional rectangles  $R_1, \dots, R_J$  that minimize the **residual sum of squares** (RSS)  $\sum_{j=1}^J \sum_{i \in R_j} (y_i - \hat{y}_{R_j})^2$  where  $y_i$  is the observed target variable for individual  $i$ ,  $i = 1, \dots, n$ , and  $\hat{y}_{R_j}$  is the estimated average response for all observations in the set  $R_j$ . In the tree-building process (also termed **training a decision tree**), a top-down binary splitting is used to obtain two new branches at each decision node. For a predictor variable  $X$ , the split  $X = s$  is chosen in such a way that it leads to the largest reduction of RSS for the two subregions  $\{X < s\}$  and  $\{X \geq s\}$ .

**Remark.** It is easy to see that the RSS is minimized at the mean value. Indeed, to minimize  $\sum_{i=1}^n (y_i - \hat{y})^2$ , we take the derivative with respect to  $\hat{y}$  and set it equal to zero. We get  $-2 \sum_{i=1}^n (y_i - \hat{y}) = 0$  or  $\sum_{i=1}^n y_i - n\hat{y} = 0$ . From here,  $\hat{y} = \frac{1}{n} \sum_{i=1}^n y_i = \bar{y}$ .  $\square$

**Remark.** The splitting procedure based on minimizing the RSS is an example of the **classification and regression tree (CART)** algorithm. More generally, this algorithm involves splitting based on the minimization of some quantity. Note that in the CART algorithm, only binary splitting is allowed.

**Historical Note.** The CART algorithm was introduced in the book "Classification and Regression Trees" by Breiman, L., Friedman, J., Olshen, R. and Stone, C, Chapman and Hall, Wadsworth, New York, 1984.

### The CHAID Splitting Criterion

The chi-squared automatic interaction detection (CHAID) splitting criterion is based on statistical tests. To grow a tree, the response variable can be either continuous or categorical but the predictor variables must be all categorical only (that is, continuous predictors are automatically segmented).

While training a tree, the next best split is determined by an F-test for the continuous response variable and by a chi-squared test for the categorical response variable. The predictor variable with the smallest (Bonferroni adjusted)  $p$ -value, i.e., the predictor variable that will yield the most significant split will be considered for the next split in the tree. If the smallest  $p$ -value for any predictor is greater than some pre-specified value of alpha, then no further splits will be performed, and the respective node will become a terminal node.

## The Bonferroni Adjusted $P$ -value

The Bonferroni correction helps to protect against inflation of the probability of Type I error when performing multiple simultaneous hypotheses testing (it, however, inflates the probability of Type II error). Suppose we would like to test simultaneously  $k$  pairs of hypotheses and maintain the probability of Type I error equal to  $\alpha$ . The probability of Type I error is then the probability that at least one test shows significance when no significance exists. That is, at least one test statistic falls in the (adjusted) rejection region. The complement of that is that all test statistics lie in the (adjusted) acceptance region, and this translates into the identity:  $1 - \alpha = \mathbb{P}(\text{all test statistics are in the (adjusted) acceptance region}) = (1 - \alpha_{\text{adjusted}})^k$ , assuming that tests are independent. From here, the Bonferroni adjusted  $p$ -value is a  $p$ -value that should be compared to  $\alpha_{\text{adjusted}} = 1 - (1 - \alpha)^{(1/k)} \approx \alpha/k$ . For example, if  $k = 5$  and five simultaneous hypothesis testings are carried out, then each  $p$ -value should be compared not to  $\alpha = 0.05$  but  $\alpha/k = 0.05/5 = 0.01$ .

**Remark.** Note that with the CHAID splitting criterion, each internal node can have more than two emanating branches, that is, a tree is not limited to binary splitting.

**Historical Note.** The CHAID splitting criterion was proposed by G.V Kass in "An Exploratory Technique for Investigating Large Quantities of Categorical Data", Journal of the Royal Statistical Society, Series C (Applied Statistics), Vol. 29, No. 2 (1980), pp. 119-127.

## The Cost-complexity Pruning Technique

For a decision tree, the **complexity of the tree  $T$**  is defined as  $|T|$ , the number of terminal nodes (leaves). The **cost** of a decision tree, denoted by  $R(T)$ , is the RSS for regression trees and the misclassification rate for classification trees. The **cost-complexity (CC) measure** is defined as  $CC(T) = R(T) + \alpha \cdot |T|$ , a linear combination of the cost and complexity, where the term  $\alpha \cdot |T|$  is called the **complexity penalty term**. The parameter  $\alpha$  is known as the **cost-complexity parameter (ccp)**.

The cost-complexity pruning algorithm (also known as the **minimal cost-complexity** pruning algorithm) works as follows. The cost-complexity of a single node is  $CC(t) = R(t) + \alpha$ . The branch  $T_t$  is defined to be a tree with  $t$  as the root node. In general, the cost of a node is greater than the sum of the costs of its terminal nodes, that is,  $R(T_t) < R(t)$ . However, the cost-complexity measure of a node  $t$  and its branch  $T_t$  can be made equal, depending on the value of  $\alpha$ . We define the **effective  $\alpha$**  of a node to be the value where  $CC(T_t) = CC(t)$ , or  $R(T_t) + \alpha_{\text{eff}} |T_t| = R(t) + \alpha_{\text{eff}}$ , giving

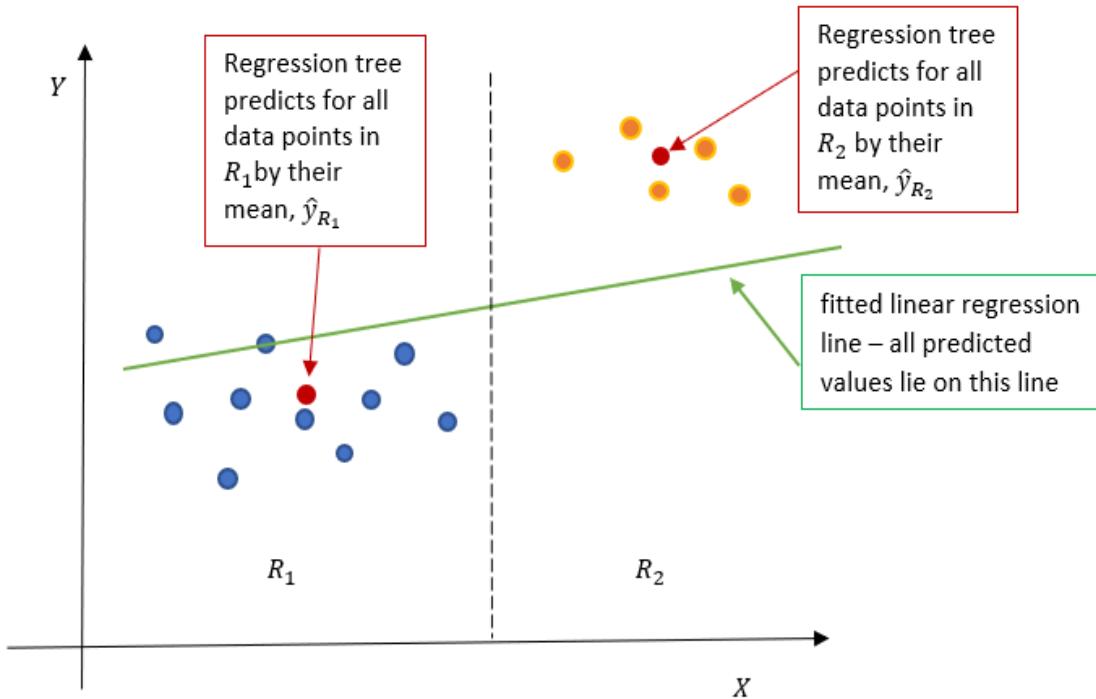
$$\alpha_{\text{eff}} = \frac{R(t) - R(T_t)}{|T_t| - 1}.$$

An internal node with the smallest value of  $\alpha_{\text{eff}}$  is the **weakest link** and will be pruned. The pruning process continues until there are no more effective  $\alpha$ 's smaller than a pre-specified value.

More about this pruning algorithm will be explained in the example below.

**Historical Note.** The cost-complexity pruning algorithm was first described in the book "Classification and Regression Trees" by Breiman, L., Friedman, J., Olshen, R. and Stone, C, Chapman and Hall, Wadsworth, New York, 1984.

**Remark.** When data are not distributed in a linear pattern, fitting a regression tree has a clear advantage over a linear regression model as illustrated below.



**Example.** The data set "housing\_data.csv" contains variables aggregated by residential neighborhoods: housing median age, total number of rooms, total number of bedrooms, total population, total number of households, median income, ocean proximity, and median house value. There are 2842 rows in this data set. We model median house value using a regression tree with the RSS splitting and cost-complexity pruning. First, we split the data into 80% training and 20% testing sets. Then we run a full regression tree (without pruning). We use the RSS splitting criterion. Note that since the tree is large, we need to specify a seed. The RSS splitting algorithm is deterministic, but if two splits are equivalent, the order of splits is carried out at random.

In SAS:

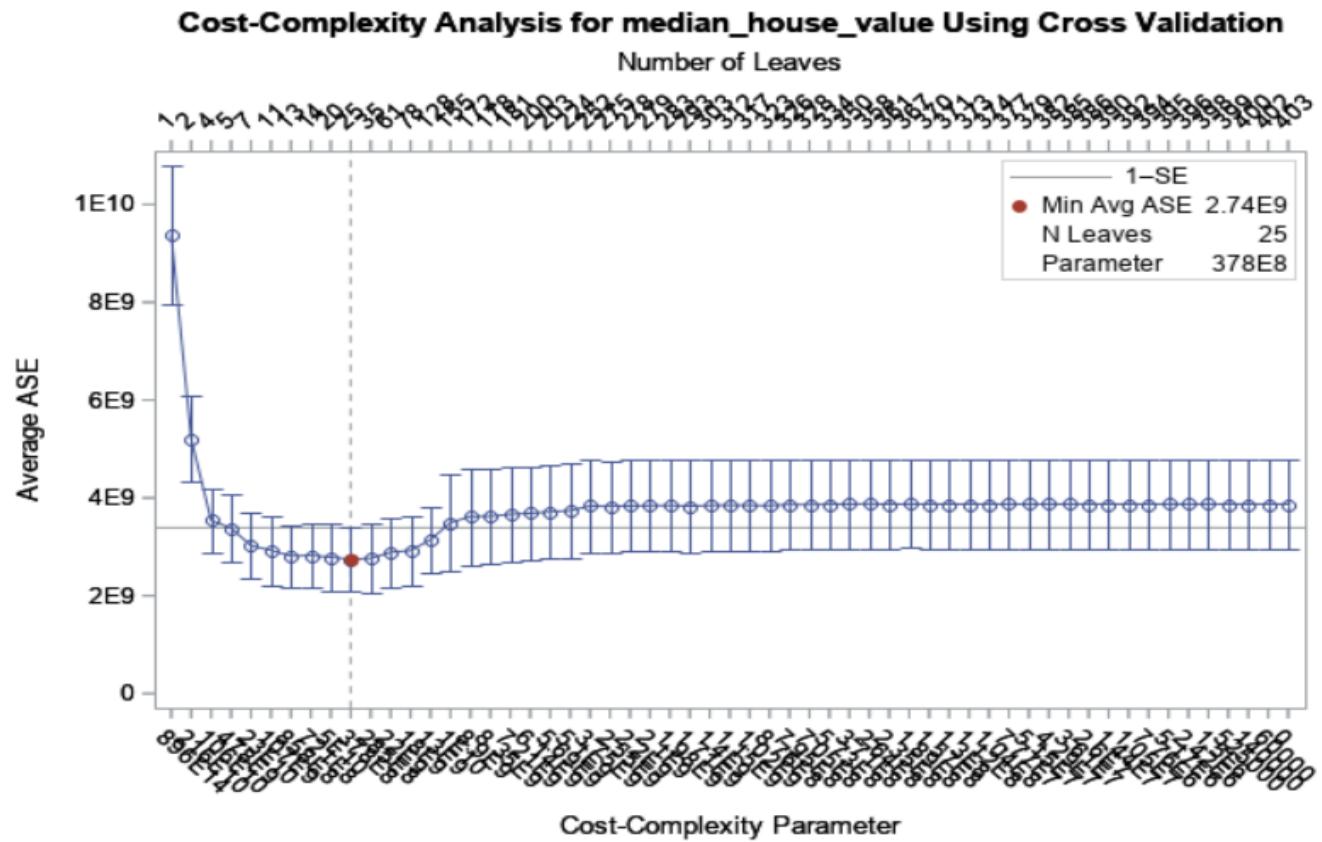
```

proc import out=housing datafile="./housing_data.csv" dbms=csv replace;
run;

/*SPLITTING DATA INTO 80% TRAINING AND 20% TESTING SETS*/
proc surveyselect data=housing rate=0.8 seed=677530
out=housing outall method=srs;
run;

/*RSS SPLITTING CRITERION - FULL TREE*/
proc hpsplit data=housing seed=304576;
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow RSS;
partition rolevar=selected(train="1");
run;

```



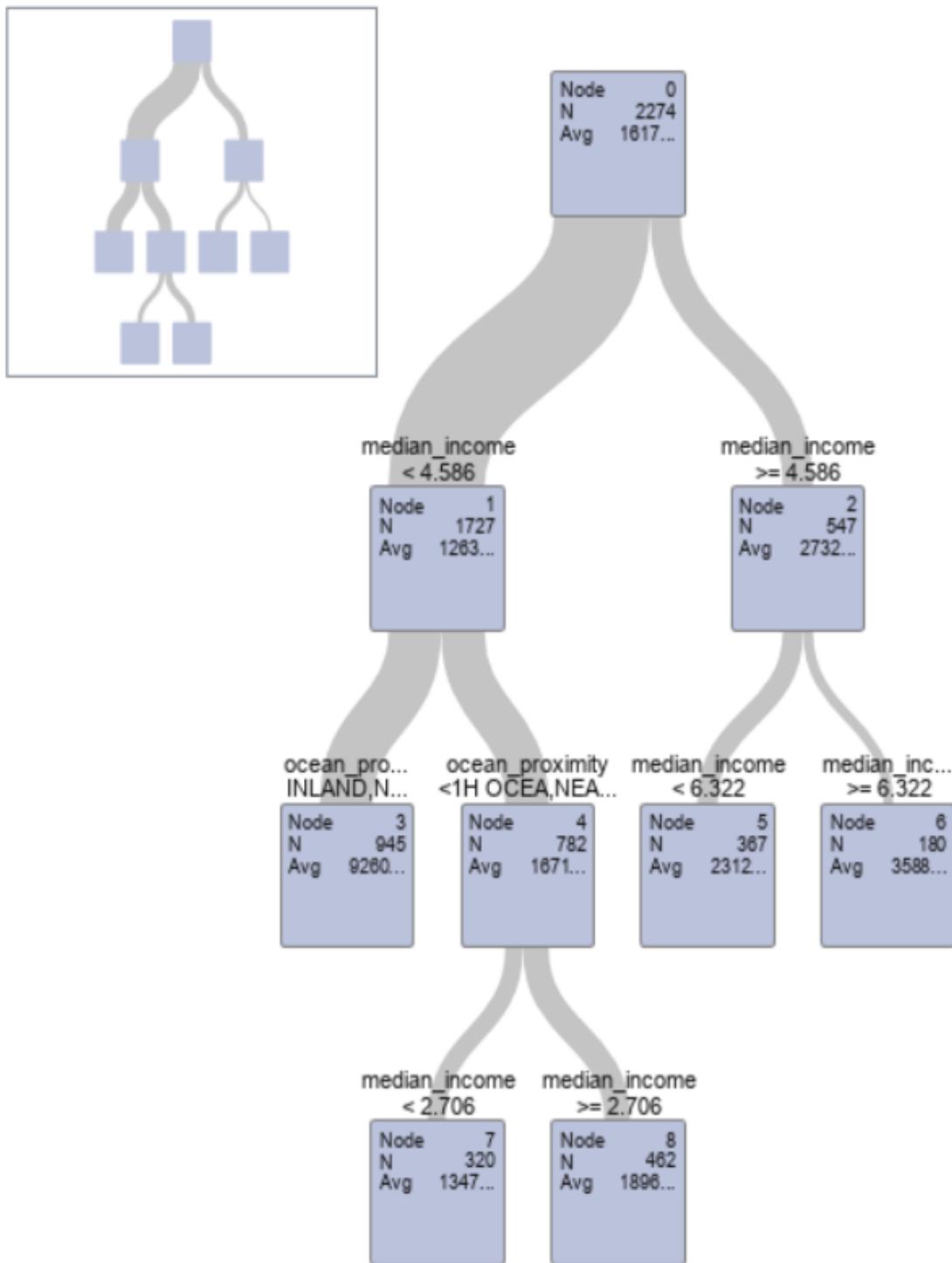
As part of the output, SAS produces a cost-complexity analysis plot. In this plot, the Average ASE (average squared error=  $RSS/n$  where  $n$  is the sample size of the training set) is plotted against the number of leaves in a tree (top scale on the  $x$ -axis). The additional scale on the  $x$ -axis (at the bottom) is for the cost-complexity parameter. To compute Average ASE, SAS does (by default) **10-fold cross-validation** by randomly dividing the training set into 10 equal parts and fitting the tree iteratively 10 times, every time holding 1/10th of the data as a **validation set**. ASE is computed for each of the 10 validation sets and then averaged.

As suggested by the plot, the global minimum corresponds to 25 leaf nodes, but this is obviously a very large tree (typically with very poor predictive accuracy). As suggested by Breiman, et. al (1984), it is more reasonable to prune a tree to the number of leaves corresponding to the smallest value of Average ASE below one standard error. The standard error is a standard deviation of the 10 ASE values computed during the cross-validation process. On the plot, the 1-SE line is the horizontal line, and the first value below it corresponds to 5 leaves. Next, we fit a pruned tree with 5 leaf nodes, using the cost-complexity pruning method.

```
/*RSS SPLITTING AND COST-COMPLEXITY PRUNING*/
proc hpsplit data=housing;
class ocean_proximity;
model median_house_value = housing_median_age total_rooms total_bedrooms
population households median_income ocean_proximity;
grow RSS;
prune costcomplexity(leaves=5);
partition rolevar=selected(train="1");
output out=predicted;
ID selected;
run;
```

Studying the tree on the next page, we can see that all the splits are done on two predictors only. The first split is done with respect to the median income ( $<4.586$ , 1727 rows vs.  $\geq 4.586$ , 547), then the latter node is split on the median income again ( $<6.322$ , 367 rows vs.  $\geq 6.322$ , 180 rows), and the former node is split on ocean proximity (inland and near ocean, 945 rows vs. others, 782 rows). The latter node is also split on median income ( $<2.706$ , 320 rows vs.  $\geq 2.706$ , 462 rows). There are 5 leaf nodes in this pruned tree.

## Subtree Starting at Node=0



Further, the data set "predicted" contains predicted responses for both training and testing data (identified by the variable "selected"). We limit the data to the testing set and compute proportions of predictions within 10%, 15%, and 20% of the observed values.

```
/*COMPUTING PREDICTION ACCURACY FOR TESTING DATA*/
data test;
set predicted;
if(selected="0");
keep _leaf_ median_house_value P_median_house_value;
run;

data accuracy;
set test;
if(abs(median_house_value-P_median_house_value)<0.10*median_house_value)
then ind10=1; else ind10=0;
if(abs(median_house_value-P_median_house_value)<0.15*median_house_value)
then ind15=1; else ind15=0;
if(abs(median_house_value-P_median_house_value)<0.20*median_house_value)
then ind20=1; else ind20=0;
run;

proc sql;
select mean(ind10) as accuracy10, mean(ind15) as accuracy15,
mean(ind20) as accuracy20
from accuracy;
quit;
```

accuracy10	accuracy15	accuracy20
0.257042	0.362676	0.471831

From here, we can see that roughly 27.5% of predictions are within 10%, 36.3% are within 15%, and 47.2% are within 20% of the true observed values in the testing data set.  $\square$

## Classification Tree

A classification tree is a decision tree for a categorical (or even binary) target (response) variable. In classification trees, a natural alternative to the RSS is the **classification error rate**, defined

as the fraction of observations in a region that do not belong to the most common class (that is, are misclassified by the tree). Two splitting methods are usually used, one is based on the Gini impurity index and the other is based on entropy.

## Gini Impurity Index

The following **Gini impurity index** is commonly used in practice:  $G = \sum_{k=1}^K p_k(1 - p_k)$  where  $p_k$  is the proportion of observations in the  $k$ th class (that is, the probability of randomly picking a data point in the  $k$ th class).

**Historical Note.** The Gini impurity index is named after an Italian statistician Corrado Gini who proposed the idea in his 1912 paper "Variability and Mutability".

**Example.** Suppose we observe 10 data points, 5 of which we color blue and the other 5 we color green (see the picture). Suppose we randomly pick a data point and then randomly classify it as blue or green according to the class distribution in the data set. That is, we classify it as blue (or green) with probability  $5/10 = 1/2$ , since we have 5 data points of each color. What's the probability we classify our data point incorrectly?

$$\mathbb{P}(\text{pick blue, classify blue}) = (1/2)(1/2) = 1/4,$$

$$\mathbb{P}(\text{pick blue, classify green}) = (1/2)(1/2) = 1/4,$$

$$\mathbb{P}(\text{pick green, classify blue}) = (1/2)(1/2) = 1/4,$$

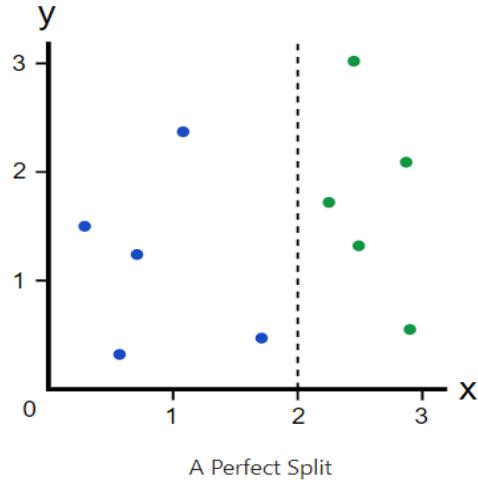
$$\mathbb{P}(\text{pick green, classify green}) = (1/2)(1/2) = 1/4.$$

Therefore, the probability of misclassification is  $\mathbb{P}(\text{pick blue, classify green}) + \mathbb{P}(\text{pick green, classify blue}) = 1/4 + 1/4 = 1/2 = 0.5$ .

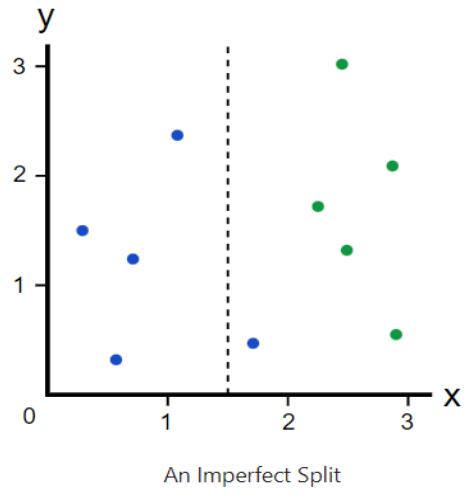
The Gini impurity index is computed as  $G = \mathbb{P}(\text{pick blue})(1 - \mathbb{P}(\text{pick blue})) + \mathbb{P}(\text{pick green})(1 - \mathbb{P}(\text{pick green})) = (0.5)(1 - 0.5) + (0.5)(1 - 0.5) = 0.25 + 0.25 = 0.5$  and coincides with the probability of misclassification.

If we now make a "perfect" vertical split at  $X = 2$  (see the figure below), then 100% of blue dots will be classified correctly as blue and 100% of green dots will be classified correctly as green, so the Gini impurity index for the region on the left is  $G_{\text{left}} = \mathbb{P}(\text{pick blue})(1 - \mathbb{P}(\text{pick blue})) + \mathbb{P}(\text{pick green})(1 - \mathbb{P}(\text{pick green})) = (1)(1 - 1) + (0)(1 - 0) = 0$ . The Gini index for the region on the right is  $G_{\text{right}} = \mathbb{P}(\text{pick blue})(1 - \mathbb{P}(\text{pick blue})) + \mathbb{P}(\text{pick green})(1 - \mathbb{P}(\text{pick green})) = (0)(1 - 0) + (1)(1 - 1) = 0$ . Thus, the "perfect" split turned a data set with a 0.5 impurity into two branches with zero impurity,

which is the lowest and the best possible impurity.



To illustrate further, consider now an "imperfect" split at  $X = 1.5$  where one blue data point falls into the region on the right (as seen in the picture below). The region on the left has only blue data points, so we know that  $G_{left} = 0$ . The region on the right has 1 blue and 5 green data points, and hence,  $G_{right} = (1/6)(1 - 1/6) + (5/6)(1 - 5/6) = 10/36 = 0.2778$ .



Finally, the **quality of the split** is determined by weighting the impurity of each region (branch) by how many data points it has. Since the region on the left has 4 data points and the region on the right has 6, we get  $(0.4)(0) + (0.6)(0.2778) = 0.1667$ . Thus, the amount of impurity that is

"removed" by this split is  $0.5 - 0.1667 = 0.3333$ . This value is called the **Gini gain**. The Gini gain for the "perfect" split is  $0.5 - ((0.5)(0) + (0.5)(0)) = 0.5$ . When training a decision tree, the best split is chosen by maximizing the Gini gain. The larger the Gini gain, the better the split.  $\square$

## Cross-entropy Loss Function

An alternative to the Gini impurity index is the **cross-entropy** (or **cross-entropy loss function** or **Shannon's entropy**) defined by the formula:

$$E = \begin{cases} - \sum_{k=1}^K p_k \ln(p_k), & \text{if } 0 < p_k \leq 1, \\ 0, & \text{if } p_k = 0. \end{cases}$$

**Historical Note.** Claude Shannon (1916-2001) was an American mathematician, electrical engineer, and cryptographer and is known as a "father of information theory".

**Example.** In our example, the cross-entropy for the entire data set is  $E = -(0.5)\ln(0.5) - (0.5)\ln(0.5) = -\ln(0.5) = 0.6931$ . For the "perfect" split, the cross-entropy for the left region is  $E_{left} = -(1)\ln(1) - 0 = 0$ , and so it is for the right region:  $E_{right} = 0 - (1)\ln(1) = 0$ . For the "imperfect" split, the cross-entropy for the left region is  $E_{left} = -(1)\ln(1) - 0 = 0$  and that for the right region is  $E_{right} = -(1/6)\ln(1/6) - (5/6)\ln(5/6) = 0.4506$ . A split is better if the reduction in the cross-entropy is larger. For the "perfect" split, the cross-entropy is reduced by  $0.6931 - [(0.4)(0) + (0.6)(0)] = 0.6931$ , whereas for the "imperfect split", it is  $0.6931 - [(0.4)(0) + (0.6)(0.4506)] = 0.4227 < 0.6931$ , so the "perfect" split wins.

With the cross-entropy loss function, the amount of impurity that is "removed" by a split is termed the **information gain**. The splitting algorithm that uses entropy and information gain as the metric is called the **Iterative Dichotomiser 3 (ID3) algorithm**. A successor of the ID3 algorithm (a recognized improved version of ID3) is **C4.5 algorithm** that uses entropy and gain ratio as the measures. The **gain ratio** is defined by the ratio of information gain and **split information**. How to calculate split information is easier to explain by example.

**Example.** In the above example, the "imperfect" split can be summarized as follows:

Split	Blue	Green	Total
left	4	0	4
right	1	5	6

Split Information =  $-\frac{4}{10} \ln \frac{4}{10} - \frac{6}{10} \ln \frac{6}{10} = 0.673012$ , and Gain Ratio = Information Gain / Split Information =  $0.4227 / 0.673012 = 0.628072$ . Gain ratios are compared for all candidate variables, and the one with the largest value is selected for the next split. Note that the gain ratio penalizes

having too many branches that a split would result in (that is, splitting in a high number of branches results in high information gain, but the split information also increases, so the gain ratio reduces the bias).  $\square$

**Historical Note.** The ID3 algorithm was invented by John Ross Quinlan in 1979, and in 1993 he published "C4.5 Programs for Machine Learning" where he introduced the C4.5 algorithm.

**Remark.** Note that at any given node, there may be a number of splits on different variables, all of which give almost the same decrease in impurity. Since data are noisy, the choice between competing splits is almost random. However, choosing an alternative split that is almost as good will lead to a different evolution of the tree from that node downward. That's why when a classification tree is developed, a seed should be specified for reproducibility of results.

## CONFUSION MATRIX

For a binary classification tree, we used the correct classification rate as a measure of model performance. Traditionally, other quantities are also used. We define them below. Suppose, hypothetically speaking, of 100 observations in a testing set, 50 yes's are predicted correctly, 10 yes's are predicted incorrectly, 35 no's are predicted correctly, and 5 no's are predicted incorrectly. We summarize this information in the following table (called **confusion matrix** or **classification matrix**):

	True "Yes"	True "No"	Total
Predicted "Yes"	50	5	55
Predicted "No"	10	35	45
Total	60	40	100

For simplicity of notation, correctly predicted yes's are called "true positive" (TP), incorrectly predicted yes's are called "false negative" (FN), correctly predicted no's are called "true negative" (TN), and incorrectly predicted no's are called "false positive" (FP).

**Example.** In our example,  $TP = 50$ ,  $FN = 10$ ,  $TN = 35$ , and  $FP = 5$ .  $\square$

The numbers of cases in these categories are used to calculate various measures of model fit:

- **Accuracy (or Correct Classification Rate):** Overall, how often is the classifier correct?

$$(TP + TN)/Total = (TP + TN)/(TP + TN + FP + FN) = (50 + 35)/100 = 0.85$$

- **Misclassification Rate:** Overall, how often is it wrong?

$$(FP + FN)/Total = (FP + FN)/(TP + TN + FP + FN) = (5 + 10)/100 = 0.15 = 1 - Accuracy$$

- **True Positive Rate (or Sensitivity or Recall):** When it's actually yes, how often does it predict yes?

$$TP/True\ yes = TP/(TP + FN) = 50/60 = 0.8333$$

- **False Negative Rate (FNR):** When it's actually yes, how often does it predict no?

$$FN/True\ yes = FN/(TP + FN) = 10/60 = 0.1667 = 1 - Sensitivity$$

- **True Negative Rate (or Specificity):** When it's actually no, how often does it predict no?

$$TN/True\ no = TN/(FP + TN) = 35/40 = 0.875$$

- **False Positive Rate (FPR):** When it's actually no, how often does it predict yes?

$$FP/True\ no = FP/(FP + TN) = 5/40 = 0.125 = 1 - Specificity$$

- **Positive Predictive Value (PPV, or Precision):** When it predicts yes, how often is it correct?

$$TP/Predicted\ yes = TP/(TP + FP) = 50/55 = 0.9091$$

- **Negative Predictive Value (NPV):** When it predicts no, how often is it correct?

$$TN/Predicted\ no = TN/(FN + TN) = 35/45 = 0.7778$$

These definitions can be presented in a theoretical confusion matrix:

	True "Yes"	True "No"	
Predicted "Yes"	True Positive (TP)	False Positive (FP)	Precision $= \frac{TP}{TP + FP}$
Predicted "No"	False Negative (FN)	True Negative (TN)	Negative Predictive Value $= \frac{TN}{FN + TN}$
	Sensitivity $= \frac{TP}{TP + FN}$ , False Negative Rate = $1 - \text{Sensitivity}$	Specificity $= \frac{TN}{FP + TN}$ , False Positive Rate = $1 - \text{Specificity}$	Accuracy $= \frac{TP + TN}{TP + TN + FP + FN}$ , Misclassification Rate $= 1 - \text{Accuracy}$

Another performance measure that is often utilized is the F1-score. It combines recall and precision into a single measure.

- **F1-score:** It is a harmonic mean of sensitivity and precision and can be calculated as follows:

$$\begin{aligned} \text{F1-score} &= \frac{2}{\frac{1}{sensitivity} + \frac{1}{precision}} = \frac{2}{\frac{TP+FN}{TP} + \frac{TP+FP}{TP}} \\ &= \frac{2TP}{2TP + FN + FP} = \frac{(2)(50)}{(2)(50) + 10 + 5} = \frac{100}{115} = 0.869565. \end{aligned}$$

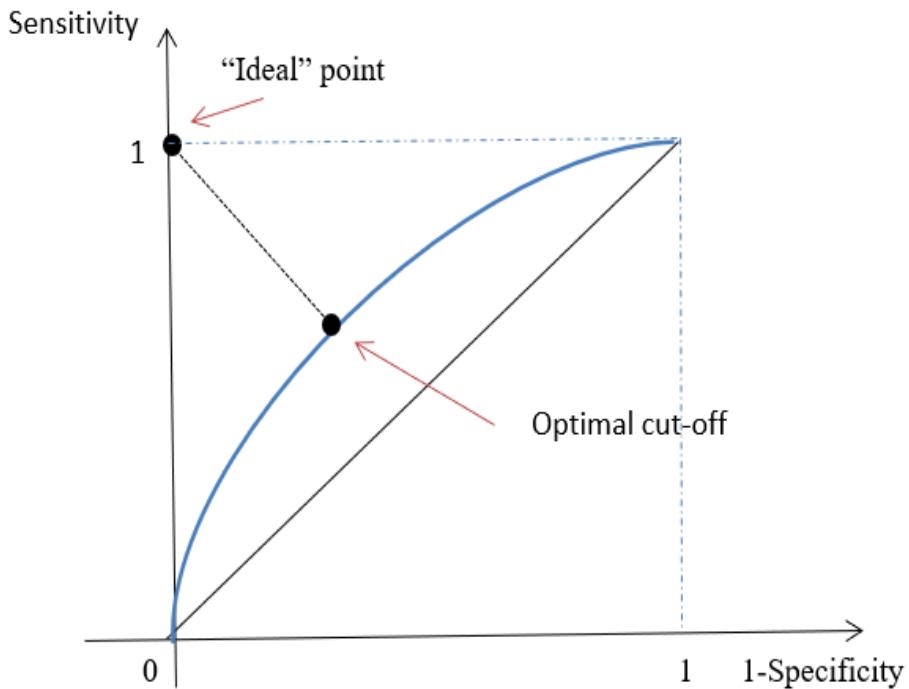
## RECEIVER OPERATING CHARACTERISTIC (ROC) CURVE

Consider the situation of a binary classification tree. We are given actual observations (yes/no) and predicted probabilities of yes/no. Before we introduced a cut-off, a number between 0 and 1, such that if the predicted probability of "yes" is above the cut-off, then we assume that the predicted value is "yes". We used cut-offs ranging between 0.01 and 0.99 with a step of 0.01 to choose the optimal cutoff that maximizes the true (correct) classification rate (equivalently, minimizes, the

misclassification rate). In the previous section, we computed the performance measures assuming the cut-off is 0.5.

A more sophisticated approach relies on the Receiver Operating Characteristic (ROC) curve, schematically presented in the figure below. A ROC curve is a plot of sensitivity (true positive rate) against 1-specificity (false positive rate) for different cut-off points. The cut-off points are often termed *classification thresholds*. A ROC curve connects the origin (0, 0) and the point (1, 1) as a curve that lies above the bisector. Note that the bisector represents a segment on which both sensitivity and specificity are equal to 0.5, corresponding to a random guess.

An ROC curve shows a trade-off between sensitivity and specificity, that is, an increase in sensitivity is accompanied by a decrease in specificity. These two quantities are known to work reciprocally.

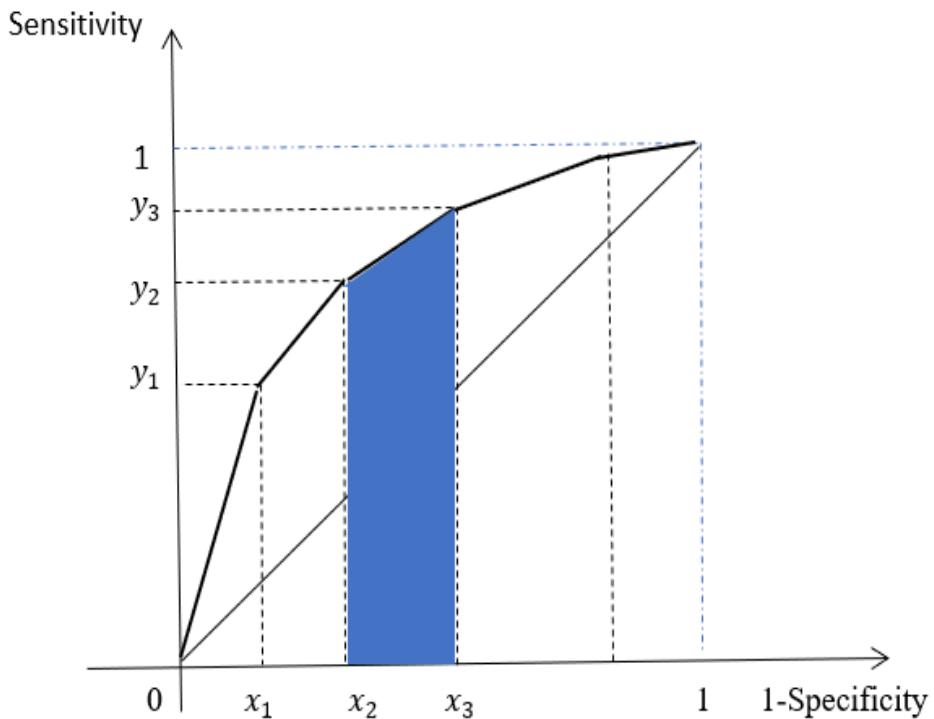


Define the point (0,1) as the “ideal” point. Indeed, at this point sensitivity (true positive rate) and specificity (true negative rate) would both be equal to one, which is clearly a hypothetical situation. Nonetheless, the point on the ROC curve closest to this “ideal” point gives the optimal cut-off for the binary classification.

## Area Under the ROC Curve

Often in practice another model performance measure is computed. It is the **area under the curve (AUC)** (or **area under the ROC curve (AUROC)**). The larger the area under the ROC curve, the further the curve is from the bisector, and thus a higher value of AUC indicates a better overall fit of the model (for any classification threshold).

In theory, the ROC curve is smooth, and computing AUC would involve calculus. In practice, however, the cut-offs are chosen on a discrete scale, so the fitted ROC curve is piece-wise linear, and thus, the area can be computed as the sum of areas of the trapezoids (see the figure below). The height of each trapezoid is the distance between two distinct consecutive values of  $1 - \text{Specificity}$ , and the top and bottom sides are distinct consecutive values of  $\text{Sensitivity}$ . Hence, the formula for the area of one trapezoid is  $((1 - \text{Specificity})_2 - (1 - \text{Specificity})_1)(\text{Sensitivity}_1 + \text{Sensitivity}_2)/2$ .



## MULTINOMIAL CLASSIFICATION TREE

A **multi-class** (or **multinomial**) **classification tree** classifies observations into one of three or more classes. The same algorithms as for the binary classification tree apply to this case as well. The output for prediction contains predicted probabilities of each of the multiple classes. We assume that the class with the highest predicted probability is the class actually predicted by the

fitted model.

## Performance Measures for Individual Classes

The performance measures used for binary classification can be extended to a multi-class classification. Unlike binary classification, there are no positive or negative classes but we can compute TP, TN, FP, and FN for each individual class.

For example, in a data set, there are 50 greens, 40 blues, and 10 reds. The predicted colors are summarized in the following confusion matrix:

True color	Predicted Color		
	green	blue	red
green	35	5	10
blue	5	30	5
red	3	2	5

We consider each color separately and compute all the performance measures. For the green color, the 2-by-2 confusion matrix has the form:

True color	Predicted Color	
	green	not green
green	35	15
not green	8	42

Now we compute the performance measures for this confusion matrix:

$$TP = 35, \quad TN = 42, \quad FP = 8, \quad FN = 15,$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = (35 + 42)/100 = 0.77,$$

$$\text{Misclassification Rate} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \text{Accuracy} = 0.23,$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} = 35/(35 + 15) = 0.70,$$

$$\text{FNR} = \frac{FN}{TP + FN} = 1 - \text{Sensitivity} = 0.30,$$

$$\text{Specificity} = \frac{TN}{FP + TN} = 42/(8 + 42) = 0.84,$$

$$\text{FPR} = \frac{FP}{FP + TN} = 1 - \text{Specificity} = 0.16,$$

$$\text{Precision} = \frac{TP}{TP + FP} = 35/(35 + 8) = 0.813953,$$

$$\text{NPV} = \frac{TN}{FN + TN} = 42/(15 + 42) = 0.736842,$$

$$\text{F1-score} = \frac{2TP}{2TP + FN + FP} = \frac{(2)(35)}{(2)(35) + 15 + 8} = 0.752688.$$

For the blue color, the 2-by-2 confusion matrix is

True color	Predicted Color	
	blue	not blue
blue	30	10
not blue	7	53

The performance measures for this binary classification are

$$TP = 30, \quad TN = 53, \quad FP = 7, \quad FN = 10,$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = (30 + 53)/100 = 0.83,$$

$$\text{Misclassification Rate} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \text{Accuracy} = 0.17,$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} = 30/(30 + 10) = 0.75,$$

$$\text{FNR} = \frac{FN}{TP + FN} = 1 - \text{Sensitivity} = 0.25,$$

$$\text{Specificity} = \frac{TN}{FP + TN} = 53/(7 + 53) = 0.95,$$

$$\text{FPR} = \frac{FP}{FP + TN} = 1 - \text{Specificity} = 0.05,$$

$$\text{Precision} = \frac{TP}{TP + FP} = 30/(30 + 7) = 0.810811,$$

$$\text{NPV} = \frac{TN}{FN + TN} = 53/(10 + 53) = 0.84127,$$

$$\text{F1-score} = \frac{2TP}{2TP + FN + FP} = \frac{(2)(30)}{(2)(30) + 10 + 7} = 0.779221.$$

Finally, for the red color, the 2-by-2 confusion matrix is

True color	Predicted Color	
	red	not red
red	5	5
not red	15	75

Now we compute the performance measures for this confusion matrix

$$TP = 5, \quad TN = 75, \quad FP = 15, \quad FN = 5,$$

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = (5 + 75)/100 = 0.80,$$

$$\text{Misclassification Rate} = \frac{FP + FN}{TP + TN + FP + FN} = 1 - \text{Accuracy} = 0.20,$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} = 5/(5 + 5) = 0.50,$$

$$\text{FNR} = \frac{FN}{TP + FN} = 1 - \text{Sensitivity} = 0.50,$$

$$\text{Specificity} = \frac{TN}{FP + TN} = 75/(15 + 75) = 0.8333,$$

$$\text{FPR} = \frac{FP}{FP + TN} = 1 - \text{Specificity} = 0.1667,$$

$$\text{Precision} = \frac{TP}{TP + FP} = 5/(5 + 15) = 0.25,$$

$$\text{NPV} = TN/(FN + TN) = 75/(5 + 75) = 0.9375,$$

$$\text{F1-score} = \frac{2TP}{2TP + FN + FP} = \frac{(2)(5)}{(2)(5) + 5 + 15} = 0.3333.$$

## Micro Measures

Turning now to the multinomial model, we compute performance measures based on total  $TP = 35 + 30 + 5 = 70$ , total  $TN = 42 + 53 + 75 = 170$ , total  $FP = 8 + 7 + 15 = 30$ , and total  $FN = 15 + 10 + 5 = 30$ . These are global measures for the whole model, called **micro-averaged** measures. The micro-averaged measures are:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} = (70 + 170)/300 = 0.80,$$

$$\text{Misclassification Rate} = 1 - \text{Accuracy} = 0.20,$$

$$\text{Sensitivity} = \frac{TP}{TP + FN} = 70/(70 + 30) = 0.70,$$

$$\text{FNR} = 1 - \text{Sensitivity} = 0.30,$$

$$\text{Specificity} = \frac{TN}{FP + TN} = 170/(30 + 170) = 0.85,$$

$$\text{FPR} = 1 - \text{Specificity} = 0.15,$$

$$\text{Precision} = \frac{TP}{TP + FP} = 70/(70 + 30) = 0.70,$$

$$\text{NPV} = \frac{TN}{FN + TN} = 170/(30 + 170) = 0.85,$$

$$\text{F1-score} = \frac{2TP}{2TP + FN + FP} = \frac{(2)(70)}{(2)(70) + 30 + 30} = 0.70.$$

## Macro Measures

**Macro** measures are computed as an arithmetic average of individual measures for each class. The macro measures are:

$$\text{Accuracy} = (0.77 + 0.83 + 0.80)/3 = 0.80,$$

$$\text{Misclassification Rate} = 1 - \text{Accuracy} = 0.20,$$

$$\text{Sensitivity} = (0.70 + 0.75 + 0.50)/3 = 0.65,$$

$$\text{FNR} = 1 - \text{Sensitivity} = 0.35,$$

$$\text{Specificity} = (0.84 + 0.95 + 0.8333)/3 = 0.87,$$

$$\text{FPR} = 1 - \text{Specificity} = 0.13,$$

$$\text{Precision} = (0.813953 + 0.810811 + 0.25)/3 = 0.624921,$$

$$\text{NPV} = (0.736842 + 0.84127 + 0.9375)/3 = 0.838537,$$

$$\text{F1-score} = (0.752688 + 0.779221 + 0.3333)/3 = 0.621736.$$

## Weighted Macro Measures

The **weighted macro** measures are weighted means of the measures for individual classes, where weights are proportional to the total number of samples in the class. There are 50 greens, 40 blues, and 10 reds, therefore, the weights are  $50/(50+40+10)=0.5$  for greens,  $40/100=0.4$  for blues, and  $10/100=0.1$  for reds. The weighted macro measures are computed as:

$$\text{Accuracy} = (0.77)(0.5) + (0.83)(0.4) + (0.80)(0.1) = 0.797,$$

$$\text{Misclassification Rate} = 1 - \text{Accuracy} = 0.203,$$

$$\text{Sensitivity} = (0.70)(0.5) + (0.75)(0.4) + (0.50)(0.1) = 0.70,$$

$$\text{FNR} = 1 - \text{Sensitivity} = 0.30,$$

$$\text{Specificity} = (0.84)(0.5) + (0.95)(0.4) + (0.8333)(0.1) = 0.88333,$$

$$\text{FPR} = 1 - \text{Specificity} = 0.11667,$$

$$\text{Precision} = (0.813953)(0.5) + (0.810811)(0.4) + (0.25)(0.1) = 0.756301,$$

$$\text{NPV} = (0.736842)(0.5) + (0.84127)(0.4) + (0.9375)(0.1) = 0.798679,$$

$$\text{F1-score} = (0.752688)(0.5) + (0.779221)(0.4) + (0.3333)(0.1) = 0.721362.$$