PREDICTION
Regression Trees/CART

Tamer Çetin

Regression Trees: Motivation

- We want to predict the price of used cars as the function of their age and other features.
- You need to specify a model that includes the most important interactions and nonlinearities of those features.
- The challenges you will face:
  - Model selection,
  - Functional forms of models,
  - Feature selection and the existence of interactions,
  - Facing/avoiding overfitting!

Regression Trees: Motivation

- We have learned that we cannot use all possible models!
- But we have learned that we can use shrinkages to get to a less complex linear regression model from a more complex model:
  - which helps avoid overfitting!
- In this lecture, we will introduce regression trees as an alternative prediction model to linear regression model for prediction purposes that:
  - can find the most important predictors and their interactions,
  - and can approximate any functional form automatically.

Regression Trees: Motivation

- Regression trees split data into small bins (sub-samples) by the value of the x variables.
- For the quantitative y, they use the average y value in those small datasets to predict $\hat{y}$.
- In this lecture, we will focus on regression tree model and the most widely used algorithm to build a regression tree model, which is usually called CART (Classification and Regression Trees).
- Regression tree is an intuitively appealing method to model nonlinearities and interactions among the x variables, but it is rarely used for prediction itself because it is prone to overfit original data.

Regression Trees: Motivation

- But still we will learn the logic behind regression trees because it forms the basic element of very powerful prediction methods such as random forests we will focus on in the next lecture.
- Using the case study ‘predicting used car value with regression trees’ and ‘used-cars’ dataset, we will illustrate;
  - How we can build a regression tree model with the help of a CART algorithm,
  - How such a model would overfit original data.

Regression Trees: Learning Outcomes

- After working through this lecture, you should be able to;
  - Build regression trees using the CART algorithm to predict y with the help of x variables.
  - Understand how to the CART algorithm approximates nonlinear functional forms and finds important interactions,
  - Understand why regression trees likely overfit data,
  - Understand how pruning works and why it reduces overfitting,
  - Understand the similarities and differences between using the CART algorithm to grow a regression tree and building a linear regression.

The Case for Regression Trees

- Note that we have a constraint that;
    - We cannot use all the possible models for prediction.
    - Thus, we cannot find the best regression model for prediction by trying all models and evaluating their performance by CV, for instance.
- In linear regression analysis, we learnt that LASSO can help us find the best prediction by variable selection and regularization.
- In this lecture, we will use regression trees as a different approach;
    - To capture the most important variables,
    - To approximate the functional forms of the $x - y$ relationships whether they are linear or not,
    - And, to identify the most important interactions.

The Case for Regression Trees

- Because regression trees algorithm constructs a prediction model using a step-by-step process;
  - First, the most important x variables,
  - Then, going forward to include some other variables,
  - Approximating nonlinearities,
  - And, creating interactions among x variables in the process.
- The process ends up with a prediction that assigns $\hat{y}$ values to all observations depending on the values of the x variables included in the tree.

The Case for Regression Trees

- Regression tree modeling is an intuitive process but still can lead to a better prediction.
- One serious problem with regression trees is that regression trees tend to severely overfit training data and produce poor fits in test set.
    - This means that they tend to overfit original data and produce inferior predictions in live data!
- So, even though there are some ways to improve those shortcomings that we will introduce in this lecture, we, in practice, use regression trees to build blocks of much more successful prediction models such RFs.

Regression Trees: Basics

- The basic idea with regression trees is to split data into small bins using the values of x variables and predict y as the average value of y within those bins.
- Creating a regression tree is called building or growing a tree.
- Trees are created by the CART algorithm;
  - Which has no formula!
- The goal of the algorithm is to arrive at a set of bins, defined by the values of various x variables.
  - It is assumed that the average value of y in those bins is the best prediction of y!

Regression Trees: Basics

- By doing so, the defined bins represent the patterns of associations between x and y leading to the best prediction of y.
- In fact, the CART algorithm does not exactly target the goal of best prediction explicitly;
  - but it approximates that goal by going step by step, setting intermediate goals for each successive step that are easier to achieve than the ultimate goal of a best final prediction.
  - So each step of the CART algorithm consists of splitting data into smaller and smaller bins.
- We can explain the process using a single x variable.

Regression Trees: Basics

- The algorithm starts with the entire sample (the entire training sample in the case of CV).
  - This first step is called the top node of the tree splitting data into two parts!
- When x is binary, there is only one way to split the sample data along its values:
  - One part has all observations with $x = 0$,
  - The other part has all observations with $x = 1$.
- That would be all with a single x variable.
- When x is not binary, there are multiple ways to do the splitting.

Regression Trees: Basics

- For the continuous variables, the algorithm finds the split that makes the best prediction of y from among all potential splits using some criterion, which is done by a search algorithm.
- The result of the search algorithm is the x value along with the split improving the prediction the most!
- The value at which the split happens is the cutoff point for that particular x variable.
- Now, the first split produces two branches from the top node.
- At the end of each branch is one new node corresponding to two respective bins (subsamples).
- Then, the algorithm restarts for each bin and considers a subsequent split for each following the same approach as done in the previous step, and so on.

Regression Trees: Basics

- The process stops when a pre-defined stopping rule tells it to stop (we will talk about the stopping rules later).
- The set of nodes after the algorithm stops is called terminal nodes:
  - The collection of terminal nodes is also called the bucket!
- Each observation in data corresponds to one bin in the bucket;
  - Which is set of bins producing the prediction of y.
- The predicted $\hat{y}$ values are the average y values within each bin.
- The process stops when the algorithm arrives at a subsample where no further splits can improve the prediction.
- Accordingly, the number of splits may differ for different branches!

Measuring Fit and Stopping Rules

- The fit of the prediction from a regression tree can be measured in the same way as the fit of any prediction model.
- The end of the algorithm gives a predicted value $\hat{y}$ that corresponds to particular x values.
- To measure the fit we compare these $\hat{y}$ values to the actual y values.
- The best tree is found via cross-validation.
- One measure of fit often used for regression trees is called the **relative error**.

Measuring Fit and Stopping Rules

- It is calculated as the ratio of the sum of squared errors of the prediction and the total sum of squared deviation of y from its overall mean.
- This is, of course, simply $1 - R2$.
- Lower relative error means a higher R-squared, which implies better model fit.
- Alternatively, we can use MSE or its square root, RMSE. MSE is nothing else than the numerator of relative error.
- Thus, whether we use MSE, RMSE, R-squared, or relative error, we arrive at the same ranking of different predictions.

Measuring Fit and Stopping Rules

- The main issue with CART is overfitting.
- To decrease the likelihood of overfitting, the splitting algorithm stops at some point called the **stopping rule**
  - A condition, or a set of conditions, that defines when the algorithm stops, based on results from the most recent step.
- Data analysts use various stopping rules when growing a regression tree, such as the minimum number of observations in a bin for further splitting or the number of observations in any terminal node.
- One example is that the split needs to improve the R-squared by at least 0.001.
- Thus, this rule tells the algorithm to stop and make the bin a terminal node if splitting it would improve the R-squared by less than 0.001.

Regression Tree with Multiple Predictor Variables

- The idea of building a regression tree is the same as before:
  - finding the best split at each and every step.
- The new thing is that, here, the algorithm compares splits according to many variables, in addition to comparing splits along different values of the same variable.
- At each node, the algorithm finds the best potential split for each x variable, which includes finding the best cutoff value for each non-binary x, and then selects the x variable that produces the best split.
- This procedure is repeated for each and every node.

Pruning a Regression Tree

- A stopping rule or using tunning parameters may lead to underfitting.
- A better solution is called **pruning** the tree.
- A CART with pruning is a complicated procedure.
- First, we grow a large tree by applying a lenient stopping rule (such as a very small complexity parameter).
- Then, from that large tree, we start deleting splits at the end of the tree, one by one.
- As we do not know which final split is the least useful, we try deleting one split at a time at the last level, and re-evaluate the model fit in the test sample.

Pruning a Regression Tree

- We try this with all such splits, and, in this way, we eventually compare many trees that are smaller by just one node.
- The result is a tree that is one node smaller, and this tree has a new set of terminal nodes.
- Then we carry on, and delete one more split at the new terminal level using the same approach, and so on.
- We use cross-validation to evaluate the fit after each deletion.
- The process goes on until the fit is improved, and it stops when we can't improve the fit by deleting any of the nodes.

A Regression Tree is a Non-parametric Regression

- The regression tree is non-parametric because there are no intercept or slope coefficients or other parameters that would characterize how average y depends on the values of the x variables.
- Or, in other words, there is no formula that would link average y to the x variables.
- Instead, the result is the set of the bins that make up the terminal nodes, the values of the x variables that define those bins, and the corresponding $\hat{y}$ values.
- For example, with a single quantitative x variable, the regression tree approximates the $y^E = f(x)$ function by a step function, whatever the true form of that f function is.

Variable Importance

- Thus, after all, we do need to understand which x variables are the most important for our prediction, and how those x variables affect $\hat{y}$.
- With a few x variables and a single regression tree, we assess that by examining the tree itself.
- However, with many x variables, that approach is challenging.
- The most widely used measure data analysts use to assess the important of each x variable in the regression tree is called **variable importance**.
- It measures how much the fit of the prediction is improved when a particular x variable is used for splitting, summed across all splits in which that variable occurs.
- Variable importance is expressed as the share of fit improvement (MSE reduction) due to the particular x variable relative to the overall improvement of fit achieved by the regression tree (as opposed to a prediction that doesn't use any x variables).
- The graph expresses variable importance in relative terms – as percent of total reduction in MSE.

OLS vs CART

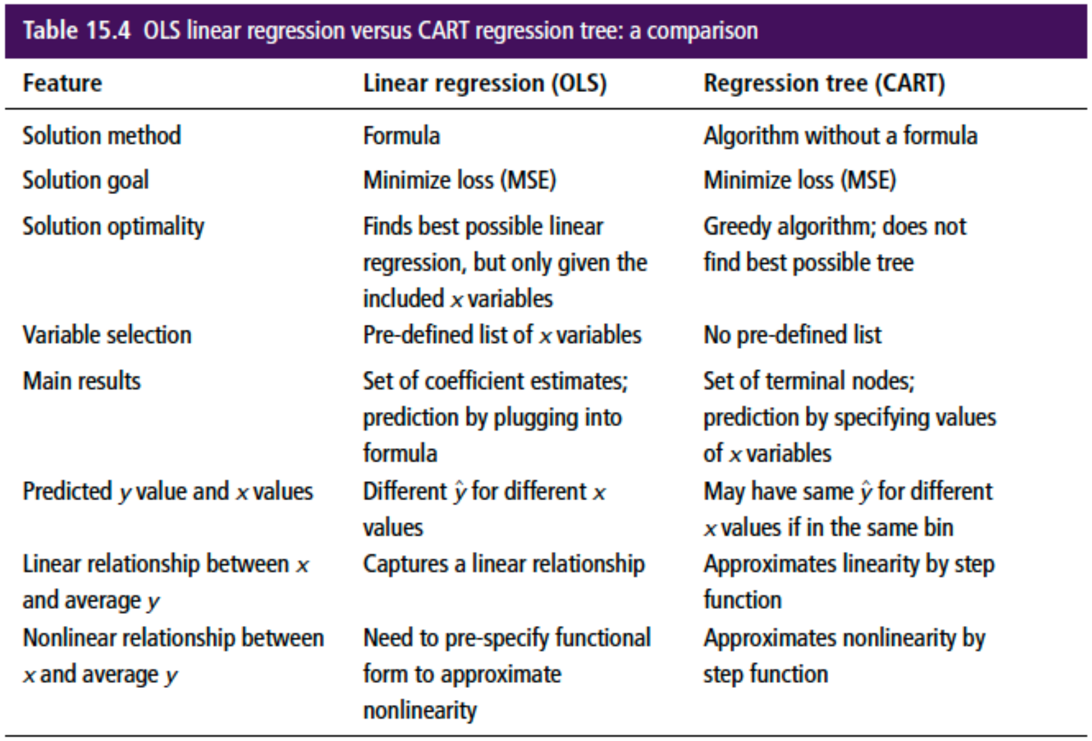


Table 15.4 OLS linear regression versus CART regression tree: a comparison

Feature	Linear regression (OLS)	Regression tree (CART)
Solution method	Formula	Algorithm without a formula
Solution goal	Minimize loss (MSE)	Minimize loss (MSE)
Solution optimality	Finds best possible linear regression, but only given the included *x* variables	Greedy algorithm; does not find best possible tree
Variable selection	Pre-defined list of *x* variables	No pre-defined list
Main results	Set of coefficient estimates; prediction by plugging into formula	Set of terminal nodes; prediction by specifying values of *x* variables
Predicted *y* value and *x* values	Different $\hat{y}$ for different *x* values	May have same $\hat{y}$ for different *x* values if in the same bin
Linear relationship between *x* and average *y*	Captures a linear relationship	Approximates linearity by step function
Nonlinear relationship between *x* and average *y*	Need to pre-specify functional form to approximate nonlinearity	Approximates nonlinearity by step function

Main Takeaways

- A regression tree is a method for predicting y based on x variables using an automated algorithm that approximates any functional form and any interaction between the x variables.
- The promise of regression trees and the CART algorithm is that they can find the best prediction of y using only the most important x variables, their most important interactions, and the most important aspects of potential nonlinear associations.
- A regression tree splits the sample into many bins according to values of the x variables and predicts y as the average within those bins.
- Regression trees can be thought of as non-parametric regressions.
- A regression tree is prone to overfitting the data even after pruning.
- For this reason, it is rarely used for prediction itself.

Random Forest and Boosting

- A regression tree can capture complicated interactions and nonlinearities for predicting a quantitative y variable, but it is prone to overfit the original data, even after appropriate pruning.
- It turns out, however, that combining multiple regression trees grown on the same data can yield a much better prediction.
- Such methods are called ensemble methods.
- There are many ensemble methods based on regression trees, and some are known to produce very good predictions.
- But these methods are rather complex and some of them are not straightforward to use.
- This section introduces two ensemble methods based on regression trees:
    - the random forest and boosting.

From a Tree to a Forest: Ensemble Methods

- Ensemble methods combine the results of many imperfect models to produce a prediction.
- Compared to a single model built to perfection, ensemble methods tend to produce much better predictions.

Random Forest

- The random forest is an ensemble prediction method in that it uses many regression trees that are grown to predict y with the help of variables x.
- The basic idea is to use the same data to grow many trees, each of which is very imperfect, and then combine those many imperfect trees into one prediction.
- The method that implements that basic idea is called **Bootstrap aggregation**, abbreviated as **bagging** (“bagg” for bootstrap aggregation).

Random Forest

- The bagging algorithm starts by taking bootstrap samples from the original data, say K of them, where K is a large number, usually in the hundreds (more on this later).
- Then, within each bootstrap sample, it grows a large tree, without pruning.
- Each tree is used to generate its own predictions.
- Then the algorithm takes these K prediction rules and combines them.
- In a training–test setup, it makes K predictions for each observation in the test sample, according to the terminal nodes of each of the K trees.
- The final step is aggregation: the final predicted y value is the simple mean of the K predicted y values for each observation.

Random Forest

- Bagging keeps the advantages of trees: each tree is built from scratch, and each approximates any function and selects the most important interactions.
- At the same time, the procedure helps avoid the disadvantages of a single tree.
- Although each individual prediction overfits the data of its own bootstrap sample, the average prediction is a lot less prone to overfitting the entire original data.
- Moreover, the average prediction is less sensitive to including or excluding observations with specific values, such as extreme values, because the different bootstrap samples include and exclude different observations.

An Introduction to Boosting and the GBM Model

- Whereas bagging in general, and the random forest in particular, aims at growing independent trees, boosting grows trees that build on each other.
- Then, similarly to bagging, it combines all those trees to make a prediction.
- The basic idea of boosting is to grow trees sequentially, using information from the previous tree to grow a better tree the next time.
- The information used from the previous tree is which observations were harder to predict.
- The new tree then puts more emphasis on fitting those observations.

An Introduction to Boosting and the GBM Model

- Typically, this is done by taking the residuals from the previous prediction and fitting a model on those residuals instead of the original y variable.
- Thus, prediction after having grown this next tree is not from the new tree only, but a combination of the new tree and the previous tree.
- Then, in the following step, the algorithm grows a yet newer tree building on that combined prediction, taking its residuals, and so on.
- The algorithm stops according to a stopping rule, such as the total number of trees grown.

An Introduction to Boosting and the GBM Model

- One particular method that uses this approach is called the **gradient boosting machine**, or **GBM**.
- The gradient part of its name refers to a search algorithm that it uses to find a better fit.
- The GBM, similarly to other boosting methods, has more tuning parameters than random forest.

An Introduction to Boosting and the GBM Model

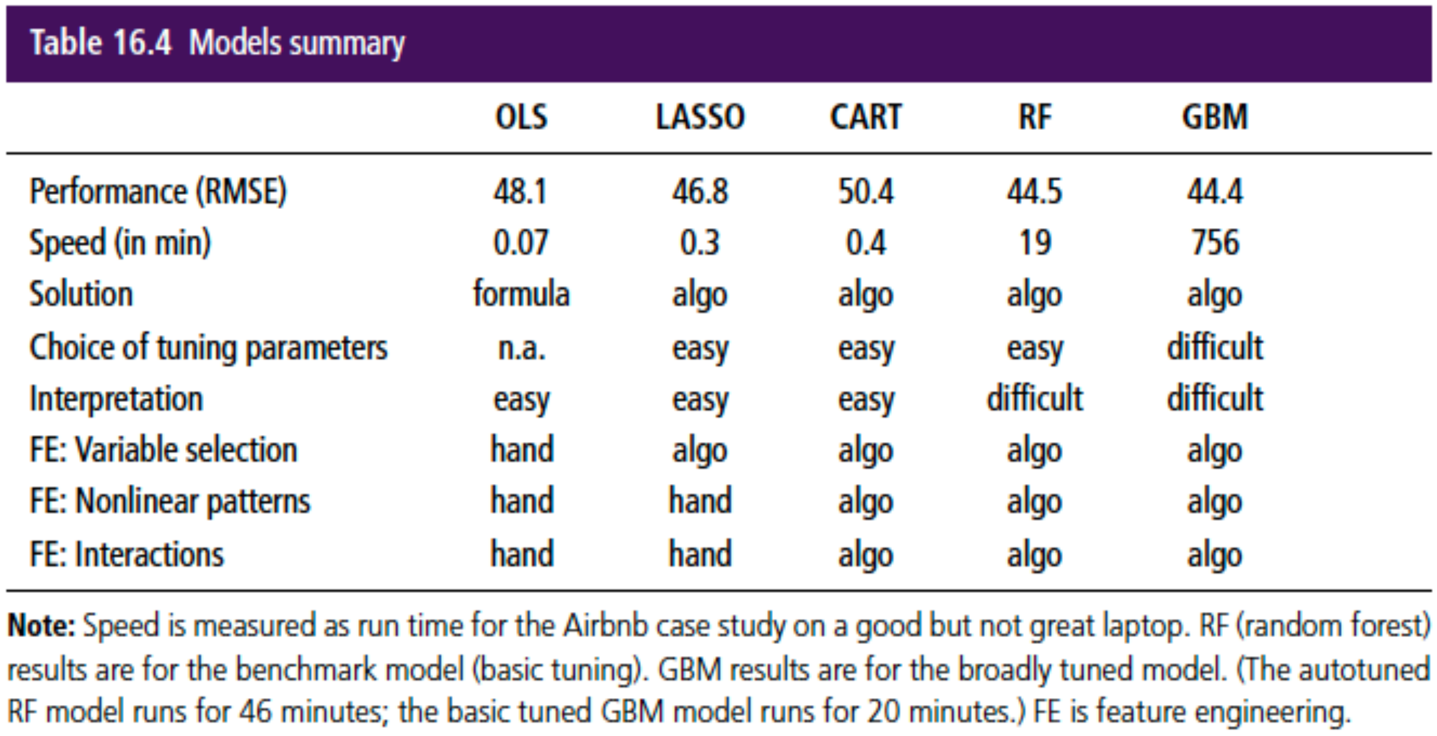


Table 16.4 Models summary

	OLS	LASSO	CART	RF	GBM
Performance (RMSE)	48.1	46.8	50.4	44.5	44.4
Speed (in min)	0.07	0.3	0.4	19	756
Solution	formula	algo	algo	algo	algo
Choice of tuning parameters	n.a.	easy	easy	easy	difficult
Interpretation	easy	easy	easy	difficult	difficult
FE: Variable selection	hand	algo	algo	algo	algo
FE: Nonlinear patterns	hand	hand	algo	algo	algo
FE: Interactions	hand	hand	algo	algo	algo

Note: Speed is measured as run time for the Airbnb case study on a good but not great laptop. RF (random forest) results are for the benchmark model (basic tuning). GBM results are for the broadly tuned model. (The autotuned RF model runs for 46 minutes; the basic tuned GBM model runs for 20 minutes.) FE is feature engineering.

Probability Prediction and Classification

- Sometimes our y variable is not quantitative.
- The most important case is when y is binary: $y = 1$ or $y = 0$.
- How can predict such a variable?
  - Probability prediction and classification
- Probability prediction means predicting the probability that $y = 1$, with the help of the predictor variables.
- Classification means predicting the binary y variable itself, with the help of the predictor variables:
  - putting each observation in one of the y categories, also called classes.

Predicting a Binary y: Probability Prediction and Classification

- Probability models aim to uncover the patterns of association between the probability of $y = 1$ and the x variables.
- **Probability predictions** aim to predict the probability that $y = 1$, conditional on the x variables.

$$\Pr[y = 1|x]$$

- Sometimes, we want each observation to be classified into one of the possible y values or **classes**.
- That's **classification**:
    - predicting whether $y = 1$ or $y = 0$.

The Practice of Predicting Probabilities

- We know conditional probabilities are expressed by
$$y^p = P[y = 1|x]$$
- We also know that the logit regression is a good probability model for prediction.
$$\hat{y}^p_{logit} = \Lambda(\hat{\beta}_0 + \hat{\beta}_1 x_1 + \hat{\beta}_2 x_2 + \cdots)$$
- One result of the Λ transformation is that the predicted probabilities are always greater than zero and less than one.
- To carry out an actual prediction we need to build models, select the best model, make the prediction using that model, and evaluate its performance.
- When predicting probabilities, the models to consider are logit models.

The Practice of Predicting Probabilities

- Feature engineering
- Functional forms
- Interactions
- Model selection
- Feature importance

Classification and the Confusion Table

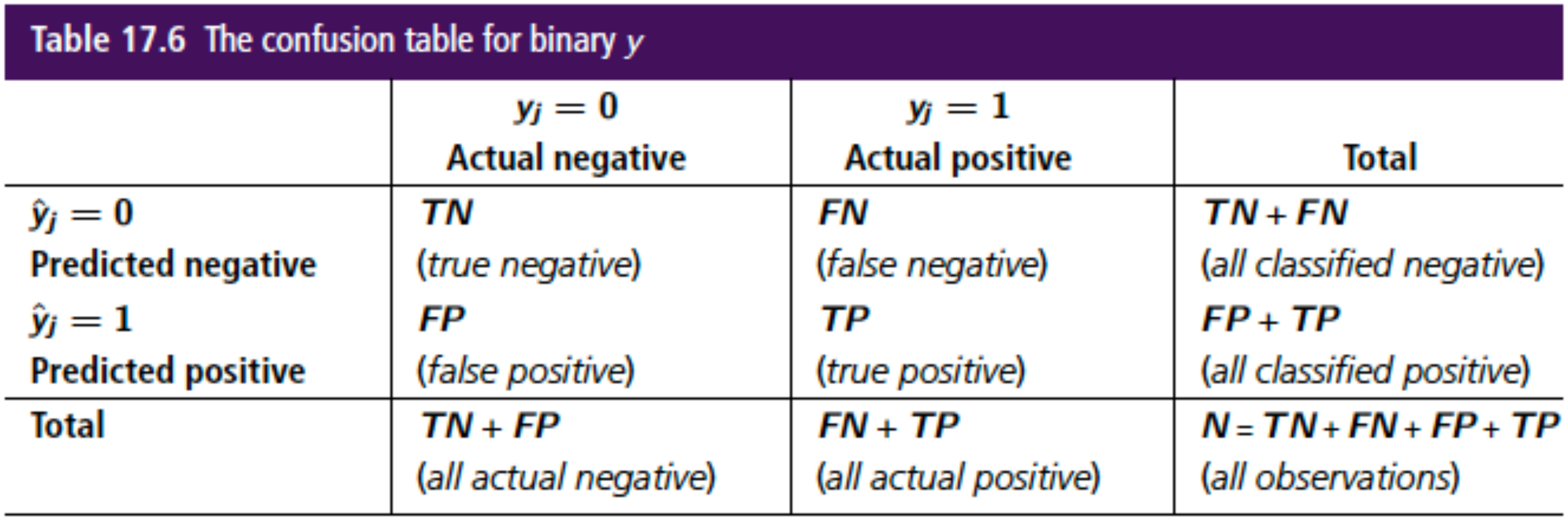


Table 17.6 The confusion table for binary y

	$y_j = 0$ **Actual negative**	$y_j = 1$ **Actual positive**	**Total**
$\hat{y}_j = 0$ **Predicted negative**	TN (*true negative*)	FN (*false negative*)	$TN + FN$ (*all classified negative*)
$\hat{y}_j = 1$ **Predicted positive**	FP (*false positive*)	TP (*true positive*)	$FP + TP$ (*all classified positive*)
Total	$TN + FP$ (*all actual negative*)	$FN + TP$ (*all actual positive*)	$N = TN + FN + FP + TP$ (*all observations*)

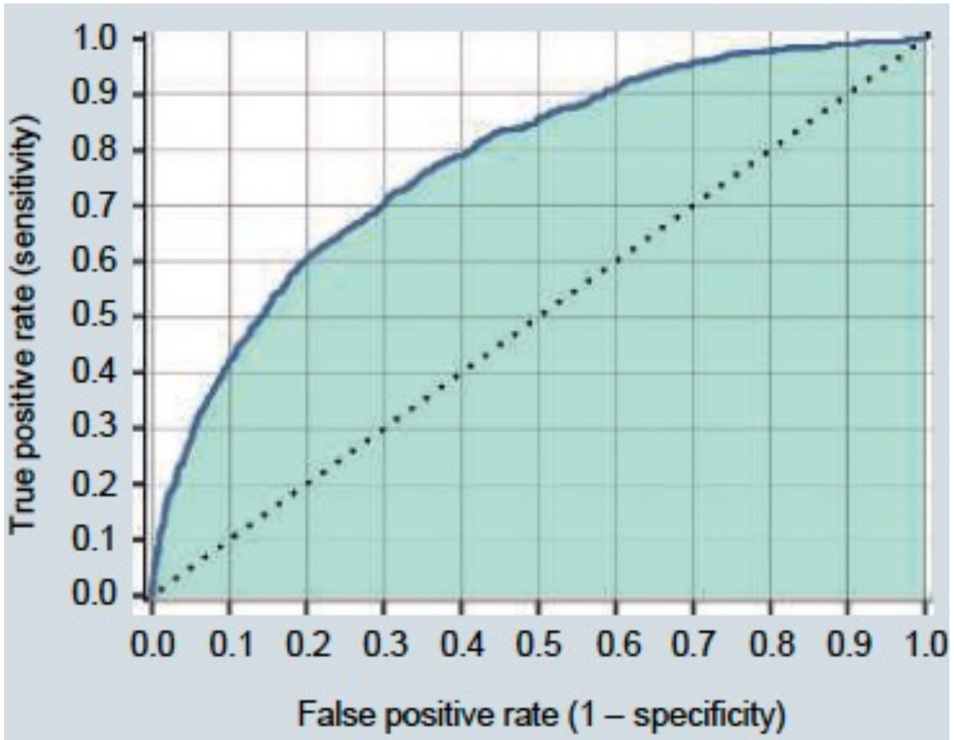
$$accuracy = \frac{TP + TN}{N}$$

$$specificity = P[\hat{y} = 0 | y = 0] = \frac{TN}{TN + FP} \qquad sensitivity = P[\hat{y} = 1 | y = 1] = \frac{TP}{TP + FN}$$

Illustrating the Trade-Off between Different Classification Thresholds: The ROC Curve

$$sensitivity = P[\hat{y} = 1 | y = 1] = \frac{TP}{TP + FN}$$

$$specificity = P[\hat{y} = 0 | y = 0] = \frac{TN}{TN + FP}$$



(b) Continuous ROC curve

Table 17.7 RMSE and AUC for various models of firm exit

Model	RMSE	AUC
Logit M1	0.374	0.738
Logit M2	0.366	0.771
Logit M3	0.364	0.777
Logit M4	0.362	0.782
Logit M5	0.363	0.777
Logit LASSO	0.362	0.768

Loss Function and Finding the Optimal Classification Threshold

- We need a threshold value to convert predicted probability into classification.
- We have seen that different thresholds lead to different frequencies of false positives and **false negatives**.
- The goal of classification is to use the predicted probabilities and find the value of the classification threshold that is best for the purpose of answering the question of the analysis.
- We need a loss function to do that.

Loss Function and Finding the Optimal Classification Threshold

- Similarly to any loss function, a **classification loss function** quantifies the consequences of decisions that are driven by the prediction.
- As we have seen, when classifying observations into classes of a binary y a false negative classification (FN) or making a **false positive** classification (FP).
- A loss function would have only two values here: a loss due to false negatives and a loss due to false positives.

$$\begin{aligned} E[loss] &= P[FN] \times loss(FN) + P[FP] \times loss(FP) \\ &= FN/N \times loss(FN) + FP/N \times loss(FP) \end{aligned}$$

Loss Function and Finding the Optimal Classification Threshold

- With predicted probabilities, we need to find the threshold value for classification that results in the smallest expected loss.
- That is the best threshold we can have, also called the **optimal classification threshold**.
- It turns out that, under ideal circumstances, we can compute the optimal classification threshold from the relative magnitude of the two losses.
- It is simply

$$optimal\ classification\ threshold = \frac{loss(FP)}{loss(FP) + loss(FN)}$$