Statistical Machine Learning

- Recent advances in statistics have been devoted to developing more powerful automated techniques for predictive modeling — both regression and classification.
- These methods fall under the umbrella of statistical machine learning, and are distinguished from classical statistical methods in that they are data-driven and do not seek to impose linear or other overall structure on the data.
- MACHINE LEARNING VERSUS STATISTICS:
  - In the context of predictive modeling, what is the difference between machine learning and statistics?
  - There is not a bright line dividing the two disciplines.
  - Machine learning tends to be more focused on developing efficient algorithms that scale to large data in order to optimize the predictive model.
  - Statistics generally pays more attention to the probabilistic theory and underlying structure of the model.

*This presentation is based on the book ‘Practical Statistics for Data Scientist’ by Peter Bruce & Andrew Bruce

K-Nearest Neighbors

- The idea behind K-Nearest Neighbors (KNN) is very simple.1 For each record to be classified or predicted:
  - Find K records that have similar features (i.e., similar predictor values).
  - For classification: Find out what the majority class is among those similar records, and assign that class to the new record.
  - For prediction (also called KNN regression): Find the average among those similar records, and predict that average for the new record.
- KNN is one of the simpler prediction/classification techniques: there is no model to be fit (as in regression).
- This doesn’t mean that using KNN is an automatic procedure. The prediction results depend on how the features are scaled, how similarity is measured, and how big K is set.
- Also, all predictors must be in numeric form.

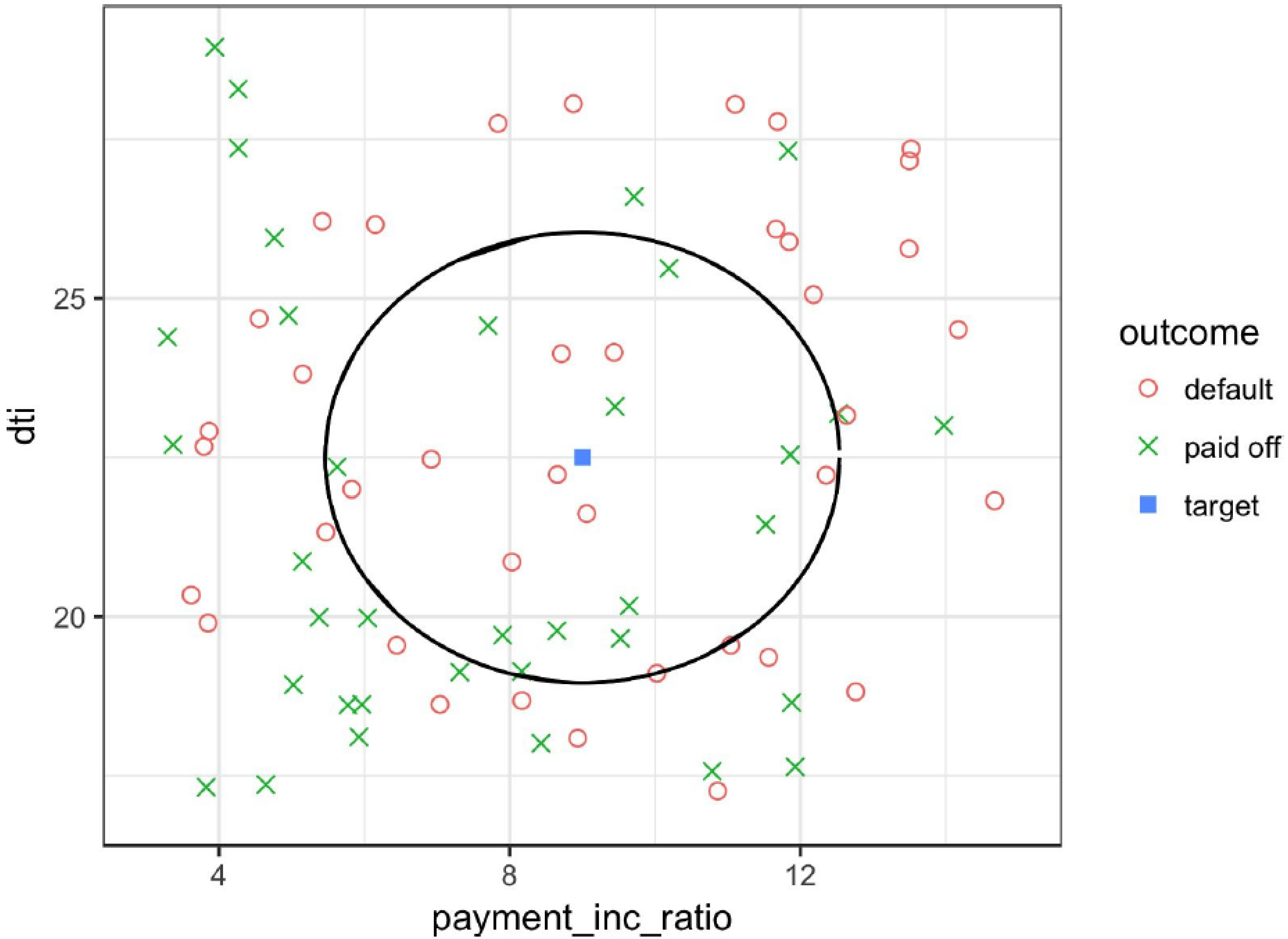


Figure 6-2. KNN prediction of loan default using two variables: debt-to-income ratio and loan payment-to-income ratio

Distance Metrics

- Euclidean distance $\sqrt{(x_1 - u_1)^2 + (x_2 - u_2)^2 + \cdots + (x_p - u_p)^2}$
- Manhattan distance $| x_1 - u_1 | + | x_2 - u_2 | + \cdots + | x_p - u_p |$

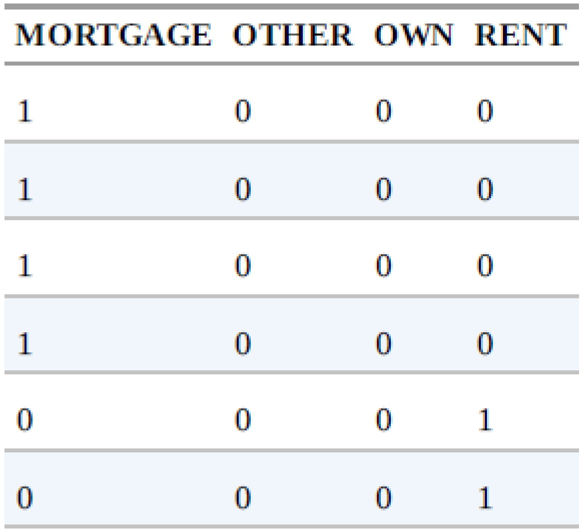
- In measuring distance between two vectors, variables (features) that are measured with comparatively large scale will dominate the measure.
- For example, for the loan data, the distance would be almost solely a function of the income and loan amount variables, which are measured in tens or hundreds of thousands. Ratio variables would count for practically nothing in comparison.
- We address this problem by standardizing the data

One Hot Encoder

- The loan data includes several factor (string) variables.
- Most statistical and machine learning models require this type of variable to be converted to a series of binary dummy variables conveying the same information.
- The phrase one hot encoding comes from digital circuit terminology, where it describes circuit settings in which only one bit is allowed to be positive (hot).

- In linear and logistic regression, one hot encoding causes problems with multicollinearity;
- In such cases, one dummy is omitted (its value can be inferred from the other values).
- This is not an issue with KNN and other methods.

Table 6-2. Representing home ownership factor data as a numeric dummy variable



MORTGAGE	OTHER	OWN	RENT
1	0	0	0
1	0	0	0
1	0	0	0
1	0	0	0
0	0	0	1
0	0	0	1

Standardization (Normalization, Z-Scores)

- In measurement, we are often not so much interested in “how much” but “how different from the average.”
- Standardization, also called normalization, puts all variables on similar scales by subtracting the mean and dividing by the standard deviation.
- In this way, we ensure that a variable does not overly influence a model simply due to the scale of its original measurement.

$$z = \frac{x - \bar{x}}{s}$$

- These are commonly referred to as z-scores. Measurements are then stated in terms of “standard deviations away from the mean.”
- In this way, a variable’s impact on a model is not affected by the scale of its original measurement.
- For KNN and a few other procedures (e.g., principal components analysis and clustering), it is essential to consider standardizing the data prior to applying the procedure.

Choosing K

- The choice of K is very important to the performance of KNN.
- Generally speaking, if K is too low, we may be over-fitting: including the noise in the data. Higher values of K provide smoothing that reduces the risk of over-fitting in the training data.
- On the other hand, if K is too high, we may over-smooth the data and miss out on KNN's ability to capture the local structure in the data, one of its main advantages.
- The K that best balances between over-fitting and over-smoothing is typically determined by accuracy metrics and, in particular, accuracy with holdout or validation data.
- There is no general rule about the best K - it depends greatly on the nature of the data.
- For highly structured data with little noise, smaller values of K work best.

BIAS-VARIANCE TRADEOFF

(as a reminder)

- The tension between over-smoothing and over-fitting is an instance of the bias-variance tradeoff, an ubiquitous problem in statistical model fitting.
- Variance refers to the modeling error that occurs because of the choice of training data; that is, if you were to choose a different set of training data, the resulting model would be different.
- Bias refers to the modeling error that occurs because you have not properly identified the underlying real-world scenario; this error would not disappear if you simply added more training data.
- When a flexible model is over-fit, the variance increases. You can reduce this by using a simpler model, but the bias may increase due to the loss of flexibility in modeling the real underlying situation.
- A general approach to handling this tradeoff is through cross-validation.

KNN as a Feature Engine

- KNN gained its popularity due to its simplicity and intuitive nature. In terms of performance, KNN by itself is usually not competitive with more sophisticated classification techniques.
- In practical model fitting, however, KNN can be used to add “local knowledge” in a staged process with other classification techniques.
  - KNN is run on the data, and for each record, a classification (or quasi-probability of a class) is derived.
  - That result is added as a new feature to the record, and another classification method is then run on the data. The original predictor variables are thus used twice.

- At first you might wonder whether this process, since it uses some predictors twice, causes a problem with multi-collinearity (see “Multicollinearity”).
- This is not an issue, since the information being incorporated into the second-stage model is highly local, derived only from a few nearby records, and is therefore additional information, and not redundant.

Tree Models

- Tree models, also called Classification and Regression Trees (CART),2 decision trees, or just trees, are an effective and popular classification (and regression) method.
- form the basis for the most widely used and powerful predictive modeling tools in data science for both regression and classification.
- A tree model is a set of “if-then-else” rules that are easy to understand and to implement.
- In contrast to regression and logistic regression, trees have the ability to discover hidden patterns corresponding to complex interactions in the data.
- However, unlike KNN or naive Bayes, simple tree models can be expressed in terms of predictor relationships that are easily interpretable.
- The resulting tree is shown in next slide. These classification rules are determined by traversing through a hierarchical tree, starting at the root until a leaf is reached.

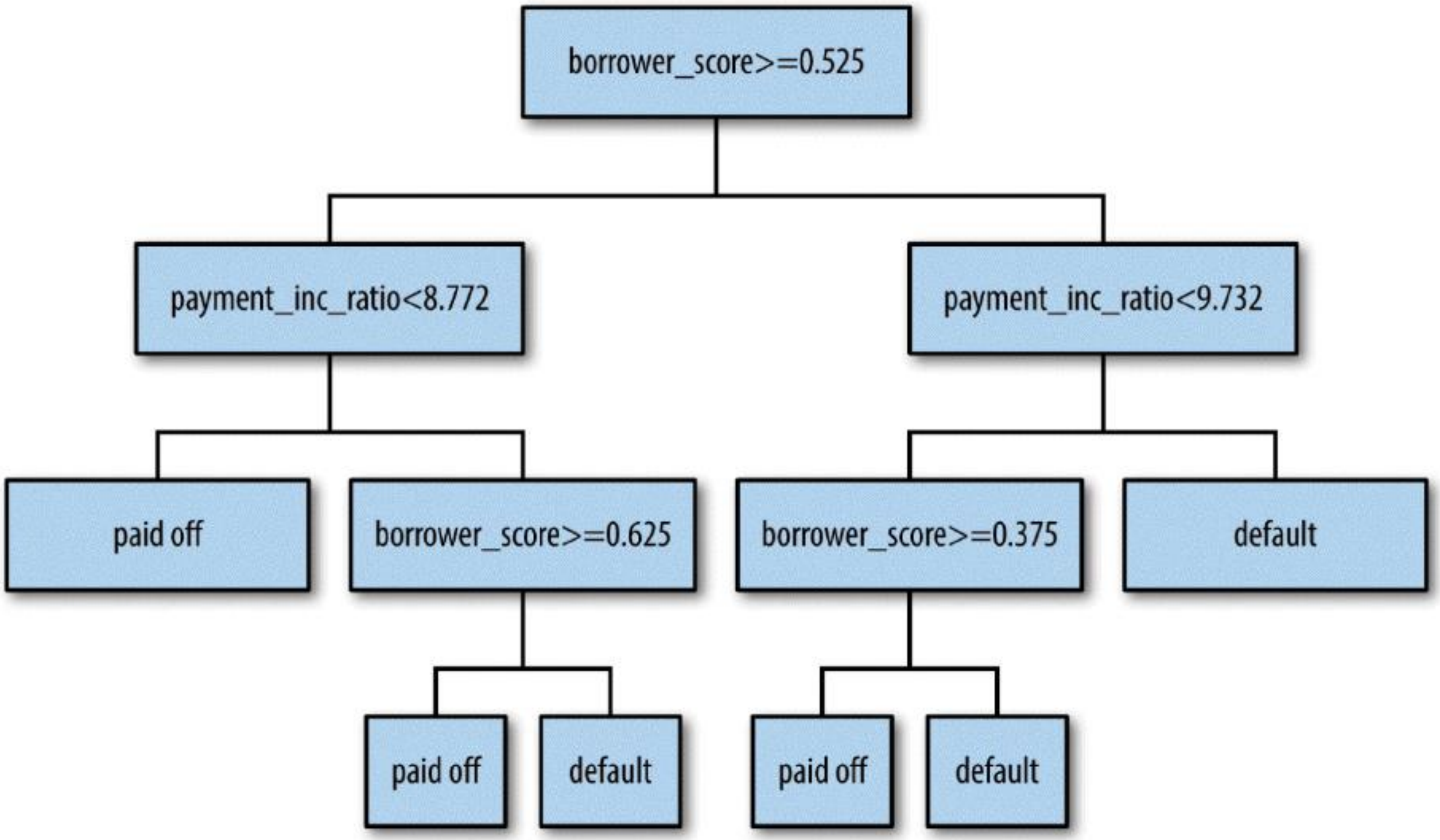


Figure 6-3. The rules for a simple tree model fit to the loan data

The Recursive Partitioning Algorithm

- The algorithm to construct a decision tree, called recursive partitioning , is straightforward and intuitive.
- The data is repeatedly partitioned using predictor values that do the best job of separating the data into relatively homogeneous partitions.

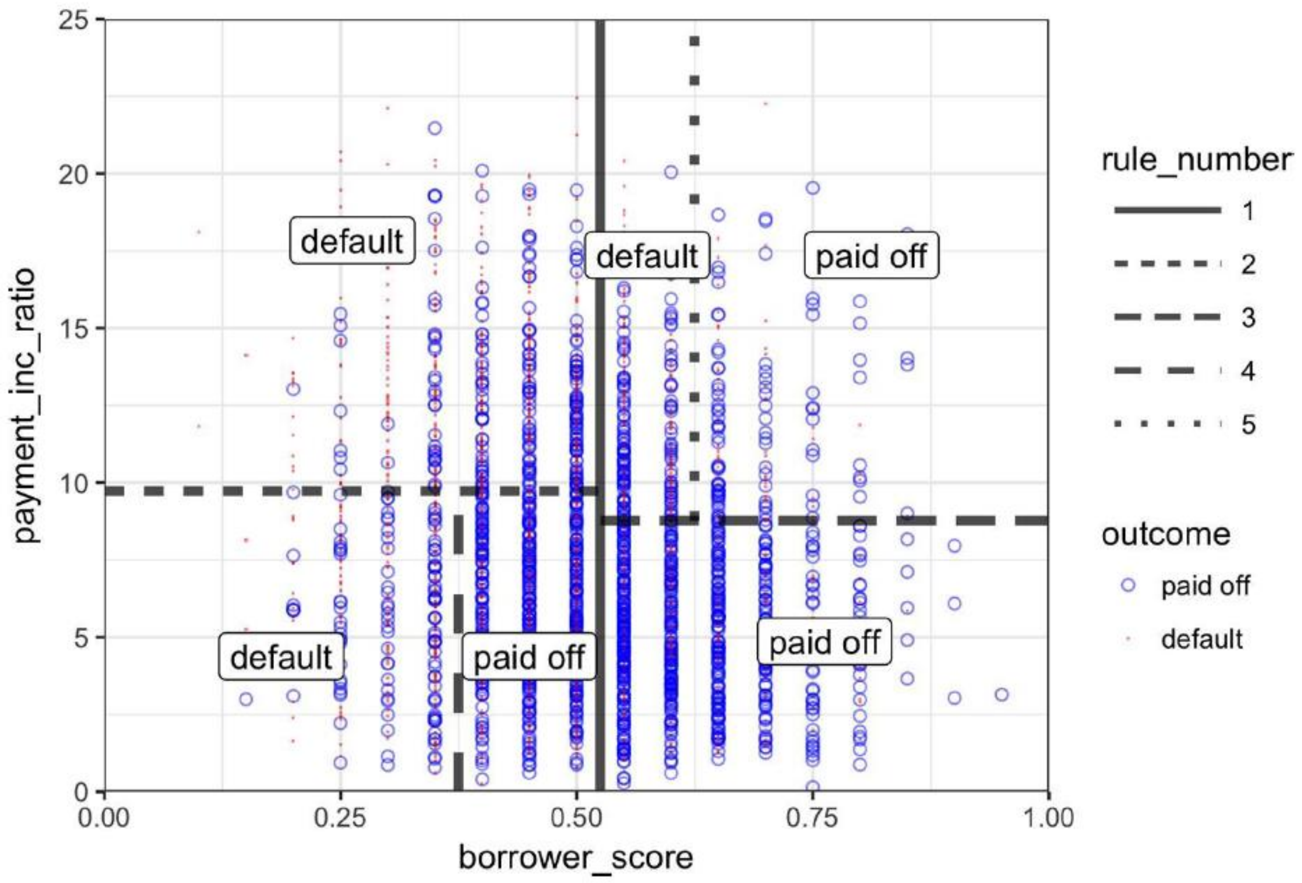


Figure 6-4. The rules for a simple tree model fit to the loan data

Measuring Homogeneity or Impurity

- You can see from the preceding algorithm that we need a way to measure homogeneity, also called class purity, within a partition. Or, equivalently, we need to measure the impurity of a partition.
- It turns out that accuracy is not a good measure for impurity. Instead, two common measures for impurity are the Gini impurity and entropy or information.
- The Gini impurity for a set of records A is:

$$I(A) = p(1-p)$$

- The entropy measure is given by:

$$I(A) = -p\log_2(p) - (1-p)\log_2(1-p)$$

Stopping the Tree from Growing

- As the tree grows bigger, the splitting rules become more detailed, and the tree gradually shifts from identifying "big" rules that identify real and reliable relationships in the data to "tiny" rules that reflect only noise.
- A fully grown tree results in completely pure leaves and, hence, 100% accuracy in classifying the data that it is trained on.
- This accuracy is, of course, illusory — we have over-fit the data, fitting the noise in the training data, not the signal that we want to identify in new data.
- We need some way to determine when to stop growing a tree at a stage that will generalize to new data.
- There are two common ways to stop splitting:
  - Avoid splitting a partition if a resulting sub-partition is too small, or if a terminal leaf is too small. In rpart, these constraints are controlled separately by the parameters **minsplit** and **minbucket**, respectively, with defaults of 20 and 7.
  - Don't split a partition if the new partition does not "significantly" reduce the impurity. In rpart, this is controlled by the complexity parameter **cp**, which is a measure of how complex a tree is — the more complex, the greater the value of cp. In practice, **cp** is used to limit tree growth by attaching a penalty to additional complexity (splits) in a tree.
- If cp is too small, then the tree will overfit the data, fitting noise and not signal. On the other hand, if cp is too large, then the tree will be too small and have little predictive power.

How Trees Are Used

- One of the big obstacles faced by predictive modelers in organizations is the perceived “black box” nature of the methods they use, which gives rise to opposition from other elements of the organization. In this regard, the tree model has two appealing aspects.
  - Tree models provide a visual tool for exploring the data, to gain an idea of what variables are important and how they relate to one another. Trees can capture nonlinear relationships among predictor variables.
  - Tree models provide a set of rules that can be effectively communicated to nonspecialists, either for implementation or to “sell” a data mining project.

- When it comes to prediction, however, harnessing the results from multiple trees is typically more powerful than just using a single tree.
- In particular, the random forest and boosted tree algorithms almost always provide superior predictive accuracy and performance but the aforementioned advantages of a single tree are lost.

Bagging and the Random Forest

- In 1907, the statistician Sir Francis Galton was visiting a county fair in England, at which a contest was being held to guess the dressed weight of an ox that was on exhibit. There were 800 guesses, and, while the individual guesses varied widely, both the mean and the median came out within 1% of the ox's true weight. James Suroweicki has explored this phenomenon in his book **The Wisdom of Crowds.**
- This principle applies to predictive models, as well: averaging (or taking majority votes) of multiple models — an ensemble of models — turns out to be more accurate than just selecting one model.
- The simple version of ensembles is as follows:
  - Develop a predictive model and record the predictions for a given data set.
  - Repeat for multiple models, on the same data.
  - For each record to be predicted, take an average (or a weighted average, or a majority vote) of the predictions.

- Ensemble methods have been applied most systematically and effectively to decision trees. Ensemble tree models are so powerful that they provide a way to build good predictive models with relatively little effort.
- Going beyond the simple ensemble algorithm, there are two main variants of ensemble models: bagging and boosting. In the case of ensemble tree models, these are referred to as random forest models and boosted tree models.

Bagging

- Bagging stands for “bootstrap aggregating”
- Bagging is like the basic algorithm for ensembles, except that, instead of fitting the various models to the same data, each new model is fit to a bootstrap resample.
  - Initialize M, the number of models to be fit, and n, the number of records to choose (n < N). Set the counter m
  - Take a bootstrap resample (i.e., with replacement) of n records from the training data to form a subsample and the bag
  - Train a model using subsample and the bag to create a set of decision rules .
  - Increment the model counter m+1. If m <= M, go to step 1.

Random Forest

- The random forest is based on applying bagging to decision trees with one important extension: in addition to sampling the records, the algorithm also samples the variables.
- With random forests, at each stage of the algorithm, the choice of variable is limited to a random subset of variables.
- Compared to the basic tree algorithm (see “The Recursive Partitioning Algorithm”), the random forest algorithm adds two more steps: the bagging discussed earlier and the bootstrap sampling of variables at each split.
- How many variables to sample at each step? A rule of thumb is to choose $\sqrt{P}$ where P is the number of predictor variables.
- The out-of-bag (OOB) estimate of error is the error rate for the trained models, applied to the data left out of the training set for that tree. Using the output from the model, the OOB error can be plotted versus the number of trees in the random forest:

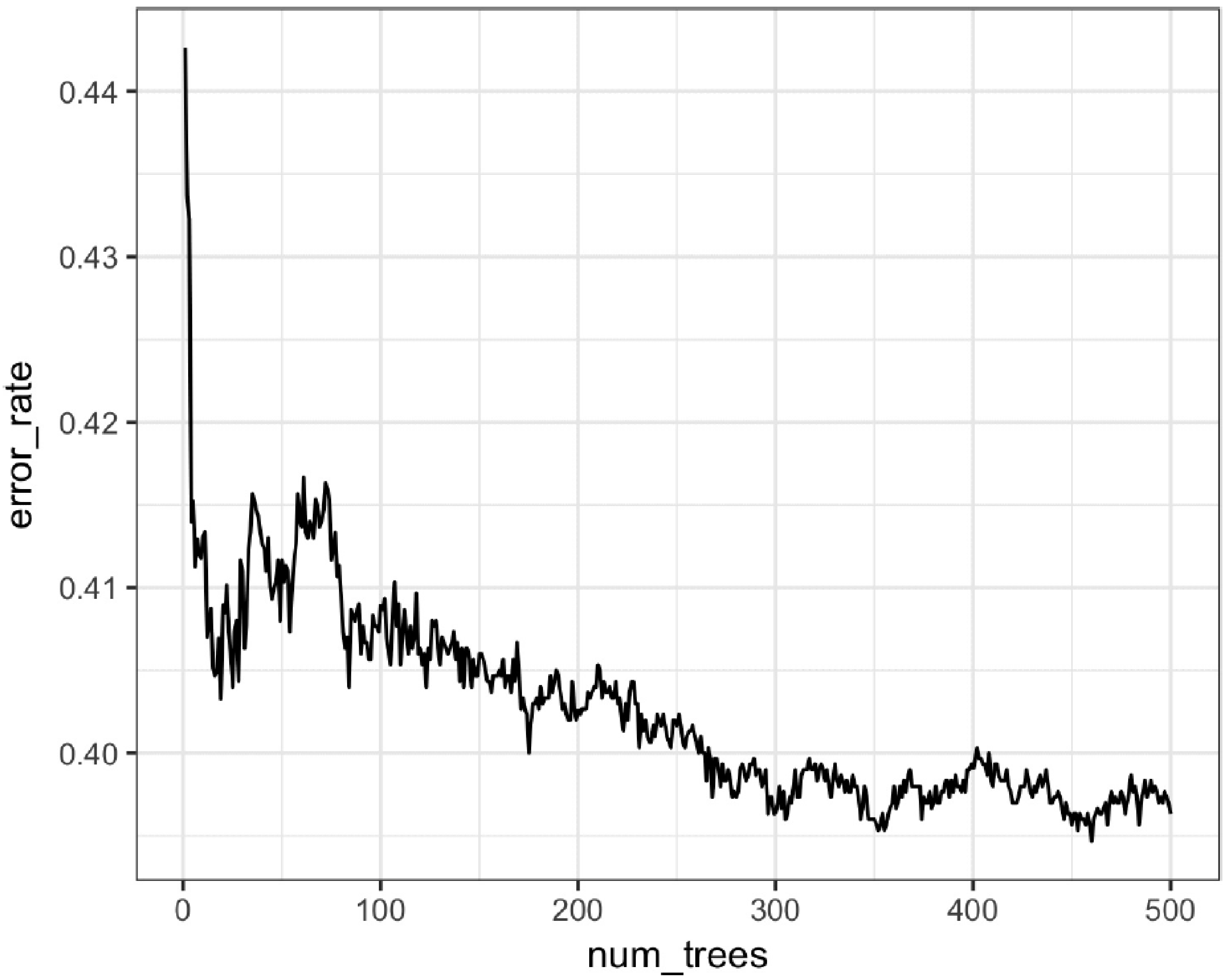


Figure 6-6. The improvement in accuracy of the random forest with the addition of more trees

- The random forest method is a “black box” method.
- It produces more accurate predictions than a simple tree, but the simple tree’s intuitive decision rules are lost.
- The predictions are also somewhat noisy: note that some borrowers with a very high score, indicating high creditworthiness, still end up with a prediction of default.
- This is a result of some unusual records in the data and demonstrates the danger of over-fitting by the random forest

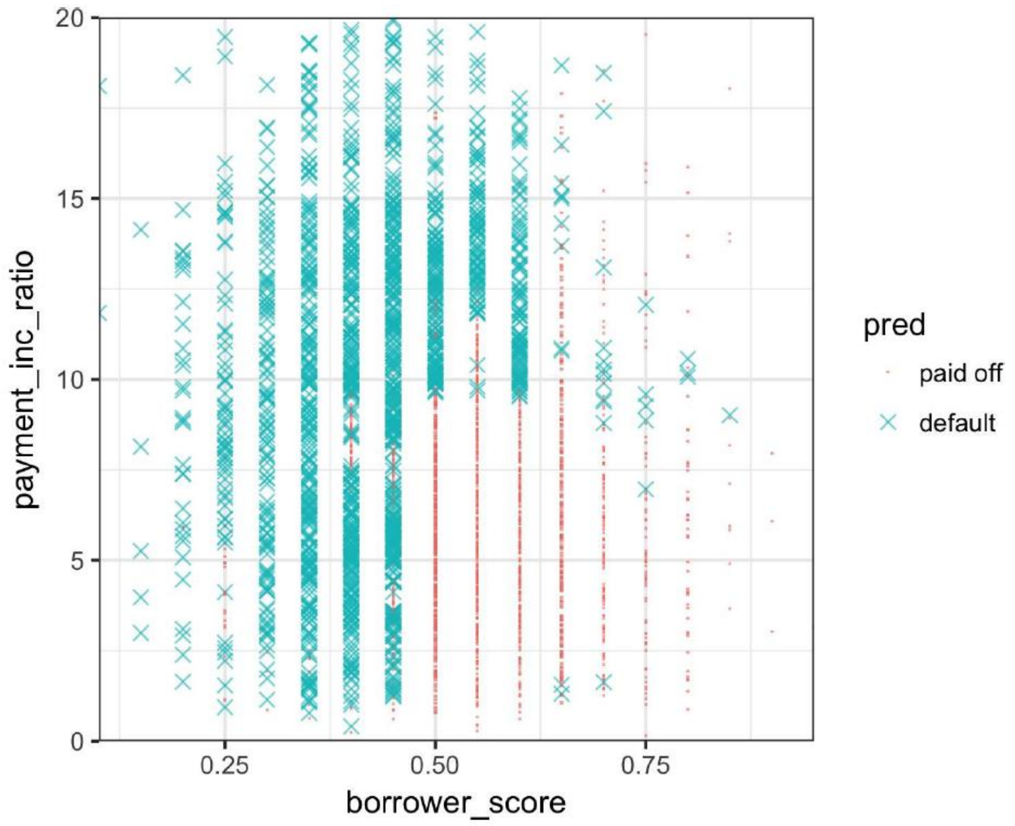
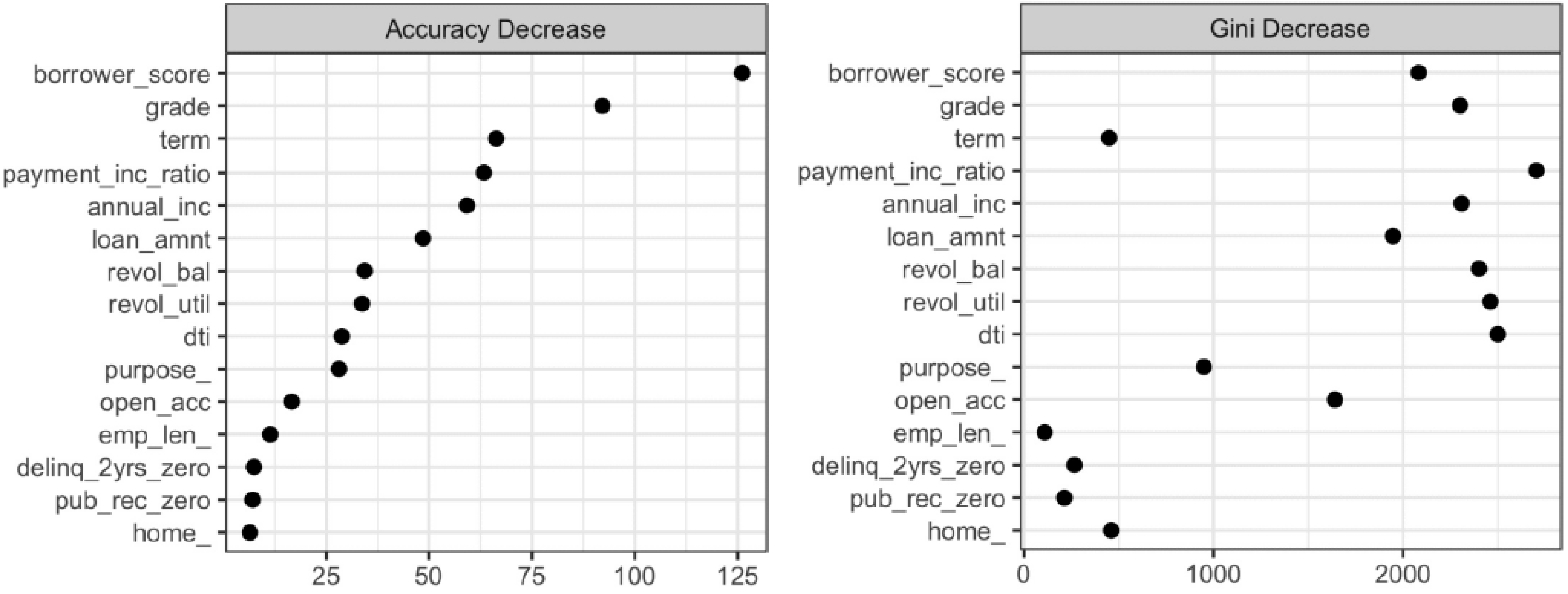


Figure 6-7. The predicted outcomes from the random forest applied to the loan default data

Variable Importance

- The power of the random forest algorithm shows itself when you build predictive models for data with many features and records.
- It has the ability to automatically determine which predictors are important and discover complex relationships between predictors corresponding to interaction terms.
- There are two ways to measure variable importance:
  - By the decrease in accuracy of the model if the values of a variable are randomly permuted (type=1). Randomly permuting the values has the effect of removing all predictive power for that variable. The accuracy is computed from the out-of-bag data (so this measure is effectively a crossvalidated estimate).
  - By the mean decrease in the Gini impurity score (see “Measuring Homogeneity or Impurity”) for all of the nodes that were split on a variable (type=2). This measures how much improvement to the purity of the nodes that variable contributes. This measure is based on the training set, and therefore less reliable than a measure calculated on out-of-bag data.



Accuracy Decrease
Gini Decrease
borrower_score
grade
term
payment_inc_ratio
annual_inc
loan_amnt
revol_bal
revol_util
dti
purpose_
open_acc
emp_len_
delinq_2yrs_zero
pub_rec_zero
home_
25
50
75
100
125
0
1000
2000

Hyperparameters

- The random forest, as with many statistical machine learning algorithms, can be considered a black-box algorithm with knobs to adjust how the box works.
- These knobs are called hyper-parameters, which are parameters that you need to set before fitting a model; they are not optimized as part of the training process.
- While traditional statistical models require choices (e.g., the choice of predictors to use in a regression model), the hyper-parameters for random forest are more critical, especially to avoid over-fitting.
- In particular, the two most important hyper-paremeters for the random forest are:
  - ***nodesize***
  - The minimum size for terminal nodes (leaves in the tree). The default is 1 for classification and 5 for regression.
  - ***maxnodes***
  - The maximum number of nodes in each decision tree. By default, there is no limit and the largest tree will be fit subject to the constraints of nodesize.

Boosting

- Ensemble models have become a standard tool for predictive modeling. Boosting is a general technique to create an ensemble of models. It was developed around the same time as bagging.
- Like bagging, boosting is most commonly used with decision trees. Despite their similarities, boosting takes a very different approach — one that comes with many more bells and whistles.

- As a result, while bagging can be done with relatively little tuning, boosting requires much greater care in its application.
- If these two methods were cars, bagging could be considered a Honda Accord (reliable and steady), whereas boosting could be considered a Porsche (powerful but requires more care).

- In linear regression models, the residuals are often examined to see if the fit can be improved. Boosting takes this concept much further and fits a series of models with each successive model fit to minimize the error of the previous models.
- Several variants of the algorithm are commonly used: Adaboost, gradient boosting, and stochastic gradient boosting.

KEY TERMS FOR BOOSTING

Ensemble

Forming a prediction by using a collection of models.

Synonym

Model averaging

Boosting

A general technique to fit a sequence of models by giving more weight to the records with large residuals for each successive round.

Adaboost

An early version of boosting based on reweighting the data based on the residuals.

Gradient boosting

A more general form of boosting that is cast in terms of minimizing a cost function.

Stochastic gradient boosting

The most general algorithm for boosting that incorporates resampling of records and columns in each round.

Regularization

A technique to avoid overfitting by adding a penalty term to the cost function on the number of parameters in the model.

Hyperparameters

Parameters that need to be set before fitting the algorithm.

The Boosting Algorithm

- The basic idea behind the various boosting algorithms is essentially the same. The easiest to understand is Adaboost, which proceeds as follows:

1. Initialize M, the maximum number of models to be fit, and set the iteration counter $m = 1$. Initialize the observation weights $w_i = 1 / N$ for $i = 1, 2, \ldots, N$. Initialize the ensemble model $\hat{F}_0 = 0$.
2. Train a model using $\hat{f}_m$ using the observation weights $w_1, w_2, \ldots, w_N$ that minimizes the weighted error e_m defined by summing the weights for the misclassified observations.
3. Add the model to the ensemble: $\hat{F}_m = \hat{F}_{m-1} + \alpha_m \hat{f}_m$ where $\alpha_m = \frac{\log 1 - e_m}{e_m}$.
4. Update the weights $w_1, w_2, \ldots, w_N$ so that the weights are increased for the observations that were misclassfied. The size of the increase depends on α_m with larger values of α_m leading to bigger weights.
5. Increment the model counter $m = m + 1$. If $m \leq M$, go to step 1.

- By increasing the weights for the observations that were misclassified, the algorithm forces the models to train more heavily on the data for which it performed poorly. The factor α ensures that models with lower error have a bigger weight.

XGBoost

- The most widely used public domain software for boosting is XGBoost, an implementation of stochastic gradient boosting
- The function xgboost has many parameters that can, and should, be adjusted (see “Hyper-parameters and Cross-Validation”).
- Two very important parameters are **subsample**, which controls the fraction of observations that should be sampled at each iteration, and **eta**, a shrinkage factor applied to α in the boosting algorithm
- Using subsample makes boosting act like the random forest except that the sampling is done without replacement. The shrinkage parameter eta is helpful to prevent over-fitting by reducing the change in the weights (a smaller change in the weights means the algorithm is less likely to over-fit to the training set).

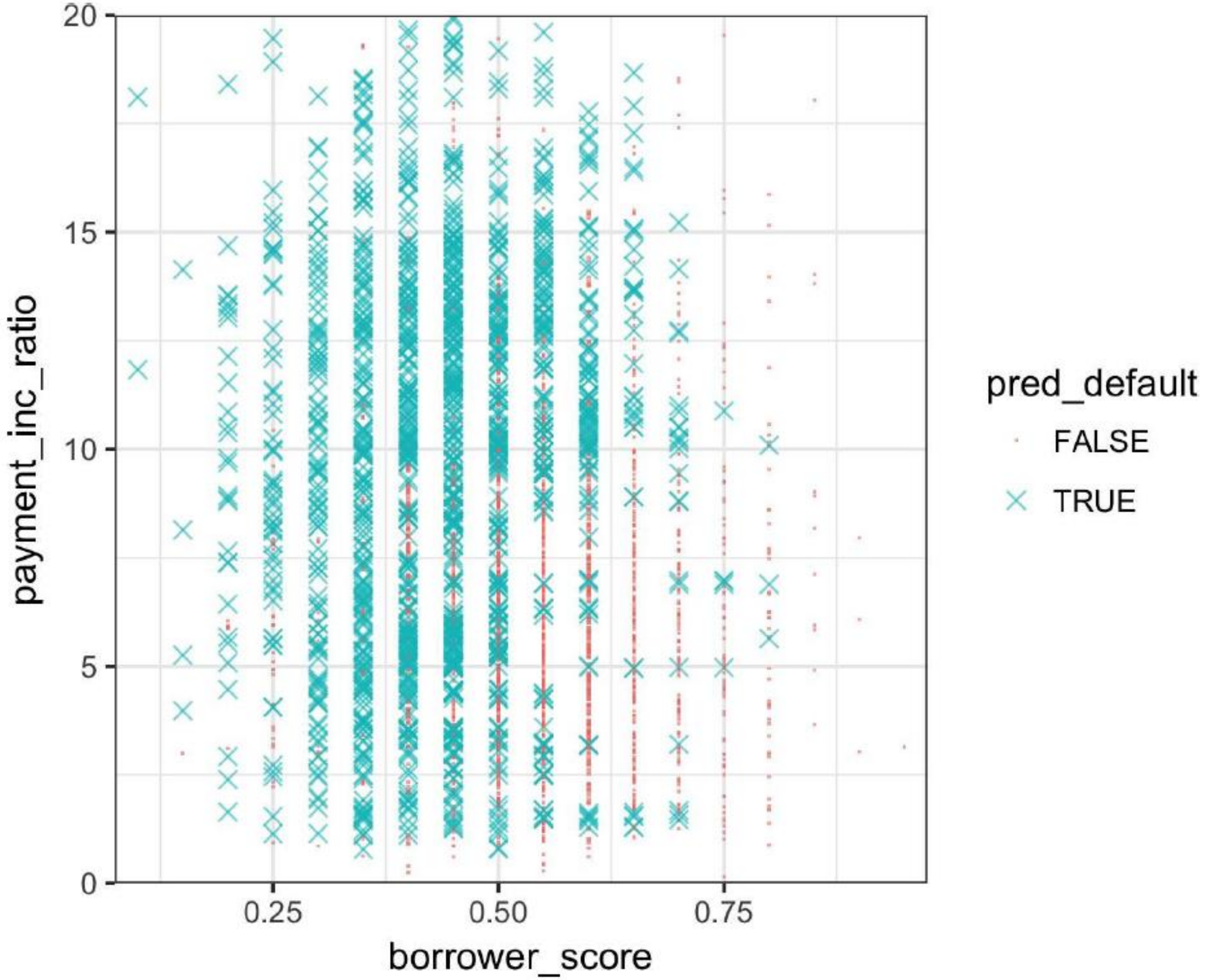
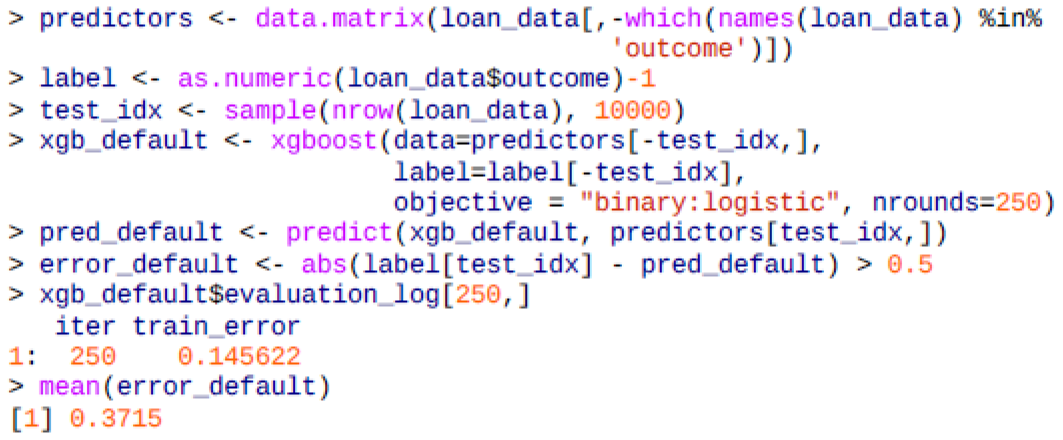


Figure 6-9. The predicted outcomes from XGBoost applied to the loan default data

Regularization: Avoiding Overfitting

- Blind application of xgboost can lead to unstable models as a result of overfitting to the training data. The problem with over-fitting is twofold:
  - The accuracy of the model on new data not in the training set will be degraded.
  - The predictions from the model are highly variable, leading to unstable results.
- Any modeling technique is potentially prone to over-fitting. For example, if too many variables are included in a regression equation, the model may end up with spurious predictions.
- However, for most statistical techniques, over-fitting can be avoided by a judicious selection of predictor variables. Even the random forest generally produces a reasonable model without tuning the parameters. This, however, is not the case for xgboost.



```
> predictors <- data.matrix(loan_data[,-which(names(loan_data) %in%
                                        'outcome')])
> label <- as.numeric(loan_data$outcome)-1
> test_idx <- sample(nrow(loan_data), 10000)
> xgb_default <- xgboost(data=predictors[-test_idx,],
                         label=label[-test_idx],
                         objective = "binary:logistic", nrounds=250)
> pred_default <- predict(xgb_default, predictors[test_idx,])
> error_default <- abs(label[test_idx] - pred_default) > 0.5
> xgb_default$evaluation_log[250,]
   iter train_error
1:  250    0.145622
> mean(error_default)
[1] 0.3715
```

- The test set consists of 10,000 randomly sampled records from the full data, and the training set consists of the remaining records. Boosting leads to an error rate of only 14.6% for the training set. The test set, however, has a much higher error rate of 36.2%.
- This is a result of overfitting: while boosting can explain the variability in the training set very well, the prediction rules do not apply to new data.
- Boosting provides several parameters to avoid overfitting, including the parameters eta and subsample.
- Another approach is regularization, a technique that modifies the cost function in order to penalize the complexity of the model.
- In xgboost, it is possible to modify the cost function by adding a term that measures the complexity of the model.
- There are two parameters in xgboost to regularize the model: alpha and lambda, which correspond to Manhattan distance and squared Euclidean distance, respectively

RIDGE REGRESSION AND THE LASSO

Adding a penalty on the complexity of a model to help avoid overfitting dates back to the 1970s. Least squares regression minimizes the residual sum of squares (RSS); see "Least Squares". *Ridge regression* minimizes the sum of squared residuals plus a penalty on the number and size of the coefficients:

$$\sum_{i=1}^{n}\left(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots \hat{b} X_p\right)^2 + \lambda\left(\hat{b}_1^2 + \cdots + \hat{b}_p^2\right)$$

The value of λ determines how much the coefficients are penalized; larger values produce models that are less likely to overfit the data. The *Lasso* is similar, except that it uses Manhattan distance instead of Euclidean distance as a penalty term:

$$\sum_{i=1}^{n}\left(Y_i - \hat{b}_0 - \hat{b}_1 X_i - \cdots \hat{b} X_p\right)^2 + \alpha\left(|\hat{b}_1| + \cdots + |\hat{b}_p|\right)$$

The `xgboost` parameters `lambda` and `alpha` are acting in a similar mannger.