validation, and test sets is done either randomly according to predetermined proportions or by specifying which records go into which partition according to some relevant variable (e.g., in time-series forecasting, the data are partitioned according to their chronological order). In most cases, the partitioning should be done randomly to minimize the chance of getting a biased partition. Note the varying nomenclature—the training partition is nearly always called "training" but the names for the other partitions can vary and overlap.

Training Partition

The training partition, typically the largest partition, contains the data used to build the various models we are examining. The same training partition is generally used to develop multiple models.

Validation Partition

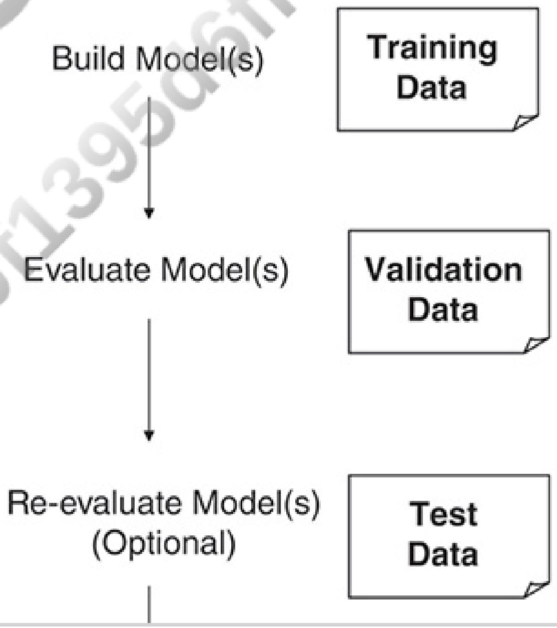
The validation partition (sometimes called the *test partition*) is used to assess the predictive performance of each model so that you can compare models and choose the best one. In some algorithms (e.g., classification and regression trees, *k*-nearest neighbors), the validation partition may be used in an automated fashion to tune and improve the model.

Test Partition

The test partition (sometimes called the *holdout* or *evaluation partition*) is used to assess the performance of the chosen model with new data.

Why have both a validation and a test partition? When we use the validation data to assess multiple models and then choose the model that performs best with the validation data, we again encounter another (lesser) facet of the overfitting problem—chance aspects of the validation data that happen to match the chosen model better than they match other models. In other words, by using the validation data to choose one of several models, the performance of the chosen model on the validation data will be overly optimistic.

The random features of the validation data that enhance the apparent performance of the chosen model will probably not be present in new data to which the model is applied. Therefore, we may have overestimated the accuracy of our model. The more models we test, the more likely it is that one of them will be particularly effective in modeling the noise in the validation data. Applying the model to the test data, which it has not seen before, will provide an unbiased estimate of how well the model will perform with new data. Figure 2.4 shows the three data partitions and their use in the data mining process. When we are concerned mainly with finding the best model and less with exactly how well it will do, we might use only training and validation partitions. Table 2.9 shows R code to partition the West Roxbury data into two sets (training and validation) or into three sets (training, validation, and test). This is done by first drawing a random sample of records into the training set, then assigning the remaining records as validation. In the case of three partitions, the validation records are chosen randomly from the data after excluding the records already sampled into the training set.



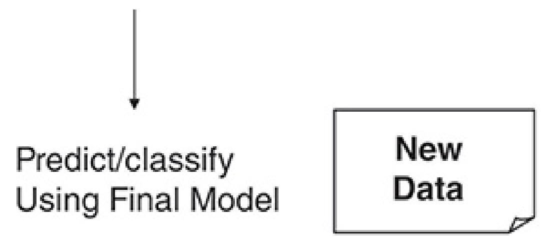


Figure 2.4 Three data partitions and their role in the data mining process

Table 2.9 Data Partitioning in R

code for partitioning the West Roxbury data into training, validation (and test) sets

```
# use set.seed() to get the same partitions when re-running the R code.
set.seed(1)

## partitioning into training (60%) and validation (40%)
# randomly sample 60% of the row IDs for training; the remaining 40% serve as
# validation
train.rows <- sample(rownames(housing.df), dim(housing.df)[1]*0.6)
# collect all the columns with training row ID into training set:
train.data <- housing.df[train.rows, ]
# assign row IDs that are not already in the training set, into validation
valid.rows <- setdiff(rownames(housing.df), train.rows)
valid.data <- housing.df[valid.rows, ]

# alternative code for validation (works only when row names are numeric):
# collect all the columns without training row ID into validation set
valid.data <- housing.df[-train.rows, ] # does not work in this case

## partitioning into training (50%), validation (30%), test (20%)
# randomly sample 50% of the row IDs for training
train.rows <- sample(rownames(housing.df), dim(housing.df)[1]*0.5)

# sample 30% of the row IDs into the validation set, drawing only from records
# not already in the training set
# use setdiff() to find records not already in the training set
valid.rows <- sample(setdiff(rownames(housing.df), train.rows),
              dim(housing.df)[1]*0.3)

# assign the remaining 20% row IDs serve as test
test.rows <- setdiff(rownames(housing.df), union(train.rows, valid.rows))

# create the 3 data frames by collecting all columns from the appropriate rows
train.data <- housing.df[train.rows, ]
valid.data <- housing.df[valid.rows, ]
test.data <- housing.df[test.rows, ]
```

Note that with some algorithms, such as nearest-neighbor algorithms, records in the validation and test partitions, and in new data, are compared to records in the training data to find the nearest neighbor(s). As *k*-nearest neighbors is discussed in this book, the use of two partitions is an essential part of the classification or prediction process, not merely a way to improve or assess it. Nonetheless, we can still interpret the error in the validation data in the same way that we would interpret error from any other model.

Cross-Validation

When the number of records in our sample is small, data partitioning might not be advisable as each partition will contain too few records for model building and performance evaluation. An alternative to data partitioning is cross-validation, which is especially useful with small samples. Cross-validation is a procedure that starts with partitioning the data into "folds," or non-overlapping subsamples. Often we choose *k* = 5 folds, meaning that the data are randomly partitioned into 5 equal parts, where each fold has 20% of the observations. A model is then fit *k* times.

Each time, one of the folds is used as the validation set and the remaining $k - 1$ folds serve as the training set. The result is that each fold is used once as the validation set, thereby producing predictions for every observation in the dataset. We can then combine the model's predictions on each of the k validation sets in order to evaluate the overall performance of the model. Sometimes cross-validation is built into a data mining algorithm, with the results of the cross-validation used for choosing the algorithm's parameters (see, e.g., Chapter 9).

2.6 Building a Predictive Model

Let us go through the steps typical to many data mining tasks using a familiar procedure: multiple linear regression. This will help you understand the overall process before we begin tackling new algorithms.

Modeling Process

We now describe in detail the various model stages using the West Roxbury home values example.

1. *Determine the purpose*. Let's assume that the purpose of our data mining project is to predict the value of homes in West Roxbury for new records.
2. *Obtain the data*. We will use the 2014 West Roxbury housing data. The dataset in question is small enough that we do not need to sample from it—we can use it in its entirety.
3. *Explore, clean, and preprocess the data*. Let's look first at the description of the variables, also known as the "data dictionary," to be sure that we understand them all. These descriptions are available on the "description" worksheet in the Excel file and in Table 2.2. The variable names and descriptions in this dataset all seem fairly straightforward, but this is not always the case. Often, variable names are cryptic and their descriptions may be unclear or missing.

   It is useful to pause and think about what the variables mean and whether they should be included in the model. Consider the variable TAX. At first glance, we consider that the tax on a home is usually a function of its assessed value, so there is some circularity in the model—we want to predict a home's value using TAX as a predictor, yet TAX itself is determined by a home's value. TAX might be a very good predictor of home value in a numerical sense, but would it be useful if we wanted to apply our model to homes whose assessed value might not be known? For this reason, we will exclude TAX from the analysis.

   It is also useful to check for outliers that might be errors. For example, suppose that the column FLOORS (number of floors) looked like the one in Table 2.10, after sorting the data in descending order based on floors.

   We can tell right away that the 15 is in error—it is unlikely that a home has 15 floors. Since all other values are between 1 and 2 the decimal was probably misplaced and the value should be 1.5.

   Lastly, we create dummy variables for categorical variables. Here we have one categorical variable: REMODEL, which has three categories.
4. *Reduce the data dimension*. The West Roxbury dataset has been prepared for presentation with fairly low dimension—it has only 13 variables, and the single categorical variable considered has only three categories (and hence adds two dummy variables when used in a linear regression model). If we had many more variables, at this stage we might want to apply a variable reduction technique, such as condensing multiple categories into a smaller number, or applying principal components analysis to consolidate multiple similar numerical variables (e.g., LIVING AREA, ROOMS, BEDROOMS, BATH, HALF BATH) into a smaller number of variables.
5. *Determine the data mining task*. The specific task is to predict the value of TOTAL VALUE using the predictor variables. This is a supervised prediction task. For simplicity, we excluded several additional variables present in the original dataset, which have many categories (BLDG TYPE, ROOF TYPE, and EXT FIN). We therefore use all the numerical variables (except TAX) and the dummies created for the remaining categorical variables.
6. *Partition the data (for supervised tasks)*. In this case we divide the data into two partitions: training and validation (see Table 2.9). The training partition is used to build the model, and the validation partition is used to see how well the model does when applied to new data. We need to specify the percent of the data used in each partition.

   Note: Although not used in our example, a test partition might also be used.
7. *Choose the technique*. In this case, it is multiple linear regression. Having divided the data into training and validation partitions, we can build a multiple linear regression model with the training data. We want to predict the value of a house in West Roxbury on the basis of all the other predictors (except TAX).
8. *Use the algorithm to perform the task*. In R, we use the *lm()* function to predict house value with the training data,

8. Use the algorithm to perform the task. In it, we use the lm() function to predict house value with the training data, then use the same model to predict values for the validation data. Chapter 6 on linear regression goes into more detail. Table 2.11 shows the predicted values for the first few records in the training data along with the actual values and the residuals (prediction errors). Note that the predicted values are often called the *fitted values*, since they are for the records to which the model was fit. The results for the validation data are shown in Table 2.12. The prediction errors for the training and validation data are compared in Table 2.13.

   Prediction error can be aggregated in several ways. Five common measures are shown in Table 2.13. The first is *mean error (ME)*, simply the average of the residuals (errors). In both cases, it is quite small relative to the units of TOTAL VALUE, indicating that, on balance, predictions average about right—our predictions are "unbiased." Of course, this simply means that the positive and negative errors balance out. It tells us nothing about how large these errors are.

   The *RMS error (RMSE)* (root-mean-squared error) is more informative of the error magnitude: it takes the square root of the average squared error, so it gives an idea of the typical error (whether positive or negative) in the same scale as that used for the original outcome variable. As we might expect, the RMS error for the validation data (161.5 thousand dollars), which the model is seeing for the first time in making these predictions, is larger than for the training data (≈ 0 thousand dollars), which were used in training the model. The other measures are discussed in Chapter 5.

9. *Interpret the results*. At this stage, we would typically try other prediction algorithms (e.g., regression trees) and see how they do error-wise. We might also try different "settings" on the various models (e.g., we could use the *best subsets* option in multiple linear regression to choose a reduced set of variables that might perform better with the validation data). After choosing the best model—typically, the model with the lowest error on the validation data while also recognizing that "simpler is better"—we use that model to predict the output variable in fresh data. These steps are covered in more detail in the analysis of cases.

10. *Deploy the model*. After the best model is chosen, it is applied to new data to predict TOTAL VALUE for homes where this value is unknown. This was, of course, the original purpose. Predicting the output value for new records is called *scoring*. For predictive tasks, scoring produces predicted numerical values. For classification tasks, scoring produces classes and/or propensities. Table 2.14 shows an example of a data frame with three homes to be scored using our linear regression model. Note that all the required predictor columns are present, and the output column is absent.

Table 2.10 Outlier in West Roxbury Data

FLOORS	ROOMS
15	8
2	10
1.5	6
1	6

Table 2.11 Predictions (fitted values) for a sample of training data

code for fitting a regression model to training data (West Roxbury)

```
reg <- lm(TOTAL_VALUE ~ .-TAX, data = housing.df, subset = train.rows) # remove TAX
tr.res <- data.frame(train.data$TOTAL_VALUE, reg$fitted.values, reg$residuals)
head(tr.res)
```

Partial Output

```
> head(tr.res)
     train.data.TOTAL_VALUE reg.fitted.values reg.residuals
270                   314.2          328.2517     -14.05167
4847                  397.3          380.2328      17.06716
4849                  382.0          437.0615     -55.06152
3242                  321.3          346.5812     -25.28116
```

```
5109                 361.3          348.2637       13.03635
4551                 351.6          330.3665       21.23354
```

Table 2.12 Predictions for a sample of validation data

code for applying the regression model to predict validation set (West Roxbury)

```
pred <- predict(reg, newdata = valid.data)
vl.res <- data.frame(valid.data$TOTAL_VALUE, pred, residuals =
          valid.data$TOTAL_VALUE - pred)
head(vl.res)
```

Partial Output

```
> head(vl.res)
   valid.data.TOTAL_VALUE     pred    residuals
4415                    397.8 366.5991  31.200864
361                     359.3 339.5305  19.769488
3072                    532.9 576.1689 -43.268873
3186                    762.5 692.3698  70.130192
4951                    470.4 409.6396  60.760404
2121                    413.6 409.2843   4.315715
```

Table 2.13 Prediction Error Metrics for training and validation data (error figures are in thousands of $)

code for computing model evaluation metrics

```
library(forecast)
# compute accuracy on training set
accuracy(reg$fitted.values, train.data$TOTAL_VALUE)

# compute accuracy on prediction set
pred <- predict(reg, newdata = valid.data)
accuracy(pred, valid.data$TOTAL_VALUE)
```

Partial Output

```
> accuracy(reg$fitted.values, train.data$TOTAL_VALUE)
                    ME       RMSE        MAE          MPE        MAPE
Test set 1.483808e-15    42.47891   32.47486    -1.095015    8.432433

> accuracy(pred, valid.data$TOTAL_VALUE)
                ME     RMSE       MAE       MPE      MAPE
Test set -1.22575 43.29939 32.26393 -1.376272 8.414937
```

Table 2.14 Data frame with three records to be scored

```
# new.data can be read from a csv file, or defined directly in R.

> new.data
     TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS
     TAX LOT.SQFT YR.BUILT GROSS.AREA LIVING.AREA FLOORS ROOMS BEDROOMS
100 3818     4200     1960       2670        1710    2.0    10        4
101 3791     6444     1940       2886        1474    1.5     6        3
102 4275     5035     1925       3264        1523    1.0     6        2
    FULL.BATH HALF.BATH KITCHEN FIREPLACE REMODEL
100         1         1       1         1    None
101         1         1       1         1    None
102         1         0       1         0  Recent

> pred <- predict(reg, newdata = new.data)
> pred
     100      101      102
303.5358 301.3919 339.8642
```

2.7 Using R for Data Mining on a Local Machine

An important aspect of the data mining process is that the heavy-duty analysis does not necessarily require a huge number of records. The dataset to be analyzed may have millions of records, of course, but in applying multiple linear regression or applying a classification tree, the use of a sample of 20,000 is likely to yield as accurate an answer as that obtained when using the entire dataset. The principle involved is the same as the principle behind polling: If sampled judiciously, 2000 voters can give an estimate of the entire population's opinion within one or two percentage points. (See "How Many Variables and How Much Data" in Section 2.12 for further discussion.)

Therefore, in most cases, the number of records required in each partition (training, validation, and test) can be accommodated within the memory limit allowed in R (to check and increase memory limit in R use function *memory.limit()*).

When we apply Big Data analytics in R, it might be useful to remove unused objects (function *rm()*) and call the garbage collection (function *gc()*) afterwards.

2.8 Automating Data Mining Solutions

In most supervised data mining applications, the goal is not a static, one-time analysis of a particular dataset. Rather, we want to develop a model that can be used on an ongoing basis to predict or classify new records. Our initial analysis will be in prototype mode, while we explore and define the problem and test different models. We will follow all the steps outlined earlier in this chapter.

At the end of that process, we will typically want our chosen model to be deployed in automated fashion. For example, the US Internal Revenue Service (IRS) receives several hundred million tax returns per year—it does not want to have to pull each tax return out into an Excel sheet or other environment separate from its main database to determine the predicted probability that the return is fraudulent. Rather, it would prefer that determination to be made as part of the normal tax filing environment and process. Music streaming services, such as Pandora or Spotify, need to determine "recommendations" for next songs quickly for each of millions of users; there is no time to extract the data for manual analysis.

In practice, this is done by building the chosen algorithm into the computational setting in which the rest of the process lies. A tax return is entered directly into the IRS system by a tax preparer, a predictive algorithm is immediately applied to the new data in the IRS system, and a predicted classification is decided by the algorithm. Business rules would then determine what happens with that classification. In the IRS case, the rule might be "if no predicted fraud, continue routine processing; if fraud is predicted, alert an examiner for possible audit."

This flow of the tax return from data entry, into the IRS system, through a predictive algorithm, then back out to a human user is an example of a "data pipeline." The different components of the system communicate with one another via Application Programming Interfaces (APIs) that establish locally valid rules for transmitting data and associated communications. An API for a data mining algorithm would establish the required elements for a predictive algorithm to work—the exact predictor variables, their order, data formats, etc. It would also establish the requirements for communicating the results of the algorithm. Algorithms to be used in an automated data pipeline will need to be compliant with the rules of the APIs where they operate.

Finally, once the computational environment is set and functioning, the data miner's work is not done. The environment in which a model operates is typically dynamic, and predictive models often have a short shelf life—one leading consultant finds they rarely continue to function effectively for more than a year. So, even in a fully deployed state, models must be periodically checked and re-evaluated. Once performance flags, it is time to return to prototype mode and see if a new model can be developed.

In this book, our focus will be on the prototyping phase—all the steps that go into properly defining the model and developing and selecting a model. You should be aware, though, that most of the actual work of implementing a data mining model lies in the automated deployment phase. Much of this work is not in the analytic domain; rather, it lies in the domains of databases and computer science, to assure that detailed nuts and bolts of an automated dataflow all work properly.

Data Mining Software: The State of the Market

by Herb Edelstein*

The data mining market has changed in some important ways since the last edition of this book. The most significant trends have been the increasing volume of information available and the growing use of the cloud for data storage and analytics. Data mining and analysis have evolved to meet these new demands.

The term "Big Data" reflects the surge in the amount and types of data collected. There is still an enormous amount of transactional data, data warehousing data, scientific data, and clickstream data. However, adding to the massive storage requirements of traditional data is the influx of information from unstructured sources (e.g., customer service calls and images), social media, and more recently the Internet of Things, which produces a flood of sensor data. The number of organizations collecting such data has greatly increased as virtually every business is expanding in these areas.

Rapid technological change has been an important factor. The price of data storage has dropped precipitously. At this writing, hard disk storage has fallen to about $50 per terabyte and solid state drives are about $200 per terabyte. Concomitant with the decrease in storage costs has been a dramatic increase in bandwidth at ever lower costs. This has enabled the spread of cloud-based computing. Cloud-based computing refers to using remote platforms for storage, data management, and now analysis. Because scaling up the hardware and software infrastructure for Big Data is so complex, many organizations are entrusting their data to outside vendors. The largest cloud players (Amazon, Google, IBM, and Microsoft) are each reporting annual revenues in excess of US$5 billion, according to *Forbes* magazine.

Managing this much data is a challenging task. While traditional relational DBMSs—such as Oracle, Microsofts SQL Server, IBMs DB2, and SAPs Adaptive Server Enterprise (formerly Sybase)—are still among the leading data management tools, open source DBMSs, such as Oracles MySQL are becoming increasingly popular. In addition, nonrelational data storage is making headway in storing extremely large amounts of data. For example, Hadoop, an open source tool for managing large distributed database architectures, has become an important player in the cloud database space. However, Hadoop is a tool for the application developer community rather than end users. Consequently, many of the data mining analytics vendors such as SAS have built interfaces to Hadoop.

All the major database management system vendors offer data mining capabilities, usually integrated into their DBMS. Leading products include Microsoft SQL Server Analysis Services, Oracle Data Mining, and Teradata Warehouse Miner. The target user for embedded data mining is a database professional. Not surprisingly, these products take advantage of database functionality, including using the DBMS to transform variables, storing models in the database, and extending the data access language to include model-building and scoring the database. A few products also supply a separate graphical interface for building data mining models. Where the DBMS has parallel processing capabilities, embedded data mining tools will generally take advantage of it, resulting in greater performance. As with the data mining suites described below, these tools offer an assortment