5 Spending our data

There are several steps to create a useful model, including parameter estimation, model selection and tuning, and performance assessment. At the start of a new project, there is usually an initial finite pool of data available for all these tasks. How should the data be applied to these steps? The idea of *data spending* is an important first consideration when modeling, especially as it relates to empirical validation.

When there are copious amounts of data available, a smart strategy is to allocate specific subsets of data for different tasks, as opposed to allocating the largest possible amount to the model parameter estimation only. There may be questions about many modeling project steps that must be answered with limited prior knowledge. For example, one possible strategy (when both data and predictors are abundant) is to spend a specific subset of data to determine which predictors are informative, before considering parameter estimation at all.

> As data are reused for multiple tasks, certain risks increase, such as the risks of adding bias or large effects from methodological errors.

If the initial pool of data available is not huge, there will be some overlap of how and when our data is "spent" or allocated, and a solid methodology for data spending is important. This chapter demonstrates the basics of *splitting* our initial pool of samples for different purposes.

5.1 COMMON METHODS FOR SPLITTING DATA

The primary approach for empirical model validation is to split the existing pool of data into two distinct sets. Some observations are used to develop and optimize the model. This *training set* is usually the majority of the data. These data are a sandbox for model building where different models can be fit, feature engineering strategies are investigated, and so on. We as modeling practitioners spend the vast majority of the modeling process using the training set as the substrate to develop the model.

The other portion of the observations are placed into the *test set*. This is held in reserve until one or two models are chosen as the methods that are most likely to succeed. The test set is then used as the final arbiter to determine the efficacy of the model. It is critical to only look at the test set once; otherwise, it becomes part of the modeling process.

How should we conduct this split of the data? This depends on the context.

Suppose we allocate 80% of the data to the training set and the remaining 20% for testing. The most common method is to use simple random sampling. The **rsample** package has tools for making data splits such as this; the function `initial_split()` was created for this purpose. It takes the data frame as an argument as well as the proportion to be placed into training. Using the previous data frame produced by the code snippet from the summary in Section 4.2:

```
library(tidymodels)
tidymodels_prefer()

# Set the random number stream using `set.seed()` so that the results can be
# reproduced later.
set.seed(123)

# Save the split information for an 80/20 split of the data
ames_split <- initial_split(ames, prop = 0.80)
ames_split
#> <Analysis/Assess/Total>
#> <2344/586/2930>
```

The printed information denotes the amount of data in the training set ($n = 2,344$), the amount in the test set ($n = 586$), and the size of the original pool of samples ($n = 2,930$).

The object `ames_split` is an `rsplit` object and only contains the partitioning information; to get the resulting data sets, we apply two more functions:

```
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)

dim(ames_train)
#> [1] 2344   74
```

These objects are data frames with the same *columns* as the original data but only the appropriate *rows* for each set.

Simple random sampling is appropriate in many cases but there are exceptions. When there is a dramatic *class imbalance* in classification problems, one class occurs much less frequently than another. Using a simple random sample may haphazardly allocate these infrequent samples disproportionately into the training or test set. To avoid this, *stratified sampling* can be used. The training/test split is conducted separately within each class and then these subsamples are combined into the overall training and test set. For regression problems, the outcome data can be artificially binned into *quartiles* and then stratified sampling conducted four separate times. This is an effective method for keeping the distributions of the outcome similar between the training and test set.

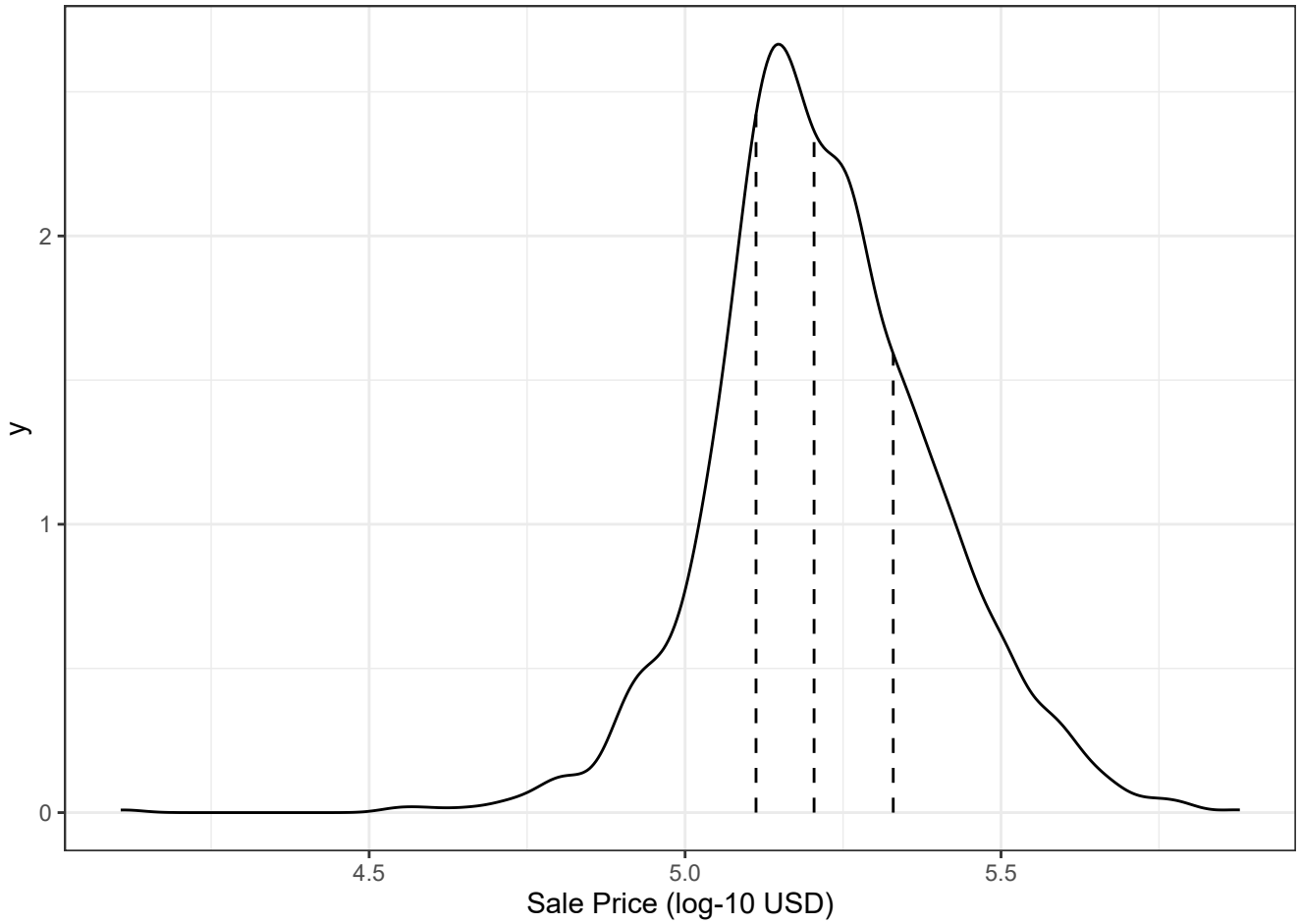

Figure 5.1: The distribution of the sale price (in log units) for the Ames housing data. The vertical lines indicate the quartiles of the data.

The distribution of the sale price outcome for the Ames housing data is shown in Figure 5.1. As previously discussed, the sale price distribution is right-skewed, with proportionally more inexpensive houses than expensive houses on either side of the center of the distribution. The worry here is that the more expensive houses would not be represented in the training set well with simple splitting; this would increase the risk that our model would be ineffective at predicting the price for such properties. The dotted vertical lines in Figure 5.1 indicate the four quartiles for these data. A stratified random sample would conduct the 80/20 split within each of these data subsets and then pool the results together. In **rsample**, this is achieved using the `strata` argument:

```
set.seed(123)
ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)

dim(ames_train)
#> [1] 2342   74
```

Only a single column can be used for stratification.

> There is very little downside to using stratified sampling.

Are there situations when random sampling is not the best choice? One case is when the data have a significant *time* component, such as time series data. Here, it is more common to use the most recent data as the test set. The **rsample** package contains a function called `initial_time_split()` that is very similar to `initial_split()`. Instead of using random sampling, the `prop` argument denotes what proportion of the first part of the data should be used as the training set; the function assumes that the data have been pre-sorted in an appropriate order.

5.2 WHAT PROPORTION SHOULD BE USED?

The proportion of data that should be allocated for splitting is highly dependent on the context of the problem at hand. Too little data in the training set hampers the model's ability to find appropriate parameter estimates. Conversely, too little data in the test set lowers the quality of the performance estimates. There are parts of the statistics community that eschew test sets in general because they believe all of the data should be used for parameter estimation. While there is merit to this argument, it is good modeling practice to have an unbiased set of observations as the final arbiter of model quality. A test set should be avoided only when the data are pathologically small.

5.3 WHAT ABOUT A VALIDATION SET?

Previously, when describing the goals of data splitting, we singled out the test set as the data that should be used to conduct a proper evaluation of model performance on the final model(s). This begs the question of, "How can we tell what is best if we don't measure performance until the test set?"

It is common to hear about *validation sets* as an answer to this question, especially in the neural network and deep learning literature. The validation set was originally defined in the early days of neural networks when researchers realized that measuring performance by re-predicting the training set samples led to results that were overly optimistic (significantly, unrealistically so). This led to models that overfit, meaning that they performed very well on the training set but poorly on the test set[9]. To combat this issue, a small validation set of data were held back and used to measure performance as the network was trained. Once the validation set error rate began to rise, the training would be halted. In other words, the validation set was a means to get a rough sense of how well the model performed prior to the test set.

> It is largely semantics as to whether validation sets are a subset of the training set or a third allocation in the initial split of the data.

Validation sets are discussed more in Section 10.2.2 as a special case of *resampling* methods that are used on the training set.

5.4 MULTI-LEVEL DATA

With the Ames housing data, a property is considered to be the *independent experimental unit*. It is safe to assume that, statistically, the data from a property are independent. For other applications, that is not always the case:

- For longitudinal data, the same independent experimental unit can be measured over multiple time points. An example would be a human subject in a medical trial.
- A batch of manufactured product might also be considered the independent experimental unit. In repeated measures designs, replicate data points from a batch are collected.

- Johnson et al. (2018) report an experiment where different trees were sampled across the top and bottom portions of a stem. Here, the tree is the experimental unit and the data hierarchy is sample within stem position within tree.

Chapter 9 of Kuhn and Johnson (2020) contains other examples.

In these situations, the data set will have multiple rows per experimental unit. Simple resampling across rows would lead to some data within an experimental unit being in the training set and others in the test set. Data splitting should occur at the independent experimental unit level of the data. For example, to produce an 80/20 split of the data, 80% of the experimental units should be allocated for the training set.

5.5 OTHER CONSIDERATIONS

Throughout this book, notice which data are exposed to the model at any given time. Remember that it is critical to quarantine the test set from any model building activities.

> The problem of *information leakage* occurs when data outside of the training set are used in the modeling process.

For example, in a machine learning competition, the test set data might be provided without the true outcome values so that the model can be scored and ranked. One potential method for improving the score might be to fit the model using the training set points that are most similar to the test set values. While the test set isn't directly used to fit the model, it still has a heavy influence. In general, this technique is highly problematic since it reduces the *generalization error* of the model to optimize performance on a specific data set. There are more subtle ways that the test set data can be utilized during training. Keeping the training data in a separate data frame from the test set is one small check to make sure that information leakage does not occur by accident.

In later chapters we discuss techniques to subsample the training set to mitigate specific issues (e.g., class imbalances). This is a valid and common technique that deliberately results in the training set data diverging from the population from which the data were drawn. It is critical that the test set

continue to mirror what the model would encounter *in the wild*. In other words, the test set should always resemble new data that will be given to the model.

Finally, at the beginning of this chapter, we warned about using the same data for different tasks. Chapter 10 will discuss solid, data-driven methodologies for data usage that will reduce the risks related to bias, overfitting, and other issues. Many of these methods mirror the data-splitting tools shown in this chapter.

5.6 CHAPTER SUMMARY

Data splitting is the fundamental tool for empirical validation of models. Even in the era of unrestrained data collection, a typical modeling project has a limited amount of appropriate data and wise "spending" of a project's data is necessary. In this chapter, we discussed several strategies for partitioning the data into distinct groups for modeling and evaluation.

At this checkpoint, the important code snippets are:

```
library(tidymodels)
data(ames)
ames <- ames %>% mutate(Sale_Price = log10(Sale_Price))

set.seed(123)
ames_split <- initial_split(ames, prop = 0.80, strata = Sale_Price)
ames_train <- training(ames_split)
ames_test  <-  testing(ames_split)
```

REFERENCES

Johnson, D, P Eckart, N Alsamadisi, H Noble, C Martin, and R Spicer. 2018. "Polar Auxin Transport Is Implicated in Vessel Differentiation and Spatial Patterning During Secondary Growth in Populus." *American Journal of Botany* 105 (2): 186–96.

Kuhn, M, and K Johnson. 2020. *Feature Engineering and Selection: A Practical Approach for Predictive Models*. CRC Press.

9. This is discussed in much greater detail in Section 12.3.↩