Look at the Big Picture

Welcome to Machine Learning Housing Corporation! The first task you are asked to perform is to build a model of housing prices in California using the California census data. This data has metrics such as the population, median income, median housing price, and so on for each block group in California. Block groups are the smallest geographical unit for which the US Census Bureau publishes sample data (a block group typically has a population of 600 to 3,000 people). We will just call them "districts" for short.

Your model should learn from this data and be able to predict the median housing price in any district, given all the other metrics.

Since you are a well-organized data scientist, the first thing you do is to pull out your Machine Learning project checklist. You can start with the one in Appendix B; it should work reasonably well for most Machine Learning projects but make sure to adapt it to your needs. In this chapter we will go through many checklist items, but we will also skip a few, either because they are self-explanatory or because they will be discussed in later chapters.

Frame the Problem

The first question to ask your boss is what exactly is the business objective; building a model is probably not the end goal. How does the company expect to use and benefit from this model? This is important because it will determine how you frame the problem, what algorithms you will select, what performance measure you will use to evaluate your model, and how much effort you should spend tweaking it.

Your boss answers that your model's output (a prediction of a district's median housing price) will be fed to another Machine Learning system (see Figure 2-2), along with many other *signals*.[3] This downstream system will determine whether it is worth investing in a given area or not. Getting this right is critical, as it directly affects revenue.

3 A piece of information fed to a Machine Learning system is often called a *signal* in reference to Shannon's information theory: you want a high signal/noise ratio.

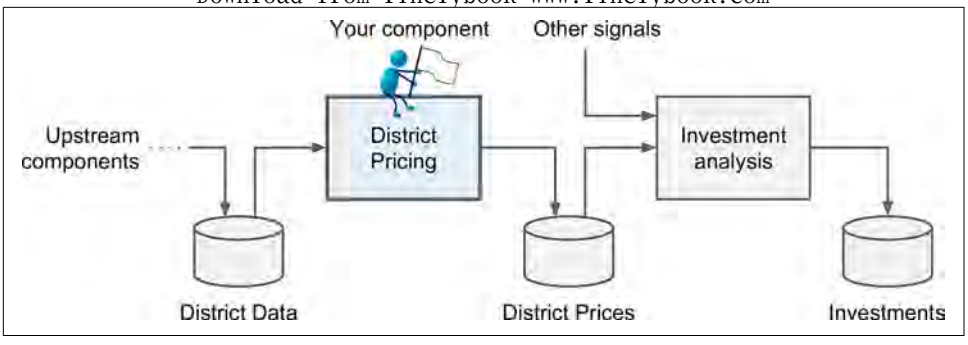

Figure 2-2. A Machine Learning pipeline for real estate investments

Pipelines

A sequence of data processing *components* is called a data *pipeline*. Pipelines are very common in Machine Learning systems, since there is a lot of data to manipulate and many data transformations to apply.

Components typically run asynchronously. Each component pulls in a large amount of data, processes it, and spits out the result in another data store, and then some time later the next component in the pipeline pulls this data and spits out its own output, and so on. Each component is fairly self-contained: the interface between components is simply the data store. This makes the system quite simple to grasp (with the help of a data flow graph), and different teams can focus on different components. Moreover, if a component breaks down, the downstream components can often continue to run normally (at least for a while) by just using the last output from the broken component. This makes the architecture quite robust.

On the other hand, a broken component can go unnoticed for some time if proper monitoring is not implemented. The data gets stale and the overall system's performance drops.

The next question to ask is what the current solution looks like (if any). It will often give you a reference performance, as well as insights on how to solve the problem. Your boss answers that the district housing prices are currently estimated manually by experts: a team gathers up-to-date information about a district (excluding median housing prices), and they use complex rules to come up with an estimate. This is costly and time-consuming, and their estimates are not great; their typical error rate is about 15%.

Okay, with all this information you are now ready to start designing your system. First, you need to frame the problem: is it supervised, unsupervised, or Reinforcement Learning? Is it a classification task, a regression task, or something else? Should

you use batch learning or online learning techniques? Before you read on, pause and try to answer these questions for yourself.

Have you found the answers? Let's see: it is clearly a typical supervised learning task since you are given *labeled* training examples (each instance comes with the expected output, i.e., the district's median housing price). Moreover, it is also a typical regression task, since you are asked to predict a value. More specifically, this is a *multivariate regression* problem since the system will use multiple features to make a prediction (it will use the district's population, the median income, etc.). In the first chapter, you predicted life satisfaction based on just one feature, the GDP per capita, so it was a *univariate regression* problem. Finally, there is no continuous flow of data coming in the system, there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory, so plain batch learning should do just fine.

If the data was huge, you could either split your batch learning work across multiple servers (using the *MapReduce* technique, as we will see later), or you could use an online learning technique instead.

Select a Performance Measure

Your next step is to select a performance measure. A typical performance measure for regression problems is the Root Mean Square Error (RMSE). It measures the *standard deviation*[4] of the errors the system makes in its predictions. For example, an RMSE equal to 50,000 means that about 68% of the system's predictions fall within $50,000 of the actual value, and about 95% of the predictions fall within $100,000 of the actual value.[5] Equation 2-1 shows the mathematical formula to compute the RMSE.

Equation 2-1. Root Mean Square Error (RMSE)

$$\text{RMSE}(\mathbf{X}, h) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}\left(h\left(\mathbf{x}^{(i)}\right) - y^{(i)}\right)^2}$$

4 The standard deviation, generally denoted σ (the Greek letter sigma), is the square root of the *variance*, which is the average of the squared deviation from the mean.

5 When a feature has a bell-shaped *normal distribution* (also called a *Gaussian distribution*), which is very common, the "68-95-99.7" rule applies: about 68% of the values fall within 1σ of the mean, 95% within 2σ, and 99.7% within 3σ.