***CHAPTER 2*. *End-to-End Machine Learning Project***

In this chapter you will work through an example project end to end, pretending to:

* ***Look at the big picture.***
* ***Get the data.***
* ***Discover and visualize the data to gain insights.***
* ***Prepare the data for Machine Learning algorithms.***
* ***Select a model and train it.***
* ***Fine-tune your model.***
* ***Present your solution.***
* ***Launch, monitor, and maintain your system.***

When you are learning about Machine Learning, it is best to experiment with real-world data, not artificial datasets. Here are a few places you can look to get data:

* Popular open data repositories
  + UC Irvine Machine Learning Repository
  + Kaggle datasets
  + Amazon’s AWS datasets
* Meta portals (they list open data repositories)
  + Data Portals
  + OpenDataMonitor
  + Quandl
* Other pages listing many popular open data repositories
  + Wikipedia’s list of Machine Learning datasets
  + Quora.com
  + The datasets subreddit

In this chapter we’ll use the California Housing Prices dataset from the StatLib repository. This dataset is based on data from the 1990 California census. It is not exactly recent (a nice house in the Bay Area was still affordable at the time), but it has many qualities for learning, so we will pretend it is recent data. For teaching purposes I’ve added a categorical attribute and removed a few features.

**Look at the Big Picture**

**TASK:** *Your first task is to use California census data to build a model of housing prices in the state.*

**ABOUT DATA:***This data includes metrics such as the population, median income, and median housing price 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 call them “districts” for short.*

**EXPECTED:***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 should do is 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**

First questions to ask your boss:

* **What exactly the business objective is?**
  + **Why to ask?**
    - Building a model is probably not the end goal.
  + **Boss answers:** 
    - Your model’s output (a prediction of a district’s median housing price) will be fed to another Machine Learning system along with many other signals[[1]](#footnote-1).
* **How does the company expect to use and benefit from this model?**
  + **Why to ask?**
    - Knowing the objective is important because it will determine
      * how you frame the problem,
      * which algorithms you will select,
      * which performance measure you will use to evaluate your model, and
      * how much effort you will spend tweaking it.
  + **Boss answers:** 
    - 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.
* **The next question to ask your boss is what the current solution looks like (if any).**
  + **Why to ask?**
    - The current situation will often give you a reference for performance, as well as insights on how to solve the problem.
  + **Boss answers:**
    - The district housing prices are currently estimated manually by experts: a team gathers up-to-date information about a district, and when they cannot get the median housing price, they estimate it using complex rules. This is costly and time-consuming, and their estimates are not great; in cases where they manage to find out the actual median housing price, they often realize that their estimates were off by more than 20%. This is why the company thinks that it would be useful to train a model to predict a district’s median housing price, given other data about that district.

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. *My (the reader’s - the one that prepared this outline document) answers are those marked in red during mentioning in punctuating bullet points. Let us see the results.*

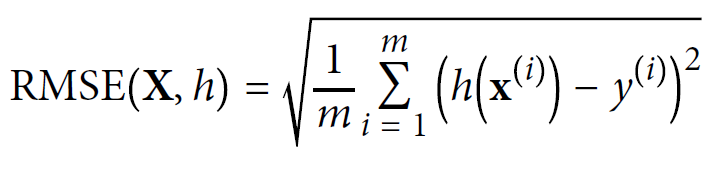
Have you found the answers? Let’s see:

* **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).
* **It is also a typical regression task**
  + since you are asked to predict a value.
  + More specifically, this is a *multiple regression* problem, since the system will use multiple features to make a prediction (it will use the district’s population, the median income, etc.).
  + It is also a *univariate regression* problem, since we are only trying to predict a single value for each district. If we were trying to predict multiple values per district, it would be a *multivariate regression* problem.
* **Plain batch learning**
  + There is no continuous flow of data coming into the system, there is no particular need to adjust to changing data rapidly, and the data is small enough to fit in memory.

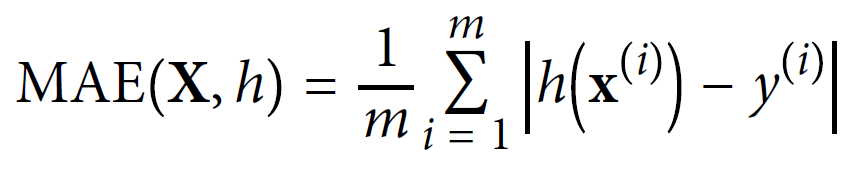
|  |
| --- |
| If the data were huge, you could either split your batch learning work across multiple servers (using the MapReduce technique) or use an online learning technique. |

**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 gives an idea of how much error the system typically makes in its predictions, with a higher weight for large errors. The mathematical formula to compute the RMSE.

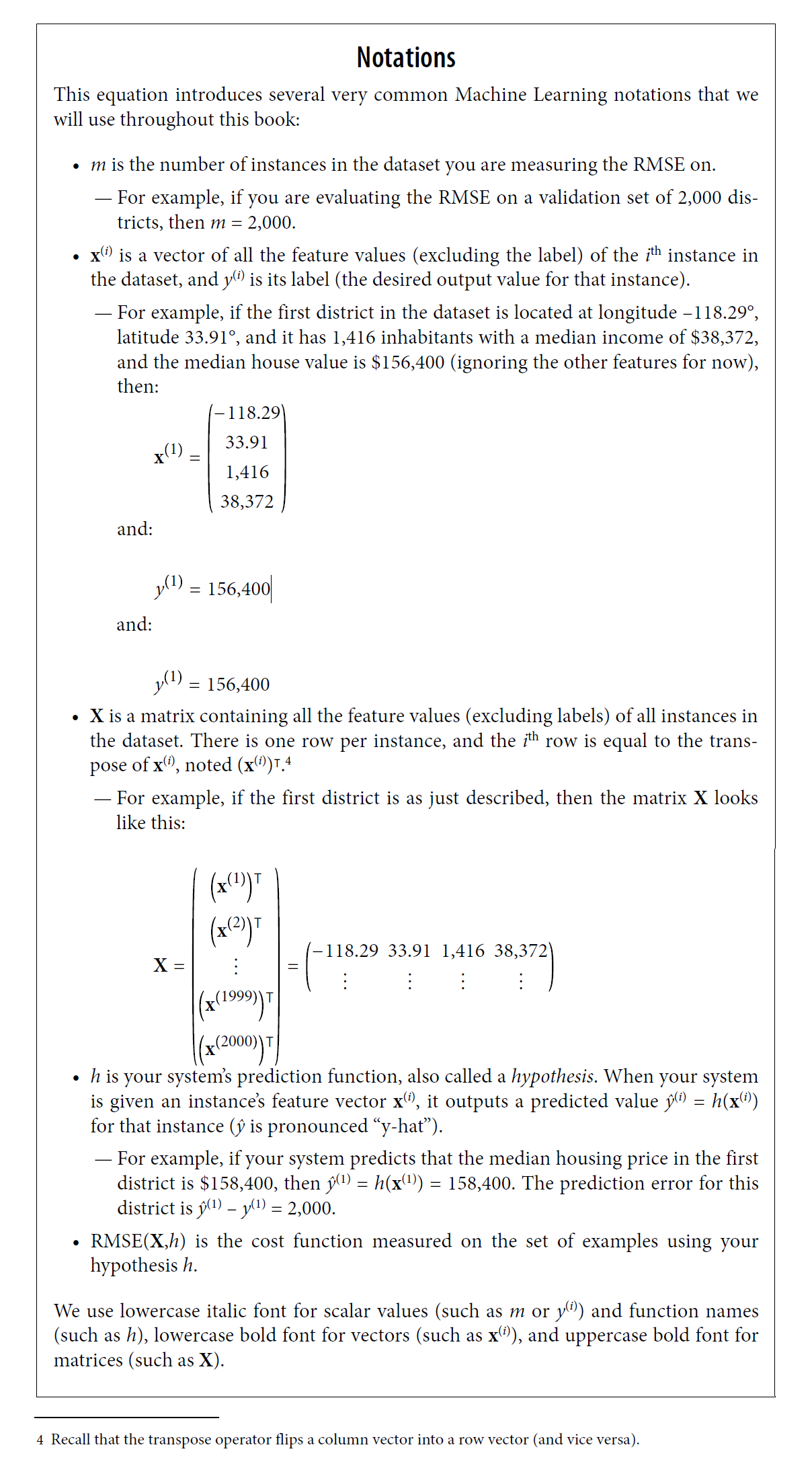


Even though the RMSE is generally the preferred performance measure for regression tasks, in some contexts you may prefer to use another function. For example, suppose that there are many outlier districts. In that case, you may consider using the *mean absolute error* (MAE, also called the average absolute deviation):



***Both the RMSE and the MAE are ways to measure the distance between two vectors: the vector of predictions and the vector of target values.*** Various distance measures, or *norms*, are possible:

* Computing the root of a sum of squares (RMSE) corresponds to the *Euclidean norm*: this is the notion of distance you are familiar with. It is also called the ℓ2 *norm*, noted ∥ ・ ∥2 (or just ∥ ・ ∥).
* Computing the sum of absolutes (MAE) corresponds to the ℓ1 *norm*, noted ∥ ・ ∥1. This is sometimes called the *Manhattan norm* because it measures the distance between two points in a city if you can only travel along orthogonal city blocks.
* More generally, the ℓ*k norm* of a vector **v** containing *n* elements is defined as ∥**v**∥*k* = (|*v*0|*k* + |*v*1|*k* + ... + |*vn*|*k*)1/*k*. ℓ0 gives the number of nonzero elements in the vector,and ℓ∞ gives the maximum absolute value in the vector.
* ***The higher the norm index, the more it focuses on large values and neglects small ones. Therefore the RMSE is more sensitive to outliers than the MAE. But when outliers are exponentially rare (like in a bell-shaped curve), the RMSE performs very well and is generally preferred.***

****

**Check the Assumptions**

Lastly, it is good practice to list and verify the assumptions that have been made so far (by you or others); this can help you catch serious issues early on. For example,

* The district prices that your system outputs are going to be fed into a downstream Machine Learning system, and you assume that these prices are going to be used as such. But what if the downstream system converts the prices into categories (e.g., “cheap,” “medium,” or “expensive”) and then uses those categories instead of the prices themselves?
  + In this case, getting the price perfectly right is not important at all; your system just needs to get the category right.
    - If that’s so, then the problem should have been framed as a classification task, not a regression task.
    - You don’t want to find this out after working on a regression system for months.
  + Fortunately, after talking with the team in charge of the downstream system, you are confident that they do indeed need the actual prices, not just categories.
    - Great! You’re all set, the lights are green, and you can start coding now!

**Exercises**

The following exercises are all based on this chapter’s housing dataset:

1. Try a Support Vector Machine regressor (sklearn.svm.SVR) with various hyperparameters, such as kernel="linear" (with various values for the C hyperparameter) or kernel="rbf" (with various values for the C and gamma hyperparameters). Don’t worry about what these hyperparameters mean for now.
2. How does the best SVR predictor perform?
3. Try replacing GridSearchCV with RandomizedSearchCV.
4. Try adding a transformer in the preparation pipeline to select only the most important attributes.
5. Try creating a single pipeline that does the full data preparation plus the final prediction.
6. Automatically explore some preparation options using GridSearchCV.

1. A piece of information fed to a Machine Learning system is often called a *signal*, in reference to Claude Shannon’s information theory, which he developed at Bell Labs to improve telecommunications. His theory: you want a high signal-to-noise ratio. [↑](#footnote-ref-1)