Chapter 1

Building Good Trainign Datasets: Data Preprocessing

1.1 Introduction

The quality of the data and the amount of useful information that it contains are key factors that determine how well a machine learning algorithm can learn.

It is absolutely critical to ensure that we examine and preprocess a dataset before we feed it to a machine learning algorithm.

Observation 1.1.1
The topics that we will cover in this chapter are as follows:

- Removing and imputing missing values from the dataset
- Getting categorical data into shape for machine learning algorithms
- Selecting relevant features for the model construction

1.2 Dealing With Missing Data

Sometimes data may be missing due to various reasons.

Observation 1.2.1
There could have been an error in the data collection process, certain measurements may not be applicable, or particular fields could have been simply left blank in a survey, for example. We typically see missing values as blank spaces in our data table or as placeholder strings such as `NaN`, which stands for *not a number* or `NULL`.

We will work through several practical techniques for dealing with missing values by removing entries from our dataset or imputing missing values from other training examples and features.

1.3 Identifying Missing Values in Tabular Data

Before we discuss several techniques for dealing with missing values, let's create a simple example DataFrame from a comma-separated values (`CSV`) file to get a better grasp of the problem:

```
1 import pandas as pd
2 from io import StringIO
3 csv_data = \
4 '''A,B,C,D
5 1.0,2.0,3.0,4.0
6 5.0,6.0,,8.0
7 10.0,11.0,12.0,'''
8 # If you are using Python 2.7, you need
```

```
# to convert the string to unicode:
# csv_data = unicode(csv_data)
df = pd.read_csv(StringIO(csv_data))
print(df)
```

Code 1.1: CSV with Missing Data.

The output is the following:

```
     A     B     C    D
0  1.0   2.0   3.0  4.0
1  5.0   6.0   NaN  8.0
2 10.0  11.0  12.0  NaN
```

Code 1.2: CSV with Missing Data Output

Using the preceding code, we read CSV-formatted data into a pandas DataFrame via the read_csv function and noticed that the two missing cells were replaced by NaN. The StringIO function in the preceding code example was simply used for the purposes of illustration. It allowed us to read the string assigned to csv_data into a pandas DataFrame as if it was a regular CSV file on our hard drive.

Observation 1.3.1

For a larger DataFrame, it can be tedious to look for missing values manually; in this case, we can use the isnull method to return a DataFrame with Boolean values that indicate whether a cell contains a numeric value (False) or if data is missing (True).

Using the sum method, we can then return the number of missing values per column as follows:

```
print(df.isnull().sum())
```

Code 1.3: Check Missing Data per Row

The output is this:

```
A    0
B    0
C    1
D    1
dtype: int64
```

Code 1.4: Output Missing Data Per Row

Although scikit-learn was originally developed for working with NumPy arrays only, it can sometimes be more convenient to preprocess data using pandas' DataFrame.

Nowadays, most scikit-learn functions support DataFrame objects as inputs, but since NumPy array handling is more mature in the scikit-learn API, it is recommended to use NumPy arrays when possible. Note that you can always access the underlying NumPy array of a DataFrame via the values attribute before you feed it into a scikit-learn estimator:

```
print(df.values)
```

Code 1.5: Dataframe df

```
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,  nan,   8.],
       [ 10.,  11.,  12.,  nan]])
```

Code 1.6: Values of Dataframe `df`

1.3.1 Eliminating Training Examples or Features with Missing Values

One of the easiest ways to deal with missing data is simply to remove the corresponding features (columns) or training examples (rows) from the dataset entirely; rows with missing values can easily be dropped via the `dropna` method:

```
print(df.dropna(axis=0))

output:

     A    B    C    D
0  1.0  2.0  3.0  4.0
```

Code 1.7: Droping Rows from `df`

Similarly, we can drop columns that have at least one NaN in any row by setting the axis argument to 1:

```
print(df.dropna(axis=1))

output:

     A     B
0  1.0   2.0
1  5.0   6.0
2 10.0  11.0
```

Code 1.8: Dropping Columns from a `DataFrame df`