4.3 The Distance Matrix

May 9, 2019

1 4.3 The Distance Matrix

In many applications, we need the distance between every pair of observations $\mathbf{x}_i$ and $\mathbf{x}_j$ in a data set. How do we represent this information? The most common way is to use an $n \times n$ matrix, where the (i, j)th entry is the distance between $\mathbf{x}_i$ and $\mathbf{x}_j$. That is,

$$D = \begin{pmatrix} d(\mathbf{x}_1, \mathbf{x}_1) & d(\mathbf{x}_1, \mathbf{x}_2) & \cdots & d(\mathbf{x}_1, \mathbf{x}_n) \\ d(\mathbf{x}_2, \mathbf{x}_1) & d(\mathbf{x}_2, \mathbf{x}_2) & \cdots & d(\mathbf{x}_2, \mathbf{x}_n) \\ \vdots & \vdots & \ddots & \vdots \\ d(\mathbf{x}_n, \mathbf{x}_1) & d(\mathbf{x}_n, \mathbf{x}_2) & \cdots & d(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix}.$$

There are a few things we can say about the $n \times n$ distance matrix D.

1. All of the entries of D are non-negative.
2. Because the distance between any observation and itself, $d(\mathbf{x}_i, \mathbf{x}_i)$, is always zero, the *diagonal* elements of this matrix, D_{ii} are all equal to 0.
3. For many distance metrics, including Euclidean and Manhattan distance, d is symmetric, meaning that $d(\mathbf{x}_i, \mathbf{x}_j) = d(\mathbf{x}_i, \mathbf{x}_j)$. Therefore, the matrix D will also be symmetric; that is, the values in the upper triangle will match their reflection in the lower triangle.

How do we calculate the distance matrix for a `DataFrame` consisting of all quantitative variables? For example, suppose we want to calculate the matrix of distances between each of the houses in the Ames housing data set, based on the number of bedrooms, number of bathrooms, and the living area (in square feet).

```
In [1]: %matplotlib inline
        import numpy as np
        import pandas as pd
        pd.options.display.max_rows = 6
        pd.options.display.max_columns = 6

        housing_df = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/mas
                                 sep="\t")

        # extract 3 quantitative variables
        housing_df_quant = housing_df[["Bedroom AbvGr", "Gr Liv Area"]].copy()
        housing_df_quant["Bathrooms"] = (
            housing_df["Full Bath"] +
```

```
        0.5 * housing_df["Half Bath"]
    )
    housing_df_quant
```

```
Out[1]:       Bedroom AbvGr  Gr Liv Area  Bathrooms
    0                   3         1656        1.0
    1                   2          896        1.0
    2                   3         1329        1.5
    ...               ...          ...        ...
    2927                3          970        1.0
    2928                2         1389        1.0
    2929                3         2000        2.5

    [2930 rows x 3 columns]
```

The Long Way: It is possible to create the distance matrix entirely in `pandas`. The idea is to first define a function that calculates the distances between a given observation and all of the other observations:

```
In [2]: def get_euclidean_dists_from_obs(obs):
            return np.sqrt(
                ((housing_df_quant - obs) ** 2).sum(axis=1)
            )

        get_euclidean_dists_from_obs(housing_df_quant.loc[0])
```

```
Out[2]: 0         0.000000
        1       760.000658
        2       327.000382
                   ...
        2927    686.000000
        2928    267.001873
        2929    344.003270
        Length: 2930, dtype: float64
```

The code for this function is very similar to the code that we wrote for Exercise 5 at the end of Section 4.1.

Now, to get a matrix of distances D, we simply need to apply this function to every row of the `DataFrame`. To achieve this, we use the `.apply()` method with `axis=1`:

```
In [3]: D = housing_df_quant.apply(
            get_euclidean_dists_from_obs,
            axis=1
        )
        D
```

```
Out[3]:             0           1           2       ...              2927  \
        0       0.000000  760.000658  327.000382     ...        686.000000
        1     760.000658    0.000000  433.001443     ...         74.006756
```

```
2     327.000382   433.001443    0.000000     ...        359.000348
...          ...          ...         ...     ...               ...
2927  686.000000    74.006756  359.000348     ...          0.000000
2928  267.001873   493.000000   60.010416     ...        419.001193
2929  344.003270  1104.001472  671.000745     ...       1030.001092

            2928         2929
0     267.001873   344.003270
1     493.000000  1104.001472
2      60.010416   671.000745
...          ...          ...
2927  419.001193  1030.001092
2928    0.000000   611.002660
2929  611.002660     0.000000

[2930 rows x 2930 columns]
```

Notice that this is a 2930×2930 symmetric matrix of non-negative numbers, with zeroes along the diagonal, just as we predicted.

The Short Way: There are many packages in Python that calculate distance matrices. One such package is scikit-learn, a machine learning package in Python. Machine learning will be discussed in depth in Chapters 5-8, and we will explore the features of scikit-learn extensively in those chapters. Because distance matrices are important in machine learning, scikit-learn provides functions for calculating distance matrices.

For example, the following code calculates the (Euclidean) distance matrix between all of the houses in the Ames housing data set:

```
In [4]: from sklearn.metrics import pairwise_distances

        D_ = pairwise_distances(housing_df_quant, metric="euclidean")
        D_

Out[4]: array([[   0.        ,  760.00065789,  327.00038226, ...,
                 686.        ,  267.00187265,  344.00327033],
               [ 760.00065789,    0.        ,  433.00144342, ...,
                  74.00675645,  493.        , 1104.00147192],
               [ 327.00038226,  433.00144342,    0.        , ...,
                 359.00034819,   60.01041576,  671.00074516],
               ...,
               [ 686.        ,   74.00675645,  359.00034819, ...,
                   0.        ,  419.00119332, 1030.00109223],
               [ 267.00187265,  493.        ,   60.01041576, ...,
                 419.00119332,    0.        ,  611.00265957],
               [ 344.00327033, 1104.00147192,  671.00074516, ...,
                1030.00109223,  611.00265957,    0.        ]])
```

Notice that the return type is a `numpy` array, instead of a `pandas` `DataFrame`. That is because scikit-learn was designed to work with `numpy` arrays. Although it will accept `pandas` `DataFrames`

as arguments, scikit-learn will convert them `numpy` arrays underneath the hood and return `numpy` arrays.

Fortunately, many of the usual `pandas` operations work on `numpy` arrays as well. For example, to get the maximum value in each row, we can use the `.max()` method with `axis=1`.

```
In [5]: D_.max(axis=1)
```

```
Out[5]: array([ 3986.00028224,  4746.00034239,  4313.00011593, ...,  4672.0002408 ,
                4253.00038208,  3642.        ])
```

2 Exercises

Exercises 1-3 ask you to work with a data set that describes the chemical composition of 1599 red wines (`https://raw.githubusercontent.com/dlsun/data-science-book/master/data/wines/reds.csv`). All 12 variables in this data set are quantitative.

```
In [6]: wines = pd.read_csv("https://raw.githubusercontent.com/dlsun/data-science-book/master/d
        wines.head()
```

```
Out[6]:    fixed acidity  volatile acidity  citric acid   ...     sulphates  alcohol  \
        0            7.4              0.70         0.00   ...          0.56      9.4
        1            7.8              0.88         0.00   ...          0.68      9.8
        2            7.8              0.76         0.04   ...          0.65      9.8
        3           11.2              0.28         0.56   ...          0.58      9.8
        4            7.4              0.70         0.00   ...          0.56      9.4

           quality
        0        5
        1        5
        2        5
        3        6
        4        5

        [5 rows x 12 columns]
```

Exercise 1. Calculate the distance between every pair of wines in this data set.

```
In [7]: wines_matrix = pairwise_distances(wines, metric="euclidean")
        wines_matrix

        wines_as_df = pd.DataFrame(data=wines_matrix, index=wines.index, columns=wines.index)
        wines_as_df.head()
```

```
Out[7]:             0          1          2       ...           1596       1597  \
        0  6.743496e-07  35.860192  20.409705     ...      19.105685  23.322597
        1  3.586019e+01   0.000000  16.404589     ...      27.385901  24.131680
        2  2.040970e+01  16.404589   0.000000     ...      19.919750  19.823713
        3  2.698542e+01  11.257696   7.296300     ...      23.873506  22.600262
```

```
4  6.743496e-07  35.860192  20.409705    ...      19.105685  23.322597

        1598
0  11.036643
1  26.101020
2  12.679709
3  18.879575
4  11.036643

[5 rows x 1599 columns]
```

Exercise 2. Using the distance matrix that you calculated in Exercise 1, calculate the distance of the wine that is most similar to each wine.

```
In [8]: np.fill_diagonal(wines_matrix, np.nan)
        np.nanmin(wines_matrix, axis=1)

Out[8]: array([  6.74349576e-07,   1.54385948e+00,   1.25056949e+00, ...,
                 0.00000000e+00,   4.63842996e-01,   1.89385448e+00])

In [9]: wines_matrix
        len(wines_matrix.min(axis=1))

Out[9]: 1599
```

Exercise 3. Using the distance matrix that you calculated in Exercise 1, determine the identity of the wine that is most similar to each wine.

```
In [10]: wines_as_df #diagonals got changed to NaN because this df was created from matrix and

Out[10]:            0          1          2       ...           1596       1597  \
         0           NaN  35.860192  20.409705    ...      19.105685  23.322597
         1     35.860192        NaN  16.404589    ...      27.385901  24.131680
         2     20.409705  16.404589        NaN    ...      19.919750  19.823713
         ...         ...        ...        ...    ...            ...        ...
         1596  19.105685  27.385901  19.919750    ...            NaN   5.189646
         1597  23.322597  24.131680  19.823713    ...       5.189646        NaN
         1598  11.036643  26.101020  12.679709    ...      11.266973  14.299639

                   1598
         0     11.036643
         1     26.101020
         2     12.679709
         ...         ...
         1596  11.266973
         1597  14.299639
         1598        NaN

         [1599 rows x 1599 columns]
```

```
In [11]: wines_as_df.idxmin(axis=1)

Out[11]: 0          4
         1        752
         2        196
                  ...
         1596    1592
         1597    1594
         1598     569
         Length: 1599, dtype: int64

In [12]: np.nanargmin(wines_matrix, axis=1)

Out[12]: array([   4,  752,  196, ..., 1592, 1594,  569])
```