CHAPTER 12

DIMENSION REDUCTION

12.1 THE NEED FOR DIMENSION REDUCTION

High dimensionality in data science refers to when there are a large number of predictors in the data set. For example, 100 predictors describe a 100-dimensional space. So, why do we need dimension reduction in data science?

1. **Multicollinearity.** Typically, large databases have many predictors. It is unlikely that all of these predictors are uncorrelated. Multicollinearity, which occurs when there is substantial correlation among the predictors, can lead to unstable regression models.
2. **Double-Counting.** Inclusion of predictors which are highly correlated tends to overemphasize a particular aspect of the model, that is, essentially double-counting this aspect. For example, suppose we are trying to estimate the age of youngsters using math knowledge, height, and weight. Since height and weight are correlated, the model is essentially double-counting the physical component of the youngster, as compared to the intellectual component.
3. **Curse of Dimensionality.** As dimensionality increases, the volume of the predictor space grows exponentially, that is, faster than the number of predictors itself. Thus, even for huge sample sizes, the high-dimension space is *sparse*. For example, the empirical rule states that about 68% of normally distributed data lies within one standard deviation of the mean. But, this is for one dimension. For 10 dimensions, only 2% of the data lies within the analogous hypersphere.
4. **Violates Parsimony.** The use of too many predictors also violates the principle of *parsimony*, the scientific principle one sees in many branches of science, that things often behave in a quite economical way. In data science, simplicity (parsimony) should be considered when comparing models, keeping the number of predictors to such a size that would be easily interpreted.
5. **Overfitting.** Keeping too many predictors in the model tends to lead to *overfitting*, in which the generality of the findings is hindered because new data do not behave the same as the training data for all the many predictors.

Data Science Using Python and R, First Edition. Chantal D. Larose and Daniel T. Larose.
© 2019 John Wiley & Sons, Inc. Published 2019 by John Wiley & Sons, Inc.

6. **Miss the Bigger Picture.** Further, analysts should try to keep an eye on the big picture, and analysis solely at the variable level might miss the fundamentals underlying relationships among the predictors. Instead, several predictors might fall naturally into a single group (a *factor* or *component*), which addresses a single aspect of the data. For example, the variables savings account balance, checking account balance, home equity, stock portfolio value, and 401 k balance might all fall together under the single component, *assets*.

In summary, dimension reduction methods use the correlation structure among the predictor variables to accomplish the following:

1. Reduce the number of predictor items.
2. Help ensure that these predictors items are uncorrelated.
3. Provide a framework for interpretability of the results.

12.2 MULTICOLLINEARITY

Data scientists need to guard against *multicollinearity*, a condition where some of the predictor variables are correlated with each other. Multicollinearity leads to instability in the solution space, leading, for example, to regression coefficients you cannot trust, because the coefficient variability is so large. Multicollinearity is an occupational hazard for data scientists, because many of the data sets have dozens if not hundreds of predictors, some of which are often correlated.

Consider Figures 12.1 and 12.2. Figure 12.1 illustrates a situation where the predictors x_1 and x_2 are not correlated with each other; that is, they are orthogonal, or independent. In such a case, the predictors form a solid basis, upon which the response surface y may rest sturdily, thereby providing stable coefficient estimates b_1 and b_2 each with small variability. On the other hand, Figure 12.2 illustrates a multicollinear situation where the predictors x_1 and x_2 are correlated with each

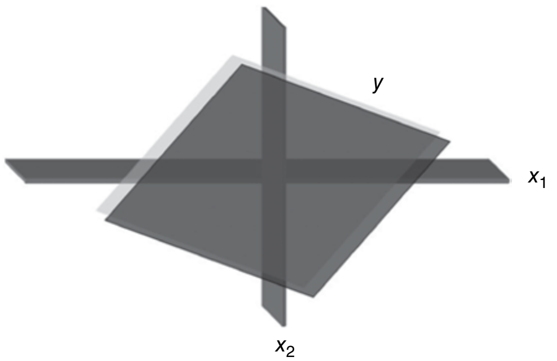


Figure 12.1 When the predictors x_1 and x_2 are uncorrelated, the response surface y rests on a solid basis, providing stable coefficient estimates.

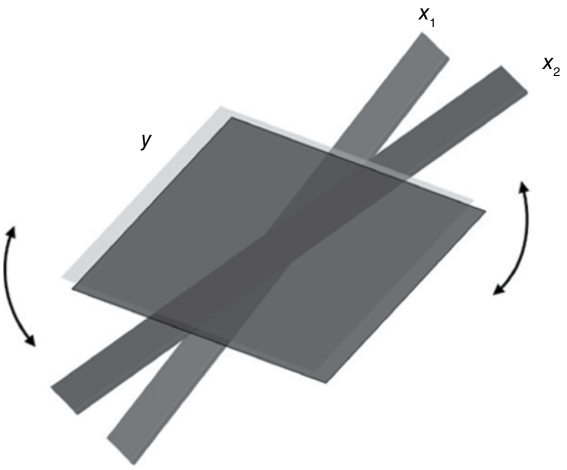


Figure 12.2 Multicollinearity: when the predictors are correlated, the response surface is unstable, resulting in dubious and highly variable coefficient estimates.

other, so that as one of them increases, so does the other. In this case, the predictors no longer form a solid basis upon which the response surface may firmly rest. Instead, when the predictors are correlated, the response surface is unstable, providing highly variable coefficient estimates b_1 and b_2.

The high variability associated with the estimates means that *different samples may produce coefficient estimates with widely different values*. For example, one sample may produce a positive coefficient estimate for x_1, while a second sample may produce a negative coefficient estimate. This situation is unacceptable when the analytic task calls for an explanation of the relationship between the response and the predictors, individually.

Let us look at a toy example to help us understand the problem. Consider the tiny population in Table 12.1.

Clearly, x_1 and x_2 are correlated, with $r = 0.938$, p-value about 0.

Now, split into two samples, as shown in Tables 12.2 and 12.3.

The regression equation for sample 1 is:

$$\hat{y} = -0.542 + 2.552\,x_1 - \mathbf{0.206}\,x_2$$

The regression equation for sample 2 is:

$$\hat{y} = -1.08 + 0.547\,x_1 \mathbf{+ 1.759}\,x_2$$

Note that the coefficient for x_2 is negative in sample 1 and positive in sample 2. This represents unstable behavior in the regression coefficients, caused by the correlation between the predictors. Essentially, *we cannot trust the values, or even the signs, of the regression coefficients*. So, interpreting the regression coefficients for our clients

TABLE 12.1 Tiny population for our toy example

Population		
x_1	x_2	**Target,** y
1	1	2.0693
1	2	2.6392
2	2	3.7501
2	3	5.6432
3	3	5.8925
3	4	6.4308
4	4	8.3950
4	5	8.4947
5	5	11.3236
5	5	10.1562

TABLE 12.2 Sample 1 from our tiny population

Sample 1		
x_1	x_2	**Target,** y
1	1	2.0693
2	2	3.7501
3	4	6.4308
4	5	8.4947
5	5	11.3236

TABLE 12.3 Sample 2 from our tiny population

Sample 2		
x_1	x_2	**Target,** y
1	2	2.6392
2	3	5.6432
3	3	5.8925
4	4	8.3950
5	5	10.1562

is probably not a good idea. The values and signs might change from sample to sample, so that interpreting the coefficients from just one sample may cost your client big bucks. This is why we need methods for dealing with this multicollinearity.

We shall use a subset of the *Cereals* data set, where we estimate the nutrition rating of breakfast cereals, using fiber, potassium, and sugar. Our regression equation is as follows:

$$rating = b_0 + b_1 \cdot fiber + b_2 \cdot potassium + b_3 \cdot sugars$$

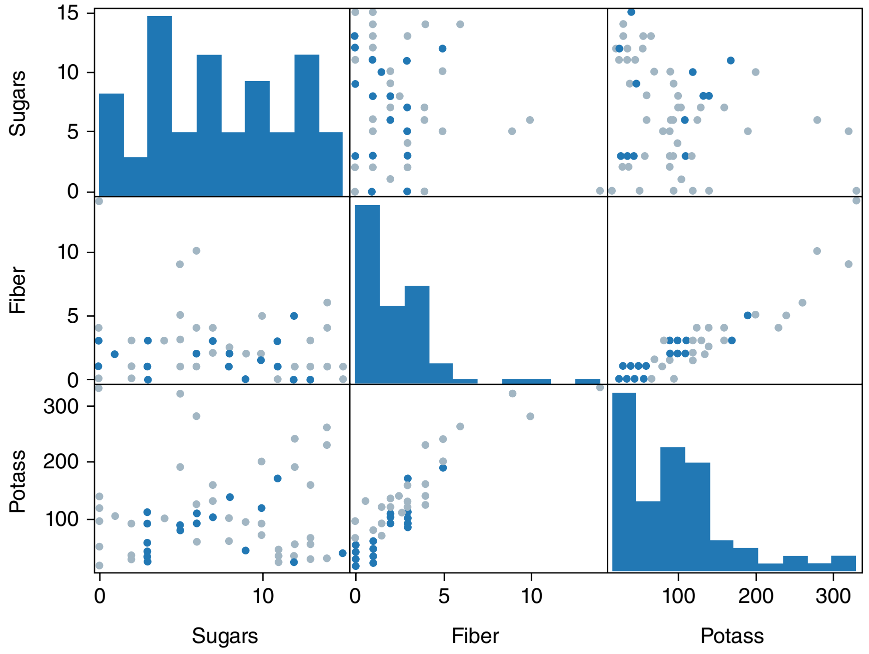


Figure 12.3 Matrix plot of the predictors from Python. Fiber and potassium are correlated.

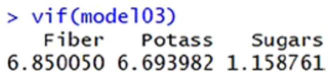
To be aware of possible multicollinearity, the analyst should investigate the correlation structure among the predictor variables. Figure 12.3 provides the matrix plot of the predictors. Clearly, *potassium and fiber are positively correlated*. In fact, their correlation coefficient is $r = 0.912$ (not shown). This strong correlation will inflate the variability of the regression coefficients, making our regression model unstable.

12.3 IDENTIFYING MULTICOLLINEARITY USING VARIANCE INFLATION FACTORS

However, suppose we did not check for the presence of correlation among our predictors, and went ahead and performed the regression anyway. Is there some way that the regression results can warn us of the presence of multicollinearity? The answer is yes: We may ask for the *variance inflation factors* (VIFs) to be reported.

The VIF for the ith predictor is given by:

$$VIF_i = \frac{1}{1 - R_i^2}$$



```
> vif(model03)
   Fiber   Potass   Sugars
6.850050 6.693982 1.158761
```

Figure 12.4 Regression results, with variance inflation factors from R indicating a multicollinearity problem.

where R_i^2 represents the R^2 value obtained by regressing x_i on the other predictor variables. Note that R_i^2 will be large when x_i is highly correlated with the other predictors, thus making VIF_i large.

A rough rule of thumb for interpreting the value of the VIF is to consider $VIF_i \geq 5$ to be an indicator of moderate multicollinearity, and to consider $VIF_i \geq 10$ to be an indicator of severe multicollinearity. A VIF of five corresponds to $R_i^2 = 0.80$, while $VIF_i = 10$ corresponds to $R_i^2 = 0.80$.

For the regression of *nutritional rating* on *fiber*, *potassium*, and *sugars*, we have the output provided in Figure 12.4. The VIF for fiber is 6.85 and the VIF for potassium is 6.69, with both values indicating moderate-to-strong multicollinearity.

12.3.1 How to Identify Multicollinearity Using Python

First, we load the required packages and read in the *cereals* data set under the name *cereals*.

```
import pandas as pd
import statsmodels.api as sm
import statsmodels.stats.outliers_influence as inf
cereals = pd.read_csv("C:/.../cereals.csv")
```

Once the data set is in Python, pull out the three predictor variables and put them in their own data frame. Call the data frame **X**.

```
X = pd.DataFrame(cereals[['Sugars', 'Fiber', 'Potass']])
```

Now that we have the predictor variables all together, use the **scatter_matrix()** command with **X** as the input to create a scatterplot matrix.

```
pd.plotting.scatter_matrix(X)
```

The result of the **scatter_matrix()** command is shown in Figure 12.3 above. Notice that the command creates both scatterplots and histograms.

To obtain the VIF values, we need to do a little data cleaning first. Use the **dropna()** command on the X data frame to remove any records with missing values.

```
X = X.dropna()
```

Then, make sure you add the constant term in the X data frame.

```
X = sm.add_constant(X)
```

Finally, run the **variance_inflation_factor()** command as given below to obtain the VIF values for all four columns in the X data frame.

```
[inf.variance_inflation_factor(X.values, i) for i in
range(X.shape[1])]
```

The output will include a VIF value for the constant term we added; ignore it. The VIF values of interest are for the three predictor variables, which are the second, third, and fourth numbers output by the **variance_inflation_factor()** command.

12.3.2 How to Identify Multicollinearity in R

To build a scatterplot matrix in R, we first need to identify which columns in the data set hold our predictor variables.

```
names(cereals)
```

You can see that the Sugars, Potass, and Fiber variables are in columns 10, 8, and 11, respectively. These are the columns you will put in the scatterplot matrix in Figure 12.5.

```
pairs(x = cereals[, c(10, 8, 11)], pch = 16)
```

The command for the scatterplot matrix is **pairs()**. The required input is **x**, which asks for the columns to include in the scatterplot matrix. We specify the columns using **cereals[, c(10, 8, 11)]**. An optional input **pch = 16** will change the scatterplot points from open circles to closed circles.

To calculate VIFs, we first need to install and open the *car* package.

```
install.packages("car"); library(car)
```

Once the *car* package is open, we build the model whose coefficients we want to examine for multicollinearity and save the model output.

```
model03 <- lm(formula = Rating ~ Fiber + Potass +
Sugars, data = cereals)
```

Finally, we use the **vif()** command on the model.

```
vif(model03)
```

The sole required input to the **vif()** command is the name we saved our model as. The output, shown in Figure 12.4, is the VIF for each predictor variable.

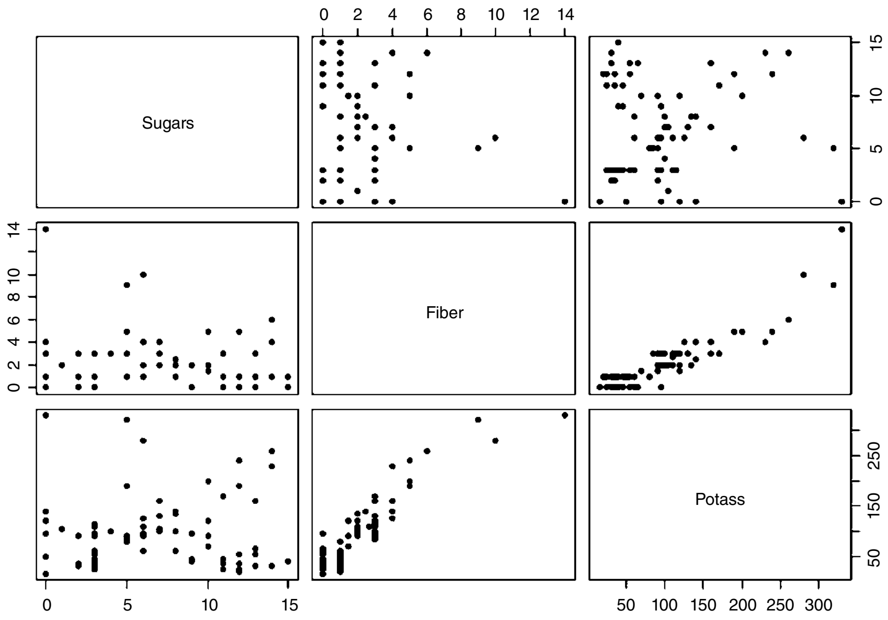


Figure 12.5 Scatterplot matrix of the predictors in R.

12.4 PRINCIPAL COMPONENTS ANALYSIS

So, now that we have identified the multicollinearity among our predictors, what do we do now?

One solution is to apply *principal components analysis*. Principal components analysis (PCA) seeks to account for the correlation structure of a set of predictor variables, using a smaller set of uncorrelated linear combinations of these variables, called *components*. The total variability produced by the complete set of m predictors can often be mostly accounted for by a smaller set of $k<m$ components. This means that there is almost as much information in the k components as there is in the original m variables. In addition, the k components are uncorrelated with each other, unlike the original correlated predictors. If desired, the analyst can then replace the original m variables with the $k < m$ components, so that the working data set now consists of n records on k components, rather than n records on m predictors. This is dimension reduction!

The analyst should note that PCA acts solely on the predictor variables and ignores the target variable. Also, the predictors should be either standardized or normalized. Mathematically, the principal components are uncorrelated linear combinations Y_i of predictors, with the following characteristics:

- The first principal component is usually the most important. It accounts for greater variability among the predictors than any other component.
- The second principal component accounts for the second-most variability and is uncorrelated with the first.
- The third principal component accounts for the third-most variability and is uncorrelated with the first two, and so on.

12.5 AN APPLICATION OF PRINCIPAL COMPONENTS ANALYSIS

To illustrate the application of PCA, we turn to the *clothing_store_PCA_training* and *clothing_store_PCA_test* data sets. We are interested in estimating the response *Sales per Visit* using the predictors *Purchase Visits, Days on File, Days between Purchases, Different Items Purchased*, and *Days since Purchase*. However, Figure 12.6 shows that there is substantial correlation among the predictors. In addition, Figure 12.7, showing the regression of *Sales per Visit* versus the predictors, indicates some moderately inflated VIF metrics.

We therefore perform PCA on these predictors, using *varimax rotation* on the training data set.

Rotating the PCA solution helps in the interpretability of the components. Examining the rotated components in Figure 12.8, we find that, if we extract only the first principal component, we will account for only 31.3% of the variance among the predictors. If we extract two components, we account for about 52.2% (see *Cumulative Var* in Figure 12.8), and so on. So, the question arises, how many components should we extract?

```
                          Days.since.Purchase.Z Purchase.Visits.Z Days.on.File.Z
Days.since.Purchase.Z                     1.000            -0.440         -0.159
Purchase.Visits.Z                        -0.440             1.000          0.364
Days.on.File.Z                           -0.159             0.364          1.000
Days.between.Purchases.Z                  0.573            -0.453          0.203
Diff.Items.Purchased.Z                   -0.379             0.821          0.303
                          Days.between.Purchases.Z Diff.Items.Purchased.Z
Days.since.Purchase.Z                        0.573                 -0.379
Purchase.Visits.Z                           -0.453                  0.821
Days.on.File.Z                               0.203                  0.303
Days.between.Purchases.Z                     1.000                 -0.371
Diff.Items.Purchased.Z                      -0.371                  1.000
```

Figure 12.6 Correlation matrix from R shows substantial correlation among the predictors.

```
   Days.since.Purchase.Z        Purchase.Visits.Z           Days.on.File.Z
                1.701947                 3.793173                 1.536395
Days.between.Purchases.Z   Diff.Items.Purchased.Z
                2.145807                 3.076706
```

Figure 12.7 Regression from R shows some tending toward moderately large VIFs.

```
Loadings:
                         RC1    RC2    RC3    RC4    RC5
Days.since.Purchase.Z                  0.935
Purchase.Visits.Z        0.725                       0.573
Days.on.File.Z                  0.971
Days.between.Purchases.Z                      0.910
Diff.Items.Purchased.Z   0.965

                 RC1   RC2   RC3   RC4   RC5
SS loadings    1.566 1.045 1.035 1.006 0.348
Proportion Var 0.313 0.209 0.207 0.201 0.070
Cumulative Var 0.313 0.522 0.729 0.930 1.000
```

Figure 12.8 Excerpt from PCA results from R.

12.6 HOW MANY COMPONENTS SHOULD WE EXTRACT?

Recall that one of the motivations for PCA was to reduce the dimensionality. The question arises, “How do we determine how many components to extract?” For example, should we retain only the first two principal components, since they explain over half (52% Cumulative Var) of the total variability? Or should we retain all five components, since they explain 100% of the variability? Well, clearly, retaining all five components does not help us to reduce the dimensionality. As usual, the answer lies somewhere between these two extremes.

12.6.1 The Eigenvalue Criterion

In Figure 12.8, the eigenvalues are labeled as “SS loadings.” An eigenvalue of 1.0 would mean that the component would explain about “one predictor’s worth” of the total variability. The rationale for using the eigenvalue criterion is that each

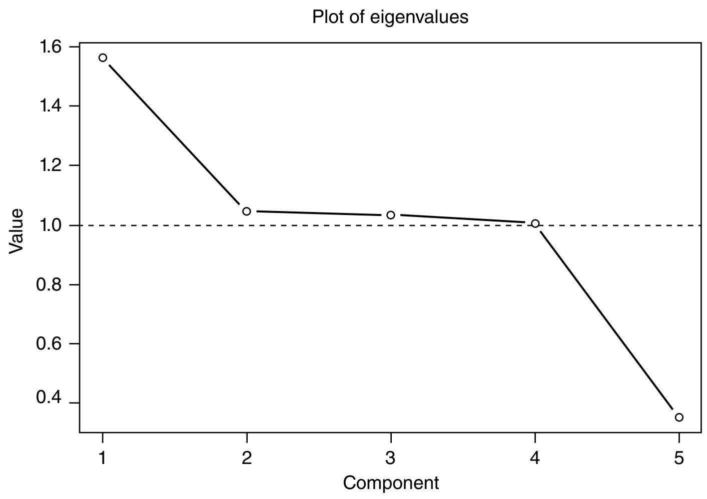


Figure 12.9 Plot of eigenvalues from R, with dotted line at eigenvalue of 1.

component should explain at least one predictor's worth of the variability, and therefore, the eigenvalue criterion states that only components with eigenvalues greater than one should be retained. Note that, if there are fewer than 20 predictors, the eigenvalue criterion tends to recommend extracting too few components, while, if there are more than 50 variables, this criterion may recommend extracting too many.

From Figure 12.9, we see that four of the rotated components have eigenvalues greater than 1, and are therefore retained. Component 5 has a rotated eigenvalue much below 1.0, so we do not include Component 5. The eigenvalue criterion therefore suggests that we extract $k = 4$ principal components.

12.6.2 The Proportion of Variance Explained Criterion

For the proportion of variance explained criterion, the client or analyst first specifies what proportion of the total variability that he or she would like the principal components to account for. Then, the analyst simply selects the components one by one until the desired proportion of variability explained is attained. For example, suppose we would like our components to explain about 70% of the variability in the predictors. Then, from Figure 12.8, we would choose components 1–3, which together explain 72.9% of the variability. On the other hand, if we wanted our components to explain 90% of the variability, then we would need to include component 4 as well, which together with the first three components would explain 93% of the variability. Without further input from the client, we might just say that the proportion of variance explained criterion suggests we go with either $k = 3$ or $k = 4$ components.

Since the eigenvalue criterion suggested $k = 4$ components, and the proportion of variance explained criterion is fine with either $k = 3$ or $k = 4$ components, we therefore by consensus settle on extracting $k = 4$ components.

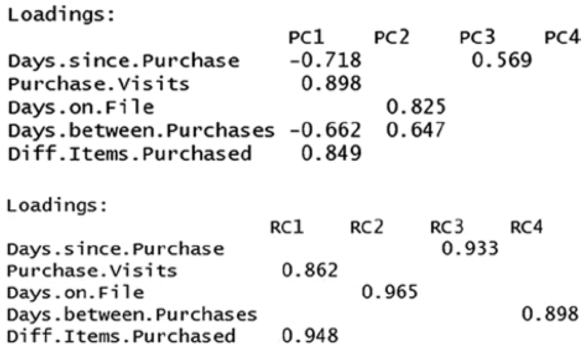
12.7 PERFORMING PCA WITH *K* = 4

Figure 12.10 shows the resulting (i) unrotated and (ii) rotated component matrices for extracting three components. Let us examine the *rotated* matrix first in Figure 12.10b. Note that the component weights less than 0.5 have been suppressed, to enhance interpretability. The first principal component (*RC1* for Rotated Component 1) is a combination of *Different Items Purchased* and *Purchase Visits*, which are positively correlated with each other, since their component weights have the same sign. Components can contain combinations of predictors that are either positively or negatively correlated with each other. Had exactly one of the component weights been negative, then that would have been an indication that *Different Items Purchased* and *Purchase Visits* were negatively correlated. The remaining principal components are "singletons," containing only a single predictor each.

Now, suppose we had not rotated the component matrix, ending up with the unrotated component matrix in Figure 12.10a. Note that the interpretations of the principal components are less clear. Component 1 is huge, containing four of the five predictors, with both positive and negative correlations mixed in. Much cleaner is the interpretation of the rotated component matrix.

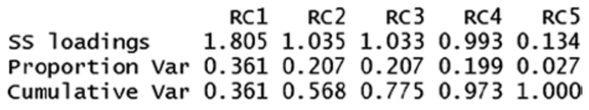
12.8 VALIDATION OF THE PRINCIPAL COMPONENTS

As with any other data science method, the results of the PCA should be validated, using the test data set. Figure 12.11 shows the proportions of variance explained by all five components, with percentages not much different from the training set



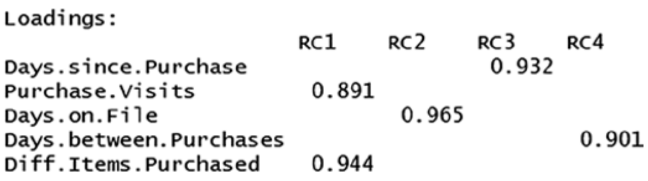
```
Loadings:
                        PC1    PC2    PC3    PC4
Days.since.Purchase    -0.718         0.569
Purchase.Visits         0.898
Days.on.File                   0.825
Days.between.Purchases -0.662  0.647
Diff.Items.Purchased    0.849

Loadings:
                       RC1    RC2    RC3    RC4
Days.since.Purchase                  0.933
Purchase.Visits        0.862
Days.on.File                  0.965
Days.between.Purchases                      0.898
Diff.Items.Purchased   0.948
```

figure 12.10 (a) component weights with no rotation, from r. (b) component weights with varimax rotation, from r.



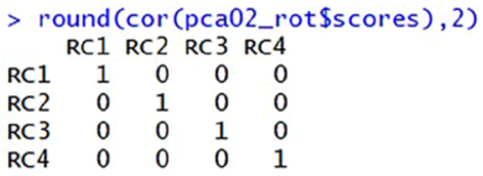
```
                 RC1   RC2   RC3   RC4   RC5
SS loadings    1.805 1.035 1.033 0.993 0.134
Proportion Var 0.361 0.207 0.207 0.199 0.027
Cumulative Var 0.361 0.568 0.775 0.973 1.000
```

Figure 12.11 Proportions of variance explained from R for the test data set.



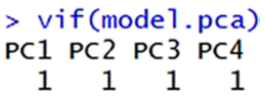
```
Loadings:
                          RC1    RC2    RC3    RC4
Days.since.Purchase                      0.932
Purchase.Visits            0.891
Days.on.File                      0.965
Days.between.Purchases                          0.901
Diff.Items.Purchased       0.944
```

Figure 12.12 Component weights from R for the test data set.



```
> round(cor(pca02_rot$scores),2)
    RC1 RC2 RC3 RC4
RC1   1   0   0   0
RC2   0   1   0   0
RC3   0   0   1   0
RC4   0   0   0   1
```

Figure 12.13 R output showing the principal components are uncorrelated.



```
> vif(model.pca)
PC1 PC2 PC3 PC4
  1   1   1   1
```

Figure 12.14 R output showing that regression using the principal components eliminates multicollinearity.

results in Figure 12.8. The four rotated components for the test set, shown in Figure 12.12, are similar to those for the training set from Figure 12.10b.

So, did PCA alleviate our multicollinearity problem? We can check by examining

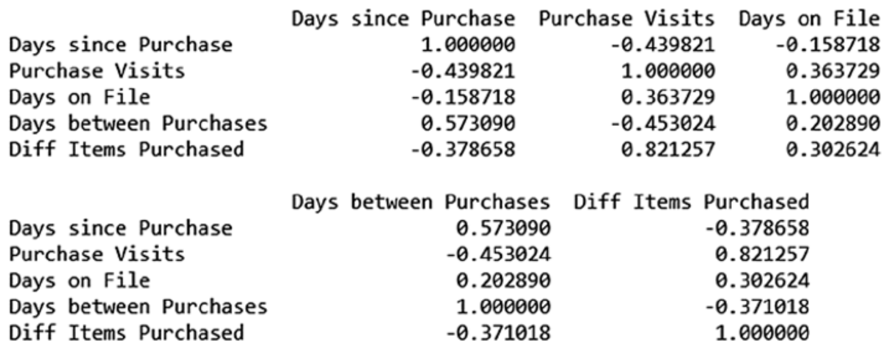
1. The correlations among the four components.
2. The predictor VIF for the regression of the response on the components.

The correlation matrix for the principal components is shown in Figure 12.13. All correlations are zero, meaning that the components are uncorrelated. Finally, we obtain the VIFs for the regression of *Sales per Visit* on the four extracted principal components substituted for the original predictors. These VIFs results shown in Figure 12.14 indicate that all VIFs equal 1, the minimum.

12.9 HOW TO PERFORM PRINCIPAL COMPONENTS ANALYSIS USING PYTHON

Load the required packages.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
```



```
                       Days since Purchase  Purchase Visits  Days on File
Days since Purchase               1.000000        -0.439821     -0.158718
Purchase Visits                  -0.439821         1.000000      0.363729
Days on File                     -0.158718         0.363729      1.000000
Days between Purchases            0.573090        -0.453024      0.202890
Diff Items Purchased             -0.378658         0.821257      0.302624

                       Days between Purchases  Diff Items Purchased
Days since Purchase                  0.573090             -0.378658
Purchase Visits                     -0.453024              0.821257
Days on File                         0.202890              0.302624
Days between Purchases               1.000000             -0.371018
Diff Items Purchased                -0.371018              1.000000
```

Figure 12.15 Correlation matrix from Python.

Read in the two data sets, *clothing_store_PCA_training* and *clothing_store_PCA_test*, as *clothes_train* and *clothes_test*.

```
clothes_train = pd.read_csv("C:/.../clothing_store_PCA_training")
clothes_test = pd.read_csv("C:/.../clothing_store_PCA_test")
```

Separate the predictor variables from the rest of the training data set using the **drop()** command. Note that we drop the target variable *Sales per Visit*, so we are left with only the predictor variables. This approach is best suited for when the target variable and predictor variables of interest are the only variables in your data. Save the variables as **X**.

```
X = clothes_train.drop('Sales per Visit', 1)
```

Obtain the correlation matrix of the X variables by using the **corr()** command.

```
X.corr()
```

The output from the **corr()** command is shown in Figure 12.15.

Now, we will run PCA with five components. First we will specify the number of components using **n_components** in the **PCA()** command, then fit the PCA to the data using **fit_transform()** with **X** as the input.

```
pca01 = PCA(n_components=5)
principComp = pca01.fit_transform(X)
```

Once PCA is run, our next task is to look at the amount of variability explained by each component and the corresponding cumulative variability explained. Use the **pca01** object and **explained_variance_ratio_** to obtain the variability explained by each component.

```
pca01.explained_variance_ratio_
```

Obtain the cumulative variability explained by running the **cumsum()** command on **pca01.explained_variance_ratio_**.

```
np.cumsum(pca01.explained_variance_ratio_)
```

Our results in this section are for the original, unrotated components. As of this writing, Python's *sklearn* package does not have an "out of the box" command to perform varimax rotation on the components.

12.10 HOW TO PERFORM PRINCIPAL COMPONENTS ANALYSIS USING R

Import the *clothing_store_PCA_training* and *clothing_store_PCA_test* data sets as *clothes_train* and *clothes_test*, respectively. To simplify code that comes later, we will separate the training and test data into X and y variables.

```
y <- clothes_train$Sales.per.Visit
X <- clothes_train[, c(1:5)]
X_test <- clothes_test[, c(1:5)]
```

Remember to standardize the predictor variables.

```
X_z <- as.data.frame(scale(X))
colnames(X_z) <- c("Days.since.Purchase.Z", "Purchase.
Visits.Z", "Days.on.File.Z",
     "Days.between.Purchases.Z", "Diff.Items.Purchased.Z")
```

To obtain the correlation matrix, we use the **cor()** command.

```
round(cor(X_z), 3)
```

The **cor()** command, with the predictor variables **X_z** as input, is placed inside the **round()** command. The second input of the **round()** command is the number of significant digits the answers will be rounded to. In our case, we have specified three significant digits. The output from the **round()** command is shown in Figure 12.6.

To obtain the VIF values, we need to run the regression model first, then use the **vif()** command from the *car* package as detailed in the previous R section.

```
model01 <- lm(formula = y ~ Days.since.Purchase.Z +
Purchase.Visits.Z + Days.on.File.Z +
      Days.between.Purchases.Z + Diff.Items.Purchased.Z,
      data = X_z)
vif(model01)
```

The output of the **vif()** command is shown in Figure 12.7.

To run PCA, we must first install and open the *psych* package.

```
install.packages("psych"); library(psych)
```

Once the package is open, we use the **principal()** command to perform PCA.

```
pca01 <- principal(r = X_z, rotate = "varimax", nfactors = 5)
```

We are using the **principal()** command with three input values. The input **r = X_z** specifies the variables we want to analyze. The **rotate = "varimax"** input tells R to perform varimax rotation on the components before presenting the results. Finally, the **nfactors = 5** input states that we want five components. We save the PCA output as **pca01**.

We are interested in the **loadings** of the **pca01** results.

```
print(pca01$loadings, cutoff = 0.49)
```

We use cutoff = 0.49 to suppress small PCA weights. The output generated by the code above is shown in Figure 12.8.

To create a plot of the eigenvalues, we need the eigenvalues of the rotated components. These are located under "SS Loadings" in Figure 12.8. Once you save these values as their own vector, plot them using the **plot()** command.

```
ss.load <- c(1.566, 1.045, 1.035, 1.006, 0.348)
plot(ss.load, type = "b", main = "Plot of Eigenvalues",
ylab = "Value",
        xlab = "Component"); abline(h = 1, lty =2)
```

The input **type = "b"** plots the eigenvalues using both points and connecting lines. The **main**, **xlab**, and **ylab** input customize the title and axes labels. The **abline()** command adds a line to the plot. The input value **h = 1** adds the line with horizontal location 1 and the input **lty = 2** specifies a dashed line. The result is shown in Figure 12.9.

To compare the results of no rotation versus varimax rotation, run the **principal()** command using **rotate = "none"** for no rotation and **rotate = "varimax"** for the varimax rotation.

```
pca02_norot <- principal(r = X, rotate = "none",
nfactors = 4)
print(pca02_norot$loadings, cutoff = 0.5)
pca02_rot <- principal(r = X, rotate = "varimax",
nfactors = 4)
print(pca02_rot$loadings, cutoff = 0.5)
```

An excerpt from the **print()** output of **pca02_norot$loadings** is shown in Figure 12.10a, while an excerpt from the **print()** output of **pca02_rot$loadings** is shown in Figure 12.10b. Note that both commands use **cutoff = 0.5** to suppress weights below 0.5.

To validate the PCA results, run the algorithm on the test data. First, standardize the data.

```
X_test_z <- scale(X_test)
```

Then, confirm that four components are recommended.

```
pca02_test <- principal(r = X_test_z, rotate =
"varimax", nfactors = 5)
pca02_test$loadings
```

An excerpt of the output is shown in Figure 12.11. After confirming that four components are recommended, examine the component weights.

```
pca02_test <- principal(r = X_test_z, rotate =
"varimax", nfactors = 4)
print(pca02_test$loadings, cutoff = 0.5)
```

An excerpt of the output is shown in Figure 12.12.

To obtain the correlation of the components in the training data set, run the **round(cor())** command on the scores from **pca02_rot**.

```
round(cor(pca02_rot$scores),2)
```

In this case, we round to two significant digits. The result is shown in Figure 12.13.

To regress the target variable on the components, you may want to save each component as its own variable. This is shown below.

```
PC1 <- pca02_rot$scores[,1]; PC2 <- pca02_rot$scores[,2]
PC3 <- pca02_rot$scores[,3]; PC4 <- pca02_rot$scores[,4]
```

Now you can run the regression model and obtain the VIFs. The code is shown below.

```
model.pca <- lm(y ~ PC1 + PC2 + PC3 + PC4); vif(model.pca)
```

The output from the **vif()** command is shown in Figure 12.14.

12.11 WHEN IS MULTICOLLINEARITY NOT A PROBLEM?

Now, depending on the task confronting the analyst, multicollinearity may not in fact present a fatal defect. Weiss[1] notes that multicollinearity "does not adversely affect the ability of the sample regression equation to predict the response variable." He adds that multicollinearity does not significantly affect point estimates of the target variable, confidence intervals for the mean response value, or prediction

[1]Weiss, *Introductory Statistics*, Ninth Edition, Pearson, London, 2010.

intervals for a randomly selected response value. However, the data scientist must therefore strictly limit the use of a multicollinear model to estimation and prediction of the target variable. Interpretation of the model would *not* be appropriate, since the individual coefficients may not make sense, in the presence of multicollinearity. To summarize, *models not accounting for multicollinearity may be used for estimation, but not for description or interpretation.*

REFERENCES

The name of the *car* package sands for Companion to Applied Regression, and the details can be found here: John Fox and Sanford Weisberg, *An {R} Companion to Applied Regression*, Second Edition, Sage, Thousand Oaks, CA, 2011.

The psych package, which let us to PCA with varimax rotation, is documented here: W. Revelle, *Psych: Procedures for Personality and Psychological Research*, Northwestern University, Evanston, IL, 2018.

EXERCISES

CLARIFYING THE CONCEPTS

1. What do we mean by high dimensionality in data science?
2. Why do we need dimension reduction methods?
3. What does principal components replace the original set of *m* predictors with?
4. Which principal component accounts for the most variability?
5. Which of the other principal components is correlated with the first principal component?
6. Why do we use rotation?
7. Explain the eigenvalue criterion?
8. What is the proportion of variance explained criterion?
9. True or false: It is not necessary to perform validation of the principal components.
10. When we use the principal components as predictors in a regression model, what value do the VIFs take? What does this indicate?

WORKING WITH THE DATA

For the following exercises, work with the *clothing_store_PCA_training* and *clothing_store_PCA_test* data sets. Use either Python or R to solve each problem.

11. Standardize or normalize the predictors.
12. Construct the correlation matrix for the predictor variables Purchase Visits, Days on File, Days between Purchases, Different Items Purchased, and Days since Purchase. Which variables are highly correlated?

13. Calculate the VIFs for each of the predictor variables. Which predictor variable VIFs indicate the multicollinearity is a problem?

14. Run PCA using varimax rotation and five components. What percent of the variability is explained by one component? By two components? By all five components?

15. Make a plot of the eigenvalues. Using the eigenvalue criterion, how many components would you retain?

16. Say we want to explain at least 80% of the variability. How many components would you retain?

17. Run PCA using varimax rotation and four components. What percent of the variability do the four components explain?

18. What variable or variables are contained in each of the components?

19. Use the four components as the predictor variables in a regression model to estimate Sales per Visit. What are the regression coefficients of the four components?

20. What are the VIFs of the four components in the regression model?

HANDS-ON ANALYSIS

For the following exercises, work with the *cereals* data set. Use either Python or R to solve each problem.

21. Standardize or normalize the predictors Sugars, Fiber, and Potass.

22. Construct the correlation matrix for Sugars, Fiber, and Potass. Which variables are highly correlated?

23. Build a regression model to estimate Rating based on Sugars, Fiber, and Potass. Obtain the VIFs from the model. Which VIFs indicate that multicollinearity is a problem?

24. Run PCA using varimax rotation and three components. What percent of the variability is explained by one component? By two components? By all three components?

25. Make a plot of the eigenvalues of the three components. Using the eigenvalue criterion, how many components would you retain?

26. Say we want to explain at least 70% of the variability. How many components would you retain?

27. Run PCA using varimax rotation and two components. What percent of the variability do the two components explain?

28. What variable or variables are contained in Component 1? What variable or variables are contained in Component 2?

29. Use the two components as the predictor variables in a regression model to estimate Rating. What are the regression coefficients of the two components?

30. What are the VIFs of the two components in the regression model?

For the following exercises, work with the *red_wine_PCA_training* and *red_wine_PCA_test* data sets. Use either Python or R to solve each problem. The target variable is the wine *quality*. The predictors are *alcohol, residual sugar, pH, density*, and *fixed acidity*.

31. Standardize or normalize the predictors.

32. Construct the correlation matrix for the predictors. Between which predictors do you find the highest correlations?

33. Build a regression model to estimate *quality* based on the predictors. Obtain the VIFs from the model. Which VIFs indicate that multicollinearity is a problem? Compare the variables with high VIF to the correlated variables from the previous exercise.

34. Perform PCA using varimax rotation. Show the rotated proportions of variance explained for extracting up to five components. What percent of the variability is explained by one component? By two components? By three components? By four components? By all five components?

35. Say we want to explain at least 90% of the variability. How many components does the proportion of variance explained criterion suggest we extract?

36. Make a plot of the eigenvalues of the five components. According to the eigenvalue criterion, how many components should we extract?

37. Combine the recommendations from the two criteria to reach a consensus as to how many components we should extract.

38. Profile each of your components, stating which variables are included, and noting their within-component correlation (positive or negative). For simplicity, consider components weights greater than 0.5 only.

39. Produce the correlation matrix for the components. What do these values mean?

40. Next, use only the components you extracted to estimate wine quality using a regression model. Do not include the original predictors.

   a. Compare the values of s and R^2_{adj} between the PCA regression and the original regression model.

   b. Explain why the original model slightly outperformed the PCA model.

   c. Explain how the PCA model may be considered superior, even though slightly outperformed?

41. In your regression from the previous exercise, what are the VIFs of the two components in the regression model? What do these values mean?