

# An Introduction to Principal Component Analysis with Examples in R

Thomas Phan  
first.last @ acm.org  
Technical Report\*  
September 1, 2016

## 1 Introduction

Principal component analysis (PCA) is a series of mathematical steps for reducing the dimensionality of data. In practical terms, it can be used to reduce the number of features in a data set by a large factor (for example, from 1000s of features to 10s of features) if the features are correlated.

This type of “feature compression” is often used for two purposes. First, if high-dimensional data is to be visualized by plotting it on a 2-D surface (such as a computer monitor or a piece of paper), then PCA can be used to reduce the data to 2-D or 3-D; in this context, PCA can be considered a complete, standalone unsupervised machine learning algorithm. Second, if a different machine learning training algorithm is taking too long to run, then PCA can be used to reduce the number of features, which in turn reduces the amount of training data and the time to train a model; here, PCA is used as a pre-processing step as part of a larger workflow. In this paper we discuss PCA largely for the first purpose of visualizing and exploring patterns in data.

It is important to note that PCA does not reduce features by selecting a subset of the original features (such as what is done with wrapper feature selection algorithms that perform feature-by-feature forward or backward search [6]). *Instead, PCA creates new, uncorrelated features that are a linear combination of the original features.* For a given data instance, its features are transformed via a dot product with a numeric vector to create a new feature; this vector is a *principal component* that serves as the direction of an axis upon which the data instance is projected. The new features are thus the projections of the original features into a new coordinate space defined by the principal components. To perform the actual dimensionality reduction, the user can follow a well-defined methodology to select the fewest new features that

explain a desired amount of data variance.

This paper is organized in the following manner. In Section 2 we explain how PCA is applied to data sets and how it creates new features from existing features. Importantly, we explain various tips for how to effectively use PCA with the R programming language in order to achieve good feature compression. In Section 3 we use PCA to explore three different data sets: Fisher’s Iris data, Kobe Bryant’s shots, and car class fuel economy. In Section 4 we show R code examples that run PCA on data sets, and in Section 5 we provide references for further reading. We conclude the paper in Section 6.

## 2 Principal Component Analysis

### 2.1 Applying PCA

Figure 1 illustrates the effect of applying PCA. Consider the four blue dots in the left of the figure; each dot may represent an original data instance with three features, and so they can be placed in 3-D space. Now, suppose that the data instances are to be reduced in dimensionality down to 2 features (in 2-D space) or even 1 feature (in 1-D space). PCA performs this compression by *projecting* the dots onto these lower-dimensional subspaces. Intuitively, one can imagine a flat 2-D plane that is placed within the 3-D space with the dots dropping directly onto that plane as shown on the upper-right of the figure. This 2-D subspace is spanned by orthogonal (perpendicular) vectors, called the first principal component (PC1) and the second principal component (PC2). Additionally, the original data instances can be compressed in dimensionality even further by projecting the points down onto a 1-D line defined by PC1.

Each principal component is simply a vector of floating-point numbers that defines an axis in the reduced feature space. For an original data set that has  $D$  dimensions (features), PCA software will generate  $D$  principal components (typically labelled PC1,

---

\*This document serves as a readable tutorial on PCA using only basic concepts from statistics and linear algebra.

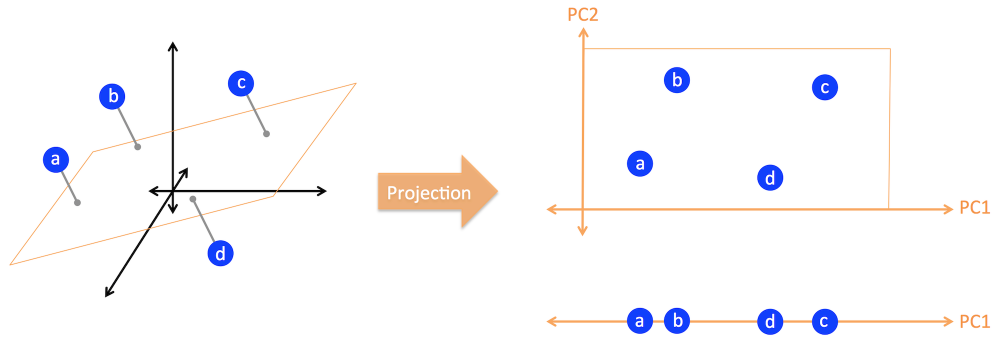


Figure 1: Effect of applying PCA to a data set. **Left:** The original data instances have 3 features and so are in 3-D space. **Top-right:** After applying PCA, the original data points can be reduced to 2 features by projecting them onto a 2-D plane. **Bottom-right:** The original data can be further reduced to 1 feature on a 1-D line.

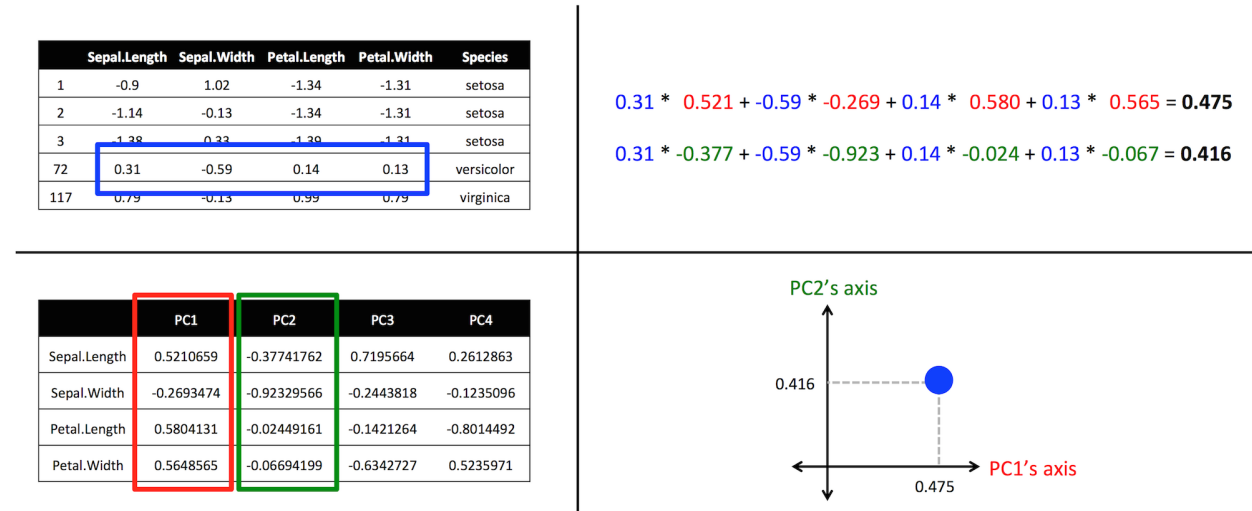


Figure 2: Calculating the projection of data instances onto lower-dimensional subspaces. **Top-left:** Original Iris data set after scaling. **Bottom-left:** Four principal components (PC1, PC2, PC3, and PC4) computed by PCA. **Top-right:** Dot product between a data instance (blue numbers) and the first two principal components, PC1 (red numbers) and PC2 (green numbers). **Bottom-right:** Projection of the data instance onto PC1 and PC2. The original 4-D data instance (0.31, -0.59, 0.14, 0.13) has effectively been reduced to a 2-D data instance (0.475, 0.416).

PC2, etc.), each of which is a vector with  $D$  numbers, where the vectors are orthogonal to each other in multi-dimensional space.

Consider the example in Figure 2, where some sample data instances from the Iris data set [3, 1] are shown in the table in the top-left. (Note that the data has been scaled and centered using R’s `scale()` function.) One specific data instance comprising the feature values (0.31, -0.59, 0.15, 0.13) is shown in the blue outline.

The table in the bottom-left shows the principal component vectors produced by the PCA software. Because the Iris data set has four dimensions (named `Sepal.Length`, `Sepal.Width`, `Petal.Length`, and `Petal.Width`), the PCA software produced four PCs each with four floating-point coefficient values. As we will describe later, the principal components (PC1, PC2, PC3, and PC4) are returned by the PCA software in a specific order of importance (from most to least variance of projected data). Several points are worth noting here. First, because the computed principal components are orthogonal, their pair-wise dot products will come out to be 0. Second, the PCA software generates the values in each principal component such that the sum of their squares is 1.0. (In linear algebra terminology, the principal components form an orthonormal basis.) Finally, in various literature, these values may be referred to as “loadings”.

Each principal component defines an axis, and a data instance can be projected onto the axis by computing the dot product (also known as the inner product) between the data instance and the principal component. For example, in the upper-right of the figure, we show the dot product between the data instance (in blue) and PC1 (in red). The resulting scalar number (0.475) is the projection of the data instance onto the axis defined by PC1. When we compute the dot product between the data instance and PC2 (in green), we get another scalar number (0.416), which is the projection of the data instance onto the axis defined by PC2. Because PC1 and PC2 are orthogonal to each other, the two projected values identify a coordinate (0.475, 0.416) in the 2-D subspace spanned by the PC1 and PC2 vectors, as shown in the bottom-right. The data instance has now effectively been reduced from four dimensions to two dimensions. If this same type of projection is carried out for all the Iris data instances, then they can all be plotted onto a 2-D graph, as will be shown in Section 3.1.

Another interpretation of this process is that the original features are transformed into new features. Example R code in Section 4 will demonstrate applying such a transformation on an entire data set.

It is important to understand why the new principal components must be orthogonal to each other. Collectively, these axes define a new coordinate system for the data instances, and because the intent of PCA is to reduce dimensionality, this new coordinate system should ideally represent the original data with as few axes as possible. PCA thus uses orthogonal axes to maximally capture the variability of the data. Consider again the bottom-right portion of Figure 2 that shows the orthogonal axes PC1 (horizontally) and PC2 (vertically). Suppose instead that PC1 and PC2 were both parallel horizontally; these parallel axes could represent the point’s position only along the horizontal axis but could not capture the position along the vertical axis. In the next three subsections, we examine in detail the concept of data variance as it relates to the principal components.

## 2.2 Computing the principal components

The goal of the PCA software is to compute the principal component vectors as was shown in Figure 2 in the lower-left table. Because a vector can define a line and an axis, each principal component thus defines an axis upon which the original data instances are projected.

Computing PC1 can be understood as an optimization problem, as illustrated in Figure 3. The leftmost sub-figure shows a sample of three blue dots placed in 2-D space. The first principal component PC1 can be any one out of an infinite number of lines that occur in this space. The PC1 optimization can then be seen as the problem of selecting the line that minimizes the total projection error (also known as the reconstruction error) that results when the original data instances are projected onto the line.

The other three sub-figures of Figure 3 show three candidate lines for PC1, where the rightmost sub-figure shows the solution. The small red lines represent the projection error: the orthogonal (perpendicular) distance between a data instance and its location on the line. PCA thus selects the line that *minimizes* the sum of these projection errors.

Importantly, finding such a line simultaneously solves a second, equivalent problem: finding the line that *maximizes* the projected data instances’ variance, as shown in the figures as the green segments. Recall that the variance<sup>1</sup> of a list  $X$  of  $n$  scalar values  $x_i$  is defined as  $\sigma^2 = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{X})^2$  and is a quantification of the spread of the numbers. When the original data points are projected onto the line, the variance can be computed over those projected

<sup>1</sup>This equation defines the population variance as opposed to the sample variance.

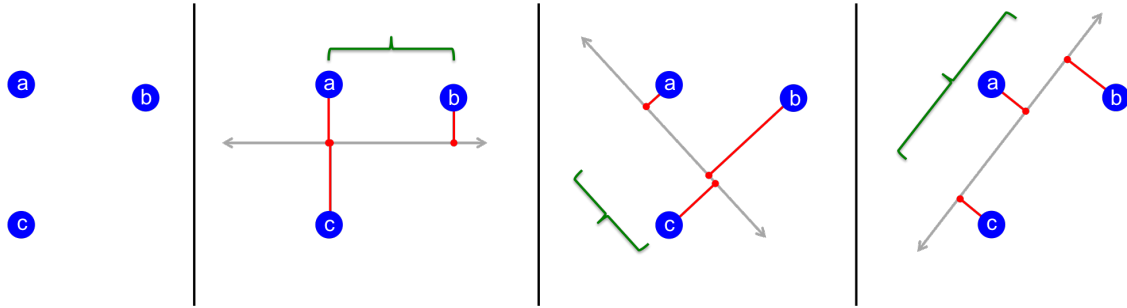


Figure 3: The selection of the first principal component as an optimization problem. The **leftmost** sub-figure shows the original data instances. The other three sub-figures show three axes (shown as grey double-ended arrows) that are candidates for the first principal component. The **red** lines indicate orthogonal (perpendicular) projection error, while the **green** segments indicate variance (spread) of the data projected onto the candidate axes. The **rightmost** sub-figure shows an axis that produces the smallest total projection error and the largest variance, and so this axis would be chosen to be the first principal component.

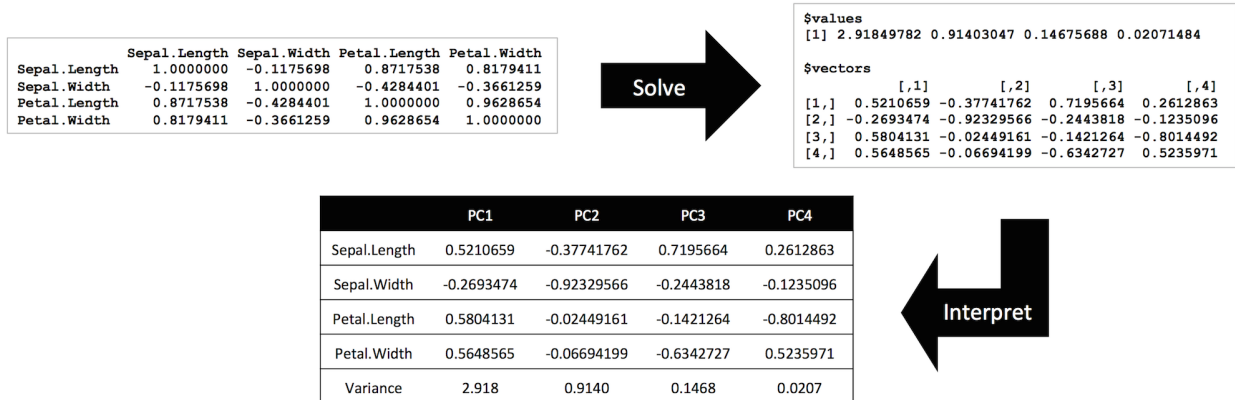


Figure 4: The calculation of the principal components as a linear algebra problem. **Top-left:**  $4 \times 4$  covariance matrix from the scaled and centered Iris data set. **Top-right:** Resulting 4 eigenvectors (in the columns of the **vectors** variable) and 4 eigenvalues (in the **values** variable). **Bottom:** Interpreting the eigenvectors as principal components and the eigenvalues as the variance of the data after projecting onto each principal component.

values. The PC1 line is thus computed such that this variance is maximized and (equivalently) the projection error is minimized.

Once PC1 is computed, it determines the first axis. PC2 (and all subsequent principal components) are then chosen by PCA to (1) be orthogonal to the previous principal components and (2) solve the variance maximization / projection error minimization problem discussed in the previous paragraph.

## 2.3 PCA solution via linear algebra

While variance maximization (or equivalently, projection error minimization) defines the geometric problem that PCA solves, the result produced by PCA software is found from a series of computations that involve linear algebra. These steps allow PCA to algorithmically compute the principal components that capture the most variance when data points are projected onto them. Later in Section 4, we will show R code that demonstrates both the explicit linear algebra steps as well as the PCA library function `prcomp()` that conveniently hides these details.

Figure 4 illustrates the needed sequence of steps. First, the values in the original data set are **scaled and centered** to prevent features with large numeric ranges from dominating other features. The implication is that data must be numeric; any categorical feature (known in R as a **factor**) must be converted to a corresponding dummy boolean integer feature.

Second, PCA computes the **covariance matrix** of the scaled and centered data, where the matrix elements contain the covariance between every pair of features. Note the following important facts about a covariance matrix. Because the same features are along both the rows and columns, the matrix is square. Additionally, because the covariance between two features  $X$  and  $Y$  is  $cov(X, Y) = cov(Y, X)$ , the matrix is symmetric. Further, because  $cov(X, X) = var(X)$ , the diagonal contains the variance of every feature.

Third, PCA computes the **eigenvectors** and **eigenvalues** of the covariance matrix. For a square matrix  $A$ , there can exist multiple pairs of corresponding eigenvector  $\vec{v}$  and scalar eigenvalue  $\lambda$  if  $A\vec{v} = \lambda\vec{v}$ . In PCA, we wish to have real-valued (as opposed to complex) eigenvectors, but not all matrices have such eigenvectors. However, a covariance matrix does have real-valued eigenvectors because square symmetric matrices are guaranteed to have them due to the Spectral Theorem. Computing all the eigenvectors and eigenvalues can be performed through eigendecomposition or through Singular Value Decomposition; R's `prcomp()` function uses SVD.

Fourth and finally, the resulting eigenvector and eigenvalue pairs are **interpreted** in a specific way (see [2, 5, 7] for mathematical proofs):

- Each eigenvector is used as a principal component; that is, the numeric values in each eigenvector are the coefficients of each principal component, which in turn can be thought of as the weights of the features in the original coordinate space.
- Each eigenvalue is the variance of the original data points projected upon the axis defined by the corresponding eigenvector. Note that since the R software returns the eigenvectors in the order of decreasing eigenvalue, we conveniently have the principal components returned in the order of decreasing projected variance.
- The new principal components define a new coordinate system in which the data instances, after being projected onto the principal components, have new features with zero covariance (and thus zero correlation). We will demonstrate this outcome in Section 4 with R code.

## 2.4 Choosing the number of principal components

For a data set with  $D$  dimensions (features), PCA returns  $D$  principal components, and it is up to the user of the software to select some number  $K$  of those  $D$  principal components to use. Note that it would probably not make sense to use all  $D$  of the principal components, as that would project the original  $D$ -dimensional data instances right back onto a  $D$ -dimensional subspace, meaning that there would be no dimensionality reduction at all.

For visualization on a flat 2-D surface such as a computer screen, selecting  $K$  to be 2 or 3 would be appropriate.

For the purpose of reducing the number of features in order to reduce the amount of data and improve the running time of a machine learning training algorithm, the PCA software user must make a choice for  $K$ . On the one hand, a very small  $K$  would be desirable because it would reduce the amount of data, but on the other hand, if too many dimensions are removed, the data may not capture important details. In the context of PCA, the notion of capturing details is quantified by the amount of total variance explained by the selected principal components. Recall that when the original data points are projected onto an axis defined by a principal component, the variance of this projected data can be computed. The amount of total variance explained is then the sum of the variances over all  $K$  principal components.

	PC1	PC2	PC3	PC4
Sepal.Length	0.5210659	-0.37741762	0.7195664	0.2612863
Sepal.Width	-0.2693474	-0.92329566	-0.2443818	-0.1235096
Petal.Length	0.5804131	-0.02449161	-0.1421264	-0.8014492
Petal.Width	0.5648565	-0.06694199	-0.6342727	0.5235971
Variance	2.918	0.9140	0.1468	0.0207
Fraction variance	0.7296	0.2285	0.0367	0.0052
Cumulative variance	0.7296	0.9581	0.9948	1.0000

Figure 5: Table showing different measures of variance across the four principal components found from the Iris data set. The fifth row shows the variance explained by each PC. The sixth row shows the fraction of total variance for each PC. The seventh row shows the cumulative fraction of total variance.

In the table of Figure 5 we again show the four principal components PC1 to PC4 from the Iris data set. Note that the fifth row contains the variance explained by each of the principal components. For example, it can be seen that PC1 produces a variance of 2.918, which is the variance of the data points after being projected onto PC1. The sixth row contains the fraction of the total variance, where the total is simply the sum of the variances over all the principal components. Here, the total variance is  $2.918 + 0.9140 + 0.1468 + 0.0207 = 3.9995$ , and so the fraction of the total variance explained by PC1 is  $\frac{2.918}{3.9995} = 0.7296$ .

The bottom row of the table finally shows the cumulative fraction of total variance explained by the principal components. *In general, we would like to choose the smallest  $K$  such that 0.85 to 0.99 (equivalently, 85% to 95%) of the total variance is explained*, where these values follow from PCA best practices. In this paper we use the value of 0.95. Because the PCA software returns the principal components in order of decreasing variance, we can simply look across the bottom row of this table from left to right to find the smallest number  $K$  of principal components such that the cumulative fraction of explained variance exceeds 0.95. In this case,  $K = 2$  principal components (PC1 and PC2) allow us to reach over 0.95 (more specifically, 0.9581).

Thus, when we say that PCA can reduce dimensionality, we mean that PCA can compute principal components and the user can choose the smallest number  $K$  of them that explain 0.95 of the variance. A subjectively satisfactory result would be when  $K$  is small relative to the original number of features  $D$ .

## 2.5 Tips for PCA usage

PCA can be used effectively if the following rules of thumb are followed:

First, one can think of PCA as a method to compress the feature space (that is, reduce the dimensionality) by “squeezing out” correlation. That is, given an input data set with features of varying correlation, PCA produces an output data set where the features are uncorrelated. Thus, PCA can perform this compression only if the original data set contains positively or negatively correlated features. Note that because correlation is simply a unitless, scaled version of covariance<sup>2</sup>, we can equivalently say that PCA reduces dimensionality if the original data has positive or negative covariance. PCA will compute a new subspace where the projected data instances have zero covariance, but if the original data had little covariance with which to start, then PCA cannot help very much to reduce dimensionality.

This problem usually manifests itself in the following manner. Suppose a data set has  $D = 100$  features and PCA returns, as expected, 100 principal components. If the original features had little correlation, it will take a large number of principal components (e.g. 80 to 90) to reach 0.95 cumulative fraction of explained variance. If the original features did have strongly correlated features, then PCA could conceivably reach 0.95 cumulative variance with a small number of principal components (e.g. 5 to 10).

Second, PCA works best if the original data is scaled and centered such that all values  $x_i$  of a feature  $x$  are modified to make the feature have mean 0.0 and standard deviation 1.0. The resulting feature values are then defined by  $x'_i = (x_i - \mu_x)/\sigma_x$ . This process is known as standardizing or z-scaling.

Third, PCA requires that data features are numeric, i.e. integers or floats. A categorical feature (an **R factor**) must be replaced by dummy binary integer variables, one for each value of the feature. For example, if a feature for **vehicle** has three possible values **CAR**, **TRUCK**, and **MOTORCYCLE**, then the **vehicle** feature would be replaced by three binary features, such as **is\_CAR**, **is\_TRUCK**, and **is\_MOTORCYCLE**.

## 3 Applying PCA for data exploration

In this section we apply PCA to three different data sets for the purpose of data exploration. All the data sets are available for free by following the relevant provided pointers.

<sup>2</sup>Suppose  $X$  and  $Y$  are two lists of numbers with standard deviations  $\sigma_X$  and  $\sigma_Y$ , respectively. The Pearson correlation between  $X$  and  $Y$  is defined to be  $cor(X, Y) = \frac{cov(X, Y)}{\sigma_X \sigma_Y}$ .

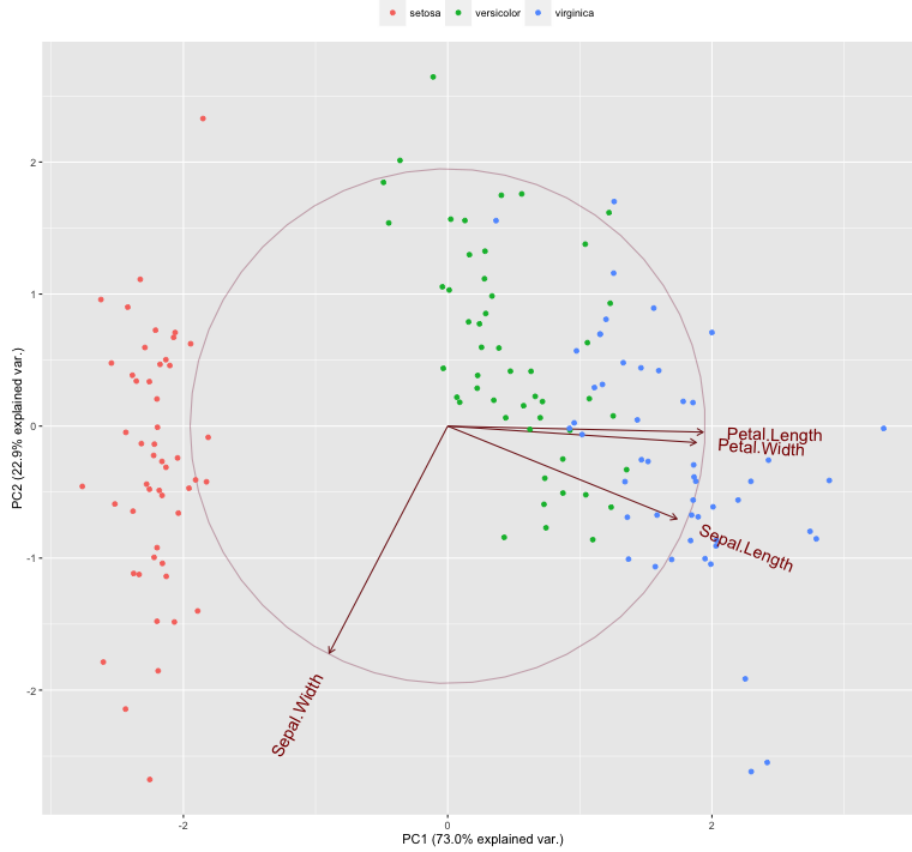


Figure 6: Biplot of the Iris data set. The PC1 axis explains 0.730 of the variance, while the PC2 axis explains 0.229 of the variance.

### 3.1 Iris flowers

As we showed in earlier sections, we ran PCA on Fisher’s Iris data set, which is part of a library that ships with R. This data set has 150 data instances and 4 features. Figure 6 is a *biplot*, which is a scatterplot that places all 150 original data instances on a 2-D plane with the first two principal components PC1 and PC2 serving as the x-axis and y-axis, respectively. The term “biplot” with its prefix “bi” comes from the fact that there are actually two data sets being represented: the original data instances shown as points in the plot and the principal components shown as the axes [4]. While this figure shows a 2-D biplot, one can create a 3-D biplot if there are three axes.

Recall from Section 2.1 that an original data instance is projected onto an axis defined by a principal component by computing the dot product between the data instance’s features and the principal component. Here, we can see that each data instance has been projected onto the two axes and placed at

its proper position in the new coordinate system.

The label on each axis shows the fraction of total variance explained by the axis, and so it can be seen that these first two components together explain 0.959 of the variance (where PC1 explains 0.730 and PC2 explains 0.229). As we will discuss in the next subsection, this result is due to the fact that the data set has high correlation among its original features. Additionally, because the Iris data set has flower species labels for each data instance, the plot also shows the species of each of the 150 data instances as a color (e.g. blue dots indicate the virginica species).

Such a biplot always has one directed arrow for each original feature, where the arrow indicates the direction in which the associated feature increases. For example, the **Petal.Length** arrow points towards the right, so flowers with large values for **Petal.Length** were placed in that direction relative to the center of the plot. The virginica flowers have this characteristic, and so it can be observed that the blue dots were, as expected, placed in the direction that the arrow points. On the other hand, the setosa

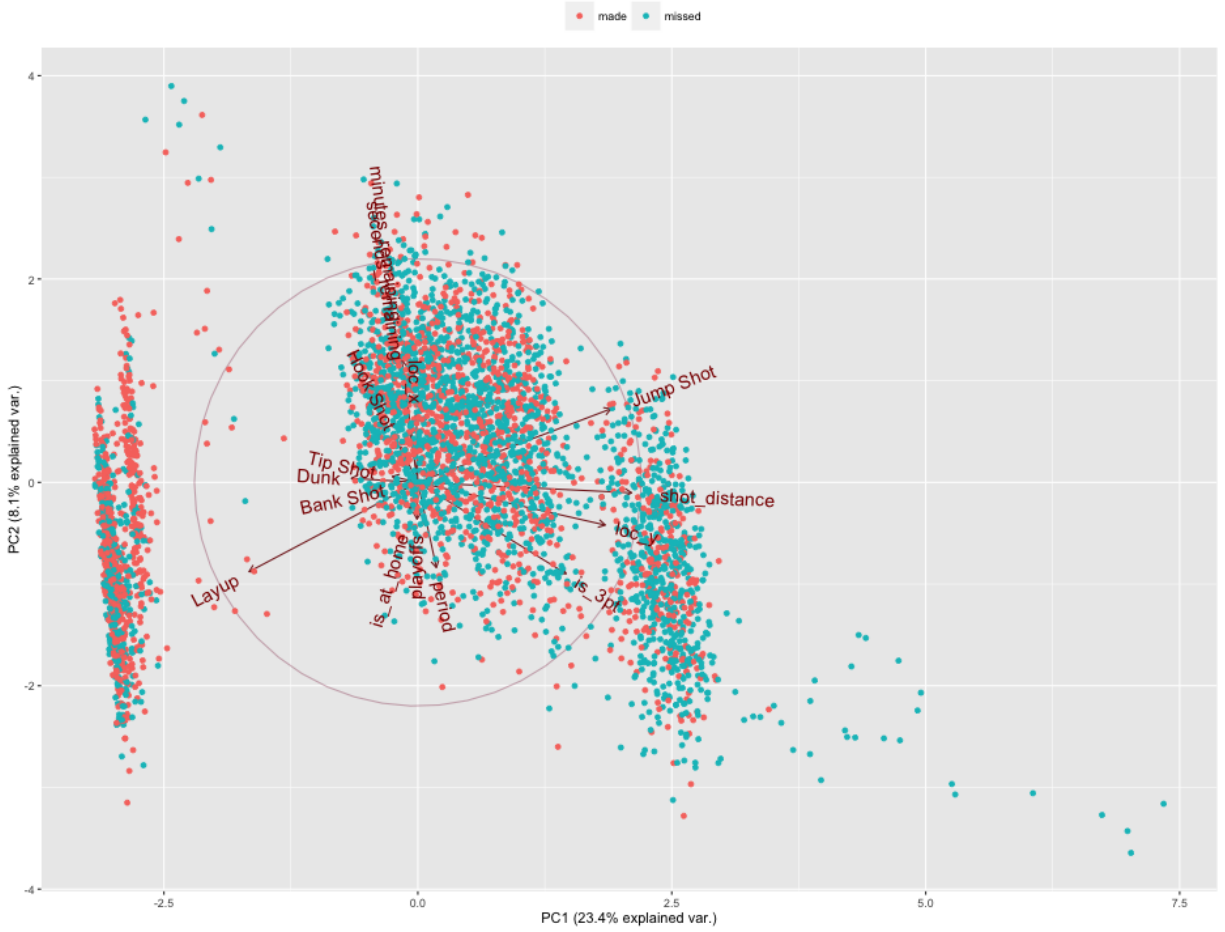


Figure 7: Biplot of the Kobe Bryant shot data set.

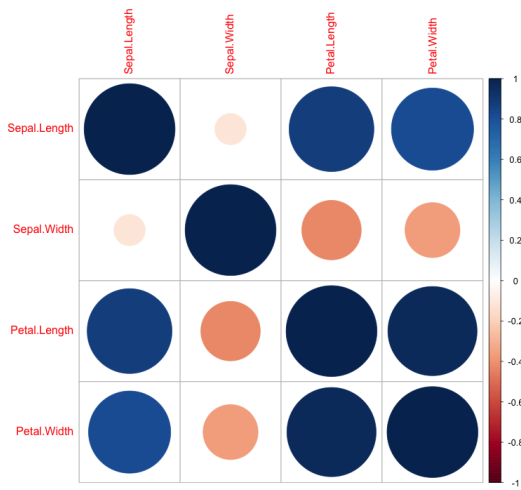


Figure 8: Pairwise feature correlation `corrplot` of the Iris data set.

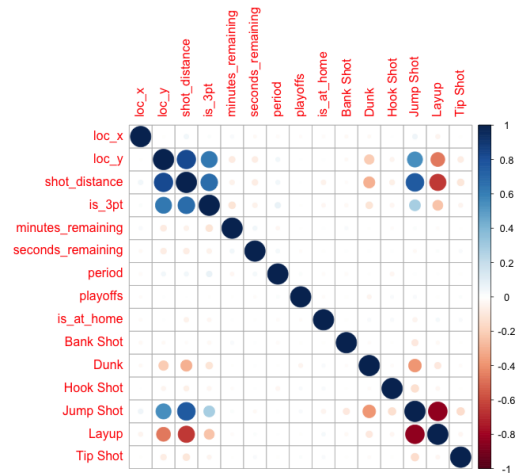


Figure 9: Pairwise feature correlation `corrplot` of the Kobe Bryant shot data set.



flowers have small petal lengths, and so their respective red dots were placed in the opposite direction. If two arrows point in the same direction or in opposite directions, then the corresponding features have strong positive or negative correlation, respectively.

### 3.2 Kobe Bryant's shots

We additionally explored a data set of shots by Kobe Bryant, an NBA basketball player, that was made available for a machine learning competition on the Kaggle.com website<sup>3</sup>. This data set has 30697 data instances and 25 features; however, to improve visualization legibility, we sampled the data down to 4197 data instances, and to keep only relevant features, we selected 15 specific features.

The biplot is shown in Figure 7, where blue points are missed shots and red points are made shots. The biplot shows that there are more missed shots to the right of the scatterplot. This trend is explained by looking at the vector arrows. The original feature `shot_distance` points to the right while the feature `Layup` points to the left; as one would expect in basketball, these features point in opposite directions because a layup shot is very close to the basket. This biplot thus shows that more shots are made closer to the basket (usually as layup shots), while more shots are missed farther away from the basket.

Note that the two axes explain only 0.312 of the total variance, so there are other axes that would help to separate the data. The low fraction is due to the original data set having little correlation between the original features.

This fact can be more easily seen by visualizing the correlation between every pair of features using R's `corrplot()` plotting function. Figure 8 shows the output of `corrplot` applied to the Iris data set. It can be observed that there is high (positive or negative) correlation between its four features, as indicated by large circles on the off-diagonal entries (for example, there is a large blue circle indicating high positive correlation between `Petal.Width` and `Sepal.Length`). The fact that the Iris data set has high feature correlation reveals why its top two principal components can explain a large fraction (0.959) of the data's variance. On the other hand, Figure 9 shows that there is little pairwise feature correlation in the Kobe Bryant shot data set, which explains why its top two principal components can explain only 0.312 of the data variance.

<sup>3</sup><https://www.kaggle.com/c/kobe-bryant-shot-selection>, retrieved April 14, 2016.

### 3.3 Car class fuel economy

We also explored a data set from the United States Department of Energy pertaining to the fuel economy of automobiles<sup>4</sup>. The data originally contained 37146 car instances with 83 features. However, to improve visualization, we sampled the data to those cars produced after the year 2000 and then sampled down again to 500 car instances. Furthermore, from inspecting the data, we saw that many of the feature variables were largely zeros and thus did not contribute useful information, so we selected 8 features that were mostly non-sparse.

Figure 10 shows the biplot of the data. The two axes explain 0.948 of the variance because the data had correlated features. In this data set, there are five classes of cars ranging in increasing physical size across: compact cars; midsize cars; large cars; sport utility vehicles (SUVs) with 2-wheel drive; and SUVs with 4-wheel drive. It can be seen that the compact cars (red dots) and midsize cars (gold) are located on the right of the plot, while the large cars (green) and SUVs (blue and magenta) are on the left side. These placements stem from the feature vector arrows, where the features for increased fuel efficiency (in miles per gallon) point to the right and the features for the size of the car engine (in engine displacement liters and number of engine cylinders) point in the opposite direction to the left. The arrow orientations seem to coincide with the intuition that, in general, larger gas-consuming car engines tend to have decreased fuel efficiency.

## 4 Example R code

In Figure 11 we show sample R code to apply PCA to the Iris data set and generate the biplot of Figure 6. The R function `prcomp()` runs SVD internally. Also note that the `prcomp()` call contains flags to perform scaling and centering.

In Figure 12 we show the use of `corrplot` to visualize the pairwise feature correlation from the Iris data set as was shown in Figure 8. Note that `corrplot` takes as input a correlation matrix, such as the one computed by the `cor` function.

Figure 13 shows complete R code to apply PCA and transform the Iris data set. On line 2, we call `prcomp()` to run PCA, and on line 4 we print the resulting principal components. On line 26 we use the `predict()` function to generate a new data set by projecting the original data instances onto those

<sup>4</sup><http://www.fueleconomy.gov/feg/download.shtml>, retrieved May 27, 2016

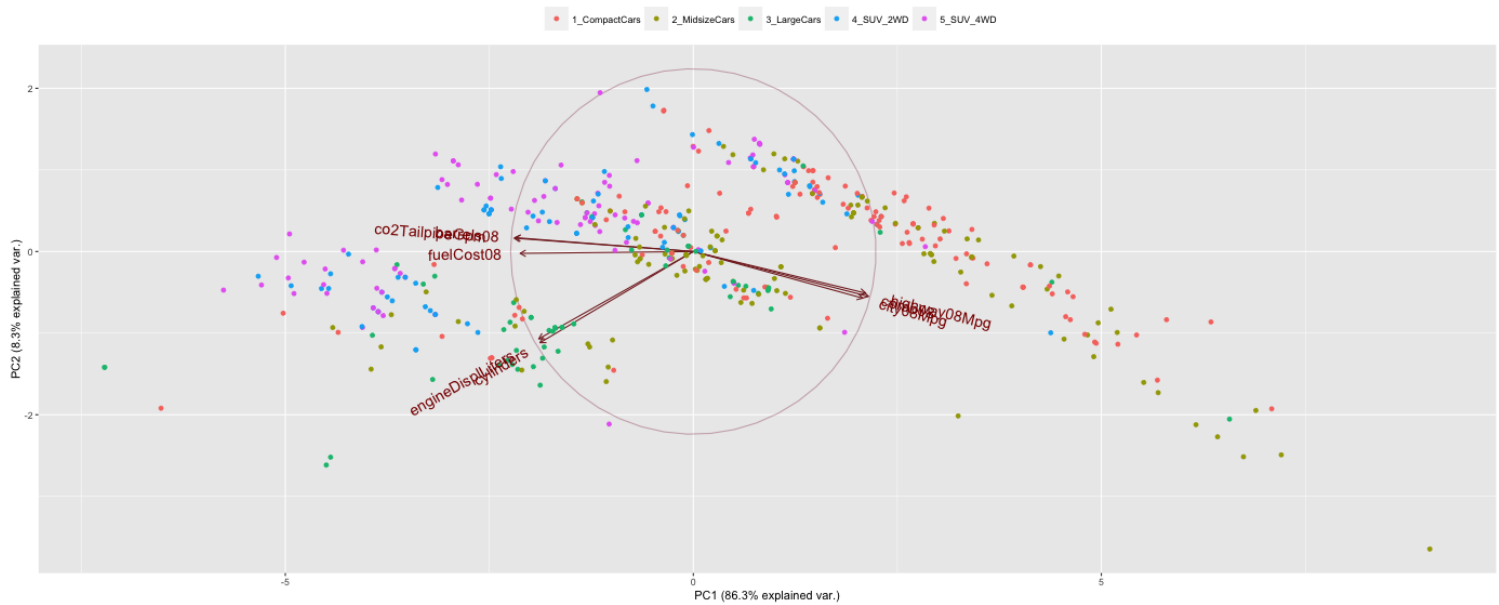


Figure 10: Biplot of the car fuel economy data set.

```

1  # Install ggbiplot by following instructions:
2  # https://github.com/vqv/ggbiplot
3  library(ggbiplot)
4
5  # Run PCA.
6  iris.pca <- prcomp(iris[,1:4], center=T, scale=T)
7
8  # Print useful information.
9  print(iris.pca)
10 print(summary(iris.pca))
11
12 # Plot the figure here.
13 g <- ggbiplot(iris.pca,
14               obs.scale = 1,
15               var.scale = 1,
16               groups = iris$Species,
17               ellipse = TRUE,
18               circle = TRUE) +
19   scale_color_discrete(name = '') +
20   theme(legend.direction = 'horizontal', legend.position = 'top')
21 print(g)

```

Figure 11: R code to generate Figure 6.

```

1  library(corrplot)
2
3  df <- subset(iris, select=c(Species))
4
5  # Create the correlation matrix.
6  correlationMatrix <- cor(df)
7
8  # Set formatting so corrplot does not truncate the bottom.
9  par(xpd=TRUE)
10 mymargin <- c(2,0,1,0)
11
12 # Run corrplot
13 print(corrplot(correlationMatrix, method="circle", mar=mymargin))

```

Figure 12: R code to generate Figure 8.

```

1 # Run PCA here with prcomp().
2 iris.pca <- prcomp(iris[,1:4], center=T, scale=T)
3
4 print(iris.pca)
5 # Standard deviations:
6 # [1] 1.7083611 0.9560494 0.3830886 0.1439265
7 # Rotation:
8 #
9 # Sepal.Length 0.5210659 -0.37741762 0.7195664 0.2612863
10 # Sepal.Width -0.2693474 -0.92329566 -0.2443818 -0.1235096
11 # Petal.Length 0.5804131 -0.02449161 -0.1421264 -0.8014492
12 # Petal.Width 0.5648565 -0.06694199 -0.6342727 0.5235971
13
14 print(summary(iris.pca))
15 # Importance of components:
16 #
17 # Standard deviation 1.7084 0.9560 0.38309 0.14393
18 # Proportion of Variance 0.7296 0.2285 0.03669 0.00518
19 # Cumulative Proportion 0.7296 0.9581 0.99482 1.00000
20
21 # -----
22 # Now, compute the new dataset aligned to the PCs by
23 # using the predict() function.
24 # -----
25
26 df.new <- predict(iris.pca, iris[,1:4])
27 head(df.new)
28 #
29 # PC1 PC2 PC3 PC4
30 # [1,] -2.2571141 -0.4784238 0.12727962 0.024087508
31 # [2,] -2.074013 0.6718827 0.23382552 0.102662845
32 # [3,] -2.356335 0.3407664 -0.04405390 0.028282305
33 # [4,] -2.291707 0.5953999 -0.09098530 -0.065735340
34 # [5,] -2.381863 -0.6446757 -0.01568565 -0.035802870
35 # [6,] -2.068701 -1.4842053 -0.02687825 0.006586116
36
37 # Show the PCA model's sdev values are the square root
38 # of the projected variances, which are along the diagonal
39 # of the covariance matrix of the projected data.
40 iris.pca$sdev^2
41 # [1] 2.91849782 0.91403047 0.14675688 0.02071484
42
43 # Compute covariance matrix for new data set.
44 round(cov(df.new), 5)
45 #
46 # PC1 PC2 PC3 PC4
47 # PC1 2.9185 0.00000 0.00000 0.00000
48 # PC2 0.0000 0.91403 0.00000 0.00000
49 # PC3 0.0000 0.00000 0.14676 0.00000
50 # PC4 0.0000 0.00000 0.00000 0.02071

```

Figure 13: R code to run PCA on the Iris data set and see the underlying principal components. Note that the PCA results contain the standard deviations of the projected data (as shown on lines 5-6 and 39-40) rather than the variances. Recall that the standard deviation is the square root of the variance.

```

1  # Scale and center the data.
2  df.scaled <- scale(iris[,1:4], center=T, scale=T)
3
4  # Compute the covariance matrix.
5  cov.df.scaled <- cov(df.scaled)
6
7  # Compute the eigenvectors and eigenvalues.
8  # Each eigenvector (column) is a principal component.
9  # Each eigenvalue is the variance explained by the
10 # associated eigenvector.
11 eigenInformation <- eigen(cov.df.scaled)
12
13 print(eigenInformation)
14 # $values
15 # [1] 2.91849782 0.91403047 0.14675688 0.02071484
16 # $vectors
17 #           [,1]      [,2]      [,3]      [,4]
18 # [1,] 0.5210659 -0.37741762 0.7195664 0.2612863
19 # [2,] -0.2693474 -0.92329566 -0.2443818 -0.1235096
20 # [3,] 0.5804131 -0.02449161 -0.1421264 -0.8014492
21 # [4,] 0.5648565 -0.06694199 -0.6342727 0.5235971
22
23 # -----
24 # Now, compute the new dataset aligned to the PCs by
25 # multiplying the eigenvector and data matrices.
26 # -----
27
28 # Create transposes in preparation for matrix mult.
29 eigenvectors.t <- t(eigenInformation$vectors) # 4x4
30 df.scaled.t <- t(df.scaled) # 4x150
31
32 # Perform matrix multiplication.
33 df.new <- eigenvectors.t %*% df.scaled.t # 4x150
34
35 # Create new data frame. First take transpose and
36 # then add column names.
37 df.new.t <- t(df.new) # 150x4
38 colnames(df.new.t) <- c("PC1", "PC2", "PC3", "PC4")
39
40 head(df.new.t)
41 #           PC1      PC2      PC3      PC4
42 # [1,] -2.257141 -0.4784238 0.12727962 0.024087508
43 # [2,] -2.074013 0.6718827 0.23382552 0.102662845
44 # [3,] -2.356335 0.3407664 -0.04405390 0.028282305
45 # [4,] -2.291707 0.5953999 -0.09098530 -0.065735340
46 # [5,] -2.381863 -0.6446757 -0.01568565 -0.035802870
47 # [6,] -2.068701 -1.4842053 -0.02687825 0.006586116
48
49 # Show that the eigenvalues are the numbers on the
50 # diagonal of the diagonalized covariance matrix.
51
52 print(eigenInformation$values)
53 # [1] 2.91849782 0.91403047 0.14675688 0.02071484
54
55 # Compute covariance matrix for new data set.
56 round(cov(df.new.t), 5)
57 #           PC1      PC2      PC3      PC4
58 # PC1 2.9185 0.00000 0.00000 0.00000
59 # PC2 0.0000 0.91403 0.00000 0.00000
60 # PC3 0.0000 0.00000 0.14676 0.00000
61 # PC4 0.0000 0.00000 0.00000 0.02071

```

Figure 14: R code to run the same PCA operation on the Iris data set from Figure 13, but instead of using `prcomp()`, the code explicitly computes the eigenvectors and eigenvalues using the `eigen()` function.

principal components. When a new covariance matrix is computed for the new data set on line 43, it can be seen that the features have zero correlation, as explained earlier in Section 2.3. Furthermore, the variances of the features in the projected space, as shown on the diagonal of the new covariance matrix, are indeed the variances computed by PCA on the original data set.

For completeness, Figure 14 shows another execution of PCA but this time *without* using `prcomp()`. Instead, on line 11 we apply the `eigen()` function to explicitly calculate the eigenvectors and eigenvalues in order to achieve the same results. Importantly, the original data set must be scaled and centered manually with the call to `scale()`. Additionally, on lines 28-37 we show that in order to produce the projected data, many steps must be taken (such as taking transposes and performing matrix multiplication), all of which are conveniently hidden in the call to `predict()` in the previous example.

## 5 Further reading

The work by Smith provides a readable introduction to the linear algebra derivation of PCA [9]. A tutorial by Martins shows how to produce biplots [8]. The STDHA website offers a wealth of tutorials for using PCA with R [10].

The textbooks by Bishop [2, Chapter 12] and James et al. [5, Chapter 10] have chapters with readable mathematical explanations of PCA. The textbook by Jolliffe provides a rigorous mathematical treatment [7].

## 6 Conclusion

Principal component analysis is an algorithm to reduce the number of dimensions of a data set by linearly combining original features into fewer features. The result is that the original data is projected onto a lower-dimensional coordinate system whose axes define new uncorrelated features.

PCA is generally used for two purposes: (1) as a standalone unsupervised machine learning algorithm, it allows high-dimensional data to be reduced to a few dimensions so that the data can be more easily visualized and explored; and (2) as a preprocessing step, it reduces the number of features, which in turn reduces the amount of data needed by other machine learning algorithms.

At the core of PCA are the principal components. These numeric vectors have a direction chosen such that projection error is minimized and (equivalently)

the variance of the projected data is maximized. The resulting optimal principal components are computed by following a series of linear algebra steps that involve (1) scaling and centering the original data instances, (2) computing a covariance matrix from the data, (3) computing the eigenvectors and eigenvalues, (4) taking the eigenvectors as principal components, and (5) taking the eigenvalues as the variance of the data when projected onto the axes defined by the principal components.

Choosing the number of principal components requires thought. For visualizing data on paper or a computer monitor, keeping two or three is reasonable. For reducing the amount of training data used by a later machine learning algorithm, it is appropriate to choose enough principal components that can explain a large fraction of the variance of the projected data.

## 7 Exercises for Self-Study

1. Suppose you run PCA on a data set with 500 numeric features. How many principal components will PCA produce?
2. Consider an algorithm called CarelessPCA which comprises the same linear algebra steps from PCA as described in Section 2.3 *except* that the data is not initially scaled and centered. Assume that you can run CarelessPCA and PCA on the same data and that the data is non-trivial (e.g. the data values are not all zeros). State whether each of the following statements is true or false in general. Explain your answers.
  - (a) The number of resulting principal components is the same for both PCA and CarelessPCA.
  - (b) The numeric values in the first principal component are the same for both PCA and CarelessPCA.
  - (c) The principal components from CarelessPCA are orthogonal to one another.
  - (d) The eigenvalues of the covariance matrix are the same for both PCA and CarelessPCA.
3. Suppose you have two different data sets A and B, both of which have 500 numeric features. You then run PCA on both of them and observe that in order to explain 95% of the variance, you need 5 principal components for data set A but 400 principal components for data set B. Provide some qualitative reasons for these results. How would you confirm your hypothesis?

## 8 Answers to Exercises

Below are answers to the exercises from the previous section.

1. PCA will produce 500 principal components, one for each of the original numeric features.
2. (a) True.  
(b) False.  
(c) True. The eigenvectors of a covariance matrix are orthogonal regardless of how the values of the data set are formed.  
(d) False. Scaling the original data will cause the variance of the projected data to be different.
3. Data set A likely contains many correlated features, while data set B likely contains few. The validity of this hypothesis could be examined by using `corrplot` to visualize the correlation matrix or (2) computing the average correlation between all the features.

- [7] I. Jolliffe. *Principal Component Analysis*, Springer, 2002.
- [8] T. Martins. “Computing and Visualizing PCA in R,” <http://www.r-bloggers.com/computing-and-visualizing-pca-in-r/>
- [9] L. Smith. “A Tutorial on Principal Components Analysis,” [http://www.cs.otago.ac.nz/cosc453/student\\_tutorials/principal\\_components.pdf](http://www.cs.otago.ac.nz/cosc453/student_tutorials/principal_components.pdf)
- [10] Statistical Tools for High-Throughput Data Analysis website. “Principal component analysis: the basics you should read,” <http://www.sthda.com/english/wiki/principal-component-analysis-the-basics-you-should-read-r-software-and-data-mining>

## Acknowledgements

We would like to thank Leonardo Jimenez Rodriguez and Justin Martineau for their feedback on earlier drafts of this document.

## References

- [1] E. Anderson. “The irises of the Gaspe Peninsula,” *Bulletin of the American Iris Society*, 59, pp. 2-5, 1935.
- [2] C. Bishop. *Pattern Recognition and Machine Learning*, Springer, 2007.
- [3] R. Fisher. “The use of multiple measurements in taxonomic problems,” *Annual Eugenics*, 7(2), pp. 179-188, 1936.
- [4] M. Greenacre. *Biplots in Practice*. <http://www.multivariatestatistics.org/chapter1.html>
- [5] G. James, D. Witten, T. Hastie, and R. Tibshirani. *An Introduction to Statistical Learning*, Springer, 2013.
- [6] G. John, R. Kohavi, and K. Pfleger. “Irrelevant Features and the Subset Selection Problem,” In *Proceedings of the Eleventh International Conference on Machine Learning*, 1994.