CHAPTER 6

Dimensionality Reduction: Squashing the Data Pancake with PCA

With automatic data collection and feature generation techniques, one can quickly obtain a large number of features. But not all of them are useful. In Chapters 3 and 4, we discussed frequency-based filtering and feature scaling as ways of pruning away uninformative features. Now we will take a close look at the topic of feature dimensionality reduction using *principal component analysis* (PCA).

This chapter marks an entry into model-based feature engineering techniques. Prior to this point, most of the techniques can be defined without referencing the data. For instance, frequency-based filtering might say, "Get rid of all counts that are smaller than *n*," a procedure that can be carried out without further input from the data itself.

Model-based techniques, on the other hand, require information from the data. For example, PCA is defined around the principal axes of the data. In previous chapters, there was always a clear-cut line between data, features, and models. From this point forward, the difference gets increasingly blurry. This is exactly where the excitement lies in current research on feature learning.

Intuition

Dimensionality reduction is about getting rid of "uninformative information" while retaining the crucial bits. There are many ways to define "uninformative." PCA focuses on the notion of linear dependency. In "The Anatomy of a Matrix" on page 182, we describe the column space of a data matrix as the span of all feature vectors. If the column space is small compared to the total number of features, then most of the features are linear combinations of a few key features. Linearly dependent features are a waste of space and computation power because the information could have been encoded in much fewer features. To avoid this situation, principal component analysis tries to reduce such "fluff" by squashing the data into a much lower-dimensional linear subspace.

Picture the set of data points in feature space. Each data point is a dot, and the whole set of data points forms a blob. In Figure 6-1(a), the data points spread out evenly across both feature dimensions, and the blob fills the space. In this example, the column space has full rank. However, if some of those features are linear combinations of others, then the blob won't look so plump; it will look more like Figure 6-1(b), a flat blob where feature 1 is a duplicate (or a scalar multiple) of feature 2. In this case, we say that the *intrinsic dimensionality* of the blob is 1, even though it lies in a two-dimensional feature space.

In practice, things are rarely exactly equal to one another. It is more likely that we see features that are very close to being equal, but not quite. In such a case, the data blob might look something like Figure 6-1(c). It's an emaciated blob. If we wanted to reduce the number of features to pass to the model, then we could replace feature 1 and feature 2 with a new feature, maybe called feature 1.5, which lies on the diagonal line between the original two features. The original dataset could then be adequately represented by one number—the position along the direction of feature 1.5—instead of two numbers, *f1* and *f2*.

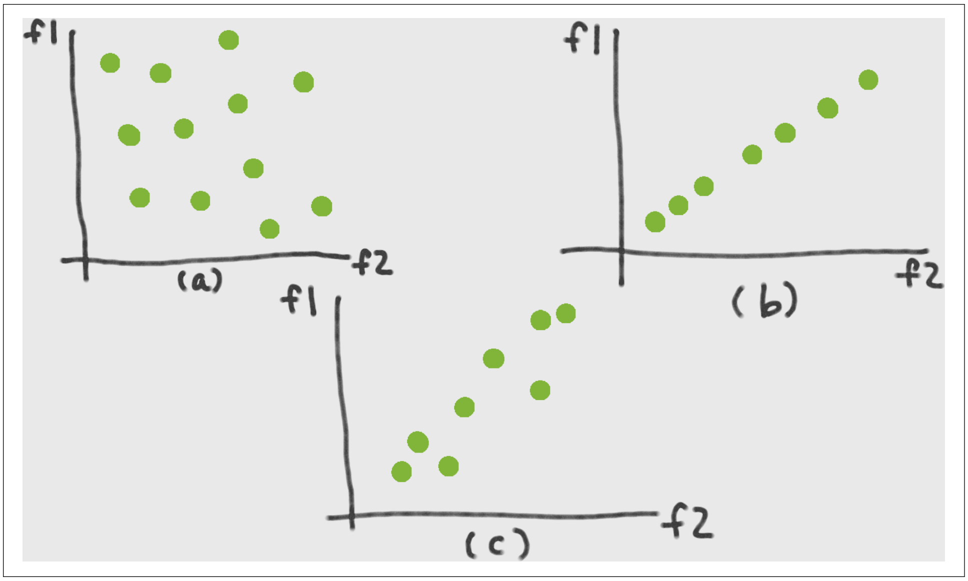

Figure 6-1. Data blobs in feature space: (a) full-rank data blob, (b) low-dimensional data blob, and (c) approximately low-dimensional data blob

The key idea here is to *replace redundant features with a few new features that adequately summarize information contained in the original feature space*. It's easy to tell what the new feature should be when there are only two features. It's much harder when the original feature space has hundreds or thousands of dimensions. We need a way to mathematically describe the new features we are looking for. Then we can use optimization techniques to find them.

One way to mathematically define "adequately summarize information" is to say that the new data blob should retain as much of the original volume as possible. We are squashing the data blob into a flat pancake, but we want the pancake to be as big as possible in the right directions. This means we need a way to measure volume.

Volume has to do with distance. But the notion of distance in a blob of data points is somewhat fuzzy. One could measure the maximum distance between any two pairs of points, but that turns out to be a very difficult function to mathematically optimize. An alternative is to measure the average distance between pairs of points, or equivalently, the average distance between each point and its mean, which is the variance. This turns out to be much easier to optimize. (Life is hard. Statisticians have learned to take convenient shortcuts.) Mathematically, this translates into maximizing the variance of the data points in the new feature space.

Tips for Navigating Linear Algebra Formulas

To stay oriented in the world of linear algebra, keep track of which quantities are scalars, which are vectors, and which way the vectors are oriented—vertically or horizontally. Know the dimensions of your matrices, because they often tell you whether the vectors of interest are in the rows or columns. Draw the matrices and vectors as rectangles on a page and make sure the shapes match. Just as one can get far in algebra by noting the units of measurement (distance is in miles, speed is in miles per hour), in linear algebra all one needs are the dimensions.

Derivation

As before, let $\boldsymbol{X}$ denote the $n \times d$ data matrix, where n is the number of data points and d the number of features. Let $\mathbf{x}$ be a column vector containing a single data point. (So $\mathbf{x}$ is the transpose of one of the rows in $\boldsymbol{X}$.) Let $\mathbf{v}$ denote one of the new feature vectors, or principal components, that we are trying to find.

Singular Value Decomposition (SVD) of a Matrix

Any rectangular matrix can be decomposed into three matrices of particular shapes and characteristics:

$$\boldsymbol{X} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

Here, $\boldsymbol{U}$ and $\boldsymbol{V}$ are orthogonal matrices (i.e., $\boldsymbol{U}^T\boldsymbol{U} = I$ and $\boldsymbol{V}^T\boldsymbol{V} = I$). $\boldsymbol{\Sigma}$ is a diagonal matrix containing the singular values of $\boldsymbol{X}$, which can be positive, zero, or negative. Suppose $\boldsymbol{X}$ has n rows and d columns and $n \geq d$. Then $\boldsymbol{U}$ has shape $n \times d$, and $\boldsymbol{\Sigma}$ and $\boldsymbol{V}$ have shape $d \times d$. (See "Singular Value Decomposition (SVD)" on page 185 for a full review of SVD and eigen decomposition of a matrix.)

Linear Projection

Let's break down the derivation of PCA step by step. Figure 6-2 illustrates the whole process.

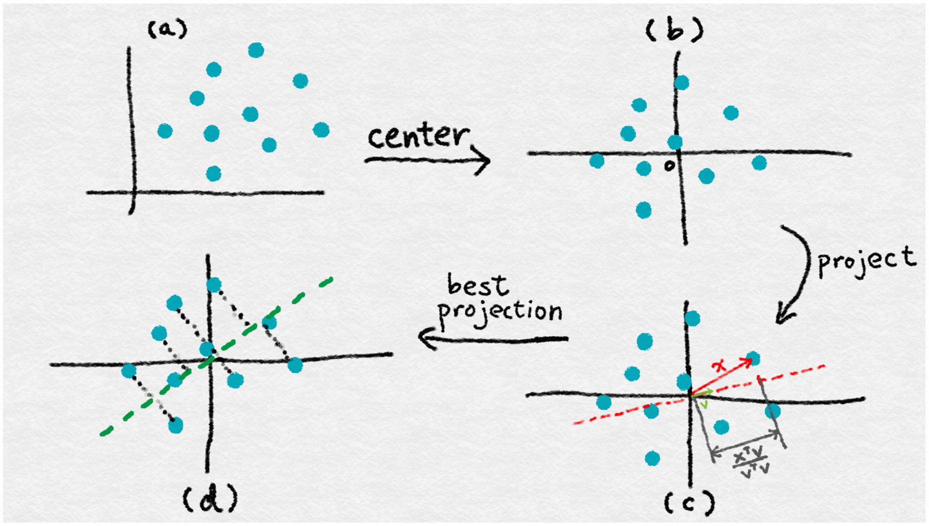

Figure 6-2. Illustration of PCA: (a) original data in feature space; (b) centered data; (c) projecting a data vector x onto another vector v; (d) direction of maximum variance of the projected coordinates (equal to the principal eigenvector of X^TX)

PCA uses linear projection to transform data into the new feature space. Figure 6-2(c) illustrates what a linear projection looks like. When we project **x** onto **v**, the length of the projection is proportional to the inner product between the two, normalized by the norm of **v** (its inner product with itself). Later on, we will constrain **v** to have unit norm. So, the only relevant part is the numerator—let's call it **v** (see Equation 6-1).

Equation 6-1. Projection coordinate

$$z = \mathbf{x}^T\mathbf{v}$$

Note that z is a scalar, whereas **x** and **v** are column vectors. Since there are a bunch of data points, we can formulate the vector **z** of all of their projection coordinates on the new feature **v** (Equation 6-2). Here, $\boldsymbol{X}$ is the familiar data matrix where each row is a data point. The resulting **z** is a column vector.

Equation 6-2. Vector of projection coordinates

$$\mathbf{z} = \boldsymbol{X}\mathbf{v}$$

Variance and Empirical Variance

The next step is to compute the variance of the projections. Variance is defined as the expectation of the squared distance to the mean (Equation 6-3).

Equation 6-3. Variance of a random variable Z

$$\mathrm{Var}(Z) = \mathrm{E}[Z - \mathrm{E}(Z)]^2$$

There is one tiny problem: our formulation of the problem says nothing about the mean, *E(Z)*; it is a free variable. One solution is to remove it from the equation by subtracting the mean from every data point. The resulting dataset has mean zero, which means that the variance is simply the expectation of Z^2. Geometrically, subtracting the mean has the effect of centering the data. (See Figure 6-2(a-b).)

A closely related quantity is the covariance between two random variables Z^1 and Z^2 (Equation 6-4). Think of this as the extension of the idea of variance (of a single random variable) to two random variables.

Equation 6-4. Covariance between two random variables Z^1 and Z^2

$$\mathrm{Cov}(Z^1, Z^2) = \mathrm{E}[(Z^1 - \mathrm{E}(Z^1)(Z^2 - \mathrm{E}(Z^2)]$$

When the random variables have mean zero, their covariance coincides with their *linear correlation*, $E[Z_1 Z_2]$. We will discuss this concept more later on.

Statistical quantities like variance and expectation are defined on a data distribution. In practice, we don't have the true distribution, but only a bunch of observed data points, z_1, ..., z_n. This is called an *empirical distribution*, and it gives us an empirical estimate of the variance (Equation 6-5).

Equation 6-5. Empirical variance of Z based on observations z

$$\mathrm{Var}_{\mathrm{emp}}(Z) = \frac{1}{n-1}\sum_{i=1}^{n} z_i^2$$

Principal Components: First Formulation

Combined with the definition of z_i in Equation 6-1, we have the formulation for maximizing the variance of the projected data given in Equation 6-6. (We drop the denominator $n-1$ from the definition of empirical variance, because it is a global constant and does not affect where the maximizing value occurs.)

Equation 6-6. Objective function of principal components

$$\max_{\mathbf{w}} \sum_{i=1}^{n} (\mathbf{x}_i^T \mathbf{w})^2, \quad \text{where } \mathbf{w}^T \mathbf{w} = 1$$

The constraint here forces the inner product of **w** with itself to be 1, which is equivalent to saying that the vector must have unit length. This is because we only care about the direction and not the magnitude of **w**. The magnitude of **w** is an unnecessary degree of freedom, so we get rid of it by setting it to an arbitrary value.

Principal Components: Matrix-Vector Formulation

Next comes the tricky step. The sum of squares term in Equation 6-6 is rather cumbersome. It'd be much cleaner in a matrix-vector format. Can we do it? The answer is yes. The key lies in the sum-of-squares identity: the sum of a bunch of squared terms is equal to the squared norm of a vector whose elements are those terms, which is equivalent to the vector's inner product with itself. With this identity in hand, we can rewrite Equation 6-6 in matrix-vector notation, as shown in Equation 6-7.

Equation 6-7. Objective function for principal components, matrix-vector formulation

$$\max_{\mathbf{w}} \mathbf{w}^T\mathbf{w}, \text{ where } \mathbf{w}^T\mathbf{w} = 1$$

This formulation of PCA presents the target more clearly: we look for an input direction that maximizes the norm of the output. Does this sound familiar? The answer lies in the *singular value decomposition* (SVD) of **X**. The optimal **w**, as it turns out, is the principal left singular vector of $\boldsymbol{X}$, which is also the principal eigenvector of $\boldsymbol{X}^T\boldsymbol{X}$. The projected data is called a principal component of the original data.

General Solution of the Principal Components

This process can be repeated. Once we find the first principal component, we can rerun Equation 6-7 with the added constraint that the new vector be orthogonal to the previously found vectors (see Equation 6-8).

Equation 6-8. Objective function for k+1st principal components

$$\max_{\mathbf{w}} \mathbf{w}^T\mathbf{w}, \text{ where } \mathbf{w}^T\mathbf{w} = 1 \text{ and } \mathbf{w}^T\mathbf{w}_1 = \ldots = \mathbf{w}^T\mathrm{w}_k = 0$$

The solution is the k+1st left singular vectors of $\boldsymbol{X}$, ordered by descending singular values. Thus, the first k principal components correspond to the first k left singular vectors of $\boldsymbol{X}$.

Transforming Features

Once the principal components are found, we can transform the features using linear projection. Let $\boldsymbol{X} = \boldsymbol{U\Sigma V}^T$ be the SVD of $\boldsymbol{X}$, and $\boldsymbol{V}_k$ the matrix whose columns contain the first k left singular vectors. $\boldsymbol{X}$ has dimensions $n \times d$, where d is the number of original features, and $\boldsymbol{V}_k$ has dimensions $d \times k$. Instead of a single projection vector as in Equation 6-2, we can simultaneously project onto multiple vectors in a projection matrix (Equation 6-9).

Equation 6-9. PCA projection matrix

$$\boldsymbol{W} = \boldsymbol{V}_k$$

The matrix of projected coordinates is easy to compute, and can be further simplified using the fact that the singular vectors are orthogonal to each other (see Equation 6-10).

Equation 6-10. Simple PCA transform

$$\boldsymbol{Z} = \boldsymbol{XW} = \boldsymbol{XV}_k = \boldsymbol{U\Sigma V}^T\boldsymbol{V}_k = \boldsymbol{U}_k\boldsymbol{\Sigma}_k$$

The projected values are simply the first k right singular vectors scaled by the first k singular values. Thus, the entire PCA solution, components and projections alike, can be conveniently obtained through the SVD of $\boldsymbol{X}$.

Implementing PCA

Many derivations of PCA involve first centering the data, then taking the eigen decomposition of the covariance matrix. But the easiest way to implement PCA is by taking the singular value decomposition of the centered data matrix.

PCA Implementation Steps

1. Center the data matrix:

   $$\boldsymbol{C} = \boldsymbol{X} - \mathbf{1}\boldsymbol{\mu}^T$$

   where $\mathbf{1}$ is a column vector containing all 1s, and $\boldsymbol{\mu}$ is a column vector containing the average of the rows of $\boldsymbol{X}$.
2. Compute the SVD:

   $$\boldsymbol{C} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T$$

3. Find the principal components. The first k principal components are the first k columns of $\boldsymbol{V}$; i.e., the right singular vectors corresponding to the k largest singular values.
4. Transform the data. The transformed data is simply the first k columns of $\boldsymbol{U}$. (If whitening is desired, then scale the vectors by the inverse singular values. This requires that the selected singular values are nonzero. See "Whitening and ZCA" on page 108.)

PCA in Action

Let's get a better sense for how PCA works by applying it to some image data. The MNIST dataset contains images of handwritten digits from 0 to 9. The original images are 28 × 28 pixels. A lower-resolution subset of the images is distributed with scikit-learn, where each image is downsampled into 8 × 8 pixels. The original data in scikit-learn has 64 dimensions. In Example 6-1, we apply PCA and visualize the dataset using the first three principal components.

Example 6-1. Principal component analysis of the scikit-learn digits dataset (a subset of the MNIST dataset)

```
>>> from sklearn import datasets
>>> from sklearn.decomposition import PCA
```

```
# Load the data
>>> digits_data = datasets.load_digits()
>>> n = len(digits_data.images)

# Each image is represented as an 8-by-8 array.
# Flatten this array as input to PCA.
>>> image_data = digits_data.images.reshape((n, -1))
>>> image_data.shape
(1797, 64)

# Groundtruth label of the number appearing in each image
>>> labels = digits_data.target
>>> labels
array([0, 1, 2, ..., 8, 9, 8])

# Fit a PCA transformer to the dataset.
# The number of components is automatically chosen to account for
# at least 80% of the total variance.
>>> pca_transformer = PCA(n_components=0.8)
>>> pca_images = pca_transformer.fit_transform(image_data)
>>> pca_transformer.explained_variance_ratio_
array([ 0.14890594,  0.13618771,  0.11794594,  0.08409979,  0.05782415,
        0.0491691 ,  0.04315987,  0.03661373,  0.03353248,  0.03078806,
        0.02372341,  0.02272697,  0.01821863])
>>> pca_transformer.explained_variance_ratio_[:3].sum()
0.40303958587675121

# Visualize the results
>>> import matplotlib.pyplot as plt
>>> from mpl_toolkits.mplot3d import Axes3D
>>> %matplotlib notebook
>>> fig = plt.figure()
>>> ax = fig.add_subplot(111, projection='3d')
>>> for i in range(100):
...     ax.scatter(pca_images[i,0], pca_images[i,1], pca_images[i,2],
...                marker=r'${}$'.format(labels[i]), s=64)

>>> ax.set_xlabel('Principal component 1')
>>> ax.set_ylabel('Principal component 2')
>>> ax.set_zlabel('Principal component 3')
```

The first 100 projected images are shown in a 3D plot in Figure 6-3. The markers correspond to the labels. The first three principal components account for roughly 40% of the total variance in the dataset. This is by no means perfect, but it allows for a handy low-dimensional visualization. We see that PCA groups similar numbers close to each other. The numbers 0 and 6 lie in the same region, as do 1 and 7, and 3 and 9. The space is roughly divided between 0, 4, and 6 on one side, and the rest of the numbers on the other.

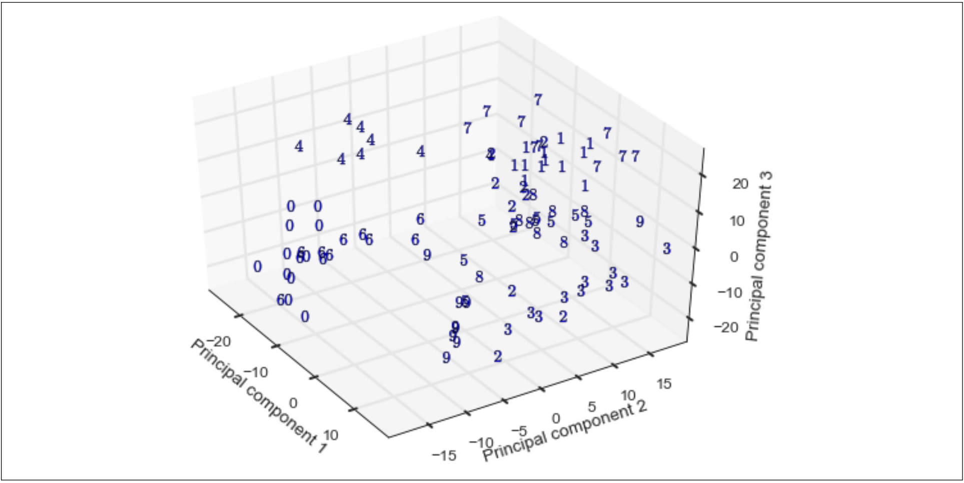

Figure 6-3. PCA projections of subset of MNIST data—markers correspond to image labels

Since there is a fair amount of overlap between numbers, it would be difficult to tell them apart using a linear classifier in the projected space. Hence, if the task is to classify the handwritten digits and the chosen model is a linear classifier, then the first three principal components are not sufficient as features. Nevertheless, it is interesting to see how much of a 64-dimensional dataset can be captured in just 3 dimensions.

Whitening and ZCA

Due to the orthogonality constraint in the objective function, PCA transformation produces a nice side effect: the transformed features are no longer correlated. In other words, the inner products between pairs of feature vectors are zero. It's easy to prove this using the orthogonality property of the singular vectors:

$$\boldsymbol{Z}^T\boldsymbol{Z} = \boldsymbol{\Sigma}_k\boldsymbol{U}_k^T\boldsymbol{U}_k\boldsymbol{\Sigma}_k = \boldsymbol{\Sigma}_k^2$$

The result is a diagonal matrix containing squares of the singular values representing the correlation of each feature vector with itself, also known as its ℓ^2 norm.

Sometimes, it is useful to also normalize the scale of the features to 1. In signal processing terms, this is known as *whitening*. It results in a set of features that have unit correlation with themselves and zero correlation with each other. Mathematically, whitening can done by multiplying the PCA transformation with the inverse singular values (see Equation 6-11).

Equation 6-11. PCA + whitening

$$\boldsymbol{W}_{white} = \boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1}$$

$$\boldsymbol{Z}_{white} = \boldsymbol{X}\boldsymbol{V}_k \boldsymbol{\Sigma}_k^{-1} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}_k\boldsymbol{\Sigma}_k^{-1} = \boldsymbol{U}_k$$

Whitening is independent from dimensionality reduction; one can perform one without the other. For example, *zero-phase component analysis* (ZCA) (Bell and Sejnowski, 1996) is a whitening transformation that is closely related to PCA, but that does not reduce the number of features. ZCA whitening uses the full set of principal components $\boldsymbol{V}$ without reduction, and includes an extra multiplication back onto $\boldsymbol{V}^T$ (Equation 6-12).

Equation 6-12. ZCA whitening

$$\boldsymbol{W}_{ZCA} = \boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^T$$

$$\boldsymbol{Z}_{zca} = \boldsymbol{X}\boldsymbol{V}\boldsymbol{\Sigma}^{-1}\boldsymbol{V}^T = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T\boldsymbol{V}\boldsymbol{\Sigma}^{-1} = \boldsymbol{U}$$

Simple PCA projection (Equation 6-10) produces coordinates in the new feature space, where the principal components serve as the basis. These coordinates represent only the length of the projected vector, not the direction. Multiplication with the principal components gives us the length and the orientation. Another valid interpretation is that the extra multiplication rotates the coordinates back into the original feature space. ($\boldsymbol{V}$ is an orthogonal matrix, and orthogonal matrices rotate their input without stretching or compression.) So, ZCA produces whitened data that is as close (in Euclidean distance) to the original data as possible.

Considerations and Limitations of PCA

When using PCA for dimensionality reduction, one must address the question of how many principal components (k) to use. Like all hyperparameters, this number can be tuned based on the quality of the resulting model. But there are also heuristics that do not involve expensive computational methods.

One possibility is to pick k to account for a desired proportion of total variance. (This option is available in the scikit-learn package `PCA`.) The variance of the projection onto the kth component is:

$$\| \boldsymbol{X}\boldsymbol{v}_k \|^2 = \| \boldsymbol{u}_k \sigma_k \|^2 = \sigma_k^2$$

which is the square of the *k*th-largest singular value of ***X***. The ordered list of singular values of a matrix is called its *spectrum*. Thus, to determine how many components to use, one can perform a simple spectral analysis of the data matrix and pick the threshold that retains enough variance.

Selecting k Based on Accounted Variance

To retain enough components to cover 80% of the total variance in the data, pick *k* such that

$$\frac{\sum_{i=1}^{k} \sigma_i^2}{\sum_{i=1}^{d} \sigma_i^2} \geq 0.8.$$

Another method for picking *k* involves the intrinsic dimensionality of a dataset. This is a hazier concept, but can also be determined from the spectrum. Basically, if the spectrum contains a few large singular values and a number of tiny ones, then one can probably just harvest the largest singular values and discard the rest. Sometimes the rest of the spectrum is not tiny, but there's a large gap between the head and the tail values. That would also be a reasonable cutoff. This method is requires visual inspection of the spectrum and hence cannot be performed as part of an automated pipeline.

One key criticism of PCA is that the transformation is fairly complex, and the results are therefore hard to interpret. The principal components and the projected vectors are real-valued and could be positive or negative. The principal components are essentially linear combinations of the (centered) rows, and the projection values are linear combinations of the columns. In a stock returns application, for instance, each factor is a linear combination of time slices of stock returns. What does that mean? It is hard to express a human-understandable reason for the learned factors. Therefore, it is hard for analysts to trust the results. If you can't explain why you should be putting billions of other people's money into particular stocks, you probably won't choose to use that model.

PCA is computationally expensive. It relies on SVD, which is an expensive procedure. To compute the full SVD of a matrix takes $O(nd^2 + d^3)$ operations (Golub and Van Loan, 2012), assuming $n \geq d$—i.e., there are more data points than features. Even if we only want *k* principal components, computing the truncated SVD (the *k* largest singular values and vectors) still takes $O((n+d)^2\, k) = O(n^2 k)$ operations. This is prohibitive when there are a large number of data points or features.

It is difficult to perform PCA in a streaming fashion, in batch updates, or from a sample of the full data. Streaming computation of the SVD, updating the SVD, and computing the SVD from a subsample are all difficult research problems. Algorithms exist, but at the cost of reduced accuracy. One implication is that one should expect lower representational accuracy when projecting test data onto principal components found in the training set. As the distribution of the data changes, one would have to recompute the principal components in the current dataset.

Lastly, it is best not to apply PCA to raw counts (word counts, music play counts, movie viewing counts, etc.). The reason for this is that such counts often contain large outliers. (The probability is pretty high that there is a fan out there who watched *The Lord of the Rings* 314,582 times, which dwarfs the rest of the counts.) As we know, PCA looks for linear correlations within the features. Correlation and variance statistics are very sensitive to large outliers; a single large number could change the statistics a lot. So, it is a good idea to first trim the data of large values ("Frequency-Based Filtering" on page 48), or apply a scaling transform like tf-idf (Chapter 4) or the log transform ("Log Transformation" on page 15).

Use Cases

PCA reduces feature space dimensionality by looking for linear correlation patterns between features. Since it involves the SVD, PCA is expensive to compute for more than a few thousand features. But for small numbers of real-valued features, it is very much worth trying.

PCA transformation discards information from the data. Thus, the downstream model may be cheaper to train, but less accurate. On the MNIST dataset, some have observed that using reduced-dimensionality data from PCA results in less accurate classification models. In these cases, there is both an upside and a downside to using PCA.

One of the coolest applications of PCA is in anomaly detection of time series. Lakhina et al. (2004) used PCA to detect and diagnose anomalies in internet traffic. They focused on volume anomalies, i.e., when there is a surge or a dip in the amount of traffic going from one network region to another. These sudden changes may be indicative of a misconfigured network or coordinated denial-of-service attacks. Either way, knowing when and where such changes occur is valuable to internet operators.

Since there is so much total traffic over the internet, isolated surges in small regions are hard to detect. A relatively small set of backbone links handle much of the traffic. Their key insight is that volume anomalies affect multiple links at the same time (because network packets need to hop through multiple nodes to reach their destination). Treat each of the links as a feature, and the amount of traffic at each time step as the measurement. A data point is a time slice of traffic measurements across all links on the network. The principal components of this matrix indicate the overall traffic trends on the network. The rest of the components represent the residual signal, which contains the anomalies.

PCA is also often used in financial modeling. In those use cases, it works as a type of *factor analysis*, a term that describes a family of statistical methods that aim to describe observed variability in data using a small number of unobserved factors. In factor analysis applications, the goal is to find the explanatory components, not the transformed data.

Financial quantities like stock returns are often correlated with each other. Stocks may move up and down at the same time (positive correlation), or move in opposite directions (negative correlation). In order to balance volatility and reduce risk, an investment portfolio needs a diverse set of stocks that are not correlated with each other. (Don't put all your eggs in one basket if that basket is going to sink.) Finding strong correlation patterns is helpful for deciding on an investment strategy.

Stock correlation patterns can be industry-wide. For example, tech stocks may go up and down together, while airline stocks tend to go down when oil prices are high. But industry may not be the best way to explain the outcome. Analysts also look for unexpected correlations in observed statistics. In particular, the *statistical factor model* (Connor, 1995) runs PCA on the matrix of time series of individual stock returns to find commonly covarying stocks. In this use case, the end goal is the principal components themselves, not the transformed data.

ZCA is useful as a preprocessing step when learning from images. In natural images, adjacent pixels often have similar colors. ZCA whitening can remove this correlation, which allows subsequent modeling efforts to focus on more interesting image structures. Krizhevsky's (2009) thesis on "Learning Multiple Layers of Features from Images" contains nice examples that illustrate the effect of ZCA whitening on natural images.

Many deep learning models use PCA or ZCA as a preprocessing step, though it is not always necessary. In "Factored 3-Way Restricted Boltzmann Machines for Modeling Natural Images", Ranzato et al. (2010) remark, "Whitening is not necessary but speeds up the convergence of the algorithm." In "An Analysis of Single-Layer Networks in Unsupervised Feature Learning", Coates et al. (2011) find that ZCA whitening is helpful for some models, but not all. (Note the models in this paper are unsupervised feature learning models, so ZCA is used as a feature engineering method for other feature engineering methods. Stacking and chaining of methods is common in machine learning pipelines.)

Summary

This concluses the discussion of PCA. The two main things to remember about PCA are its mechanism (linear projection) and objective (to maximize the variance of projected data). The solution involves the eigen decomposition of the covariance matrix, which is closely related to the SVD of the data matrix. One can also remember PCA with the mental picture of squashing the data into a pancake that is as fluffy as possible.

PCA is an example of model-driven feature engineering. (One should immediately suspect that a model is lurking in the background whenever an objective function enters the scene.) The modeling assumption here is that variance adequately represents the information contained in the data. Equivalently, the model looks for linear correlations between features. This is used in several applications to reduce the correlation or find common factors in the input.

PCA is a well-known dimensionality reduction method. But it has its limitations, such as high computational cost and uninterpretable outcome. It is useful as a preprocessing step, especially when there are linear correlations between features.

When seen as a method for eliminating linear correlation, PCA is related to the concept of whitening. Its cousin, ZCA, whitens the data in an interpretable way, but does not reduce dimensionality.

Bibliography

Bell, Anthony J. and Terrence J. Sejnowski. "Edges Are the 'Independent Components' of Natural Scenes." *Advances in Neural Information Processing Systems* 9 (1996): 831–837.

Coates, Adam, Andrew Y. Ng, and Honglak Lee. "An Analysis of Single-Layer Networks in Unsupervised Feature Learning." *Proceedings of the 14th International conference on Artificial Intelligence and Statistics* (2011): 215–223.

Connor, Gregory. "The Three Types of Factor Models: A Comparison of Their Explanatory Power." *Financial Analysts Journal* 51:3 (1995) 42–46.

Golub, Gene H., and Charles F. Van Loan. *Matrix Computations*. 4th ed. Baltimore, MD: Johns Hopkins University Press, 2012.

Krizhevsky, Alex. "Learning Multiple Layers of Features from Tiny Images." MSc thesis, University of Toronto, 2009.

Lakhina, Anukool, Mark Crovella, and Christophe Diot. "Diagnosing Network-wide Traffic Anomalies." *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (2004): 219–230.

Ranzato, Marc'Aurelio, Alex Krizhevsky, and Geoffrey E. Hinton. "Factored 3-Way Restricted Boltzmann Machines for Modeling Natural Images." *Proceedings of the 13th International Conference on Artificial Intelligence and Statistics* (2010): 621–628.