PCA for Dimenssion Reduction

Principal component analysis can be broken down into five steps. It uses standardization, covariance, eigenvectors and eigenvalues

Definitions

1. Covariance
2. Correlation
3. Eigen vector
4. Eigen values

HOW DO YOU DO A PRINCIPAL COMPONENT ANALYSIS?

1. Standardize the range of continuous initial variables, mean centric.
2. Compute the covariance matrix to identify correlations.
3. Compute the eigenvectors and eigenvalues of the covariance matrix to identify the principal components.
4. Create a feature vector to decide which principal components to keep.
5. Recast the data along the principal components axes.

What Is Principal Component Analysis?

Principal component analysis, or PCA, is a dimensionality reduction method that is often used to reduce the dimensionality of large data sets, by transforming a large set of variables into a smaller one that still contains most of the information in the large set.

Reducing the number of variables of a data set naturally comes at the expense of accuracy, but the trick in dimensionality reduction is to trade a little accuracy for simplicity. Because smaller data sets are easier to explore and visualize and make analyzing data points much easier and faster for machine learning algorithms without extraneous variables to process.

The idea of PCA is simple — **reduce the number of variables of a data set, while preserving as much information as possible.**

Step-by-Step Explanation of PCA

STEP 1: STANDARDIZATION

The aim of this step is to standardize the range of the continuous initial variables so that each one of them contributes equally to the analysis.

More specifically, the reason why it is critical to perform standardization prior to PCA, is that the latter is quite sensitive regarding the variances of the initial variables. That is, if there are large differences between the ranges of initial variables, those variables with larger ranges will dominate over those with small ranges (for example, a variable that ranges between 0 and 100 will dominate over a variable that ranges between 0 and 1), which will lead to biased results. So, transforming the data to comparable scales can prevent this problem.

Mathematically, this can be done by subtracting the mean and dividing by the standard deviation for each value of each variable.

$$z = \frac{value - mean}{standard\ deviation}$$

Once the standardization is done, all the variables will be transformed to the same scale.

STEP 2: COVARIANCE MATRIX COMPUTATION

The aim of this step is to understand how the variables of the input data set are varying from the mean with respect to each other, or in other words, to see if there is any relationship between them. Because sometimes, variables are highly correlated in such a way that they contain redundant information. So, in order to identify these correlations, we compute the covariance matrix.

The covariance matrix is a *p* × *p* symmetric matrix (where *p* is the number of dimensions) that has as entries the covariances associated with all possible pairs of the initial variables. For example, for a 3-dimensional data set with 3 variables *x*, *y*, and *z*, the covariance matrix is a 3×3 data matrix of this from:

$$\begin{bmatrix} Cov(x,x) & Cov(x,y) & Cov(x,z) \\ Cov(y,x) & Cov(y,y) & Cov(y,z) \\ Cov(z,x) & Cov(z,y) & Cov(z,z) \end{bmatrix}$$

Covariance Matrix for 3-Dimensional Data.

Since the covariance of a variable with itself is its variance (Cov(a,a)=Var(a)), in the main diagonal (Top left to bottom right) we actually have the variances of each initial variable. And since the covariance is commutative (Cov(a,b)=Cov(b,a)), the entries of the covariance matrix are symmetric with respect to the main diagonal, which means that the upper and the lower triangular portions are equal.

What do the covariances that we have as entries of the matrix tell us about the correlations between the variables?

It's actually the sign of the covariance that matters:

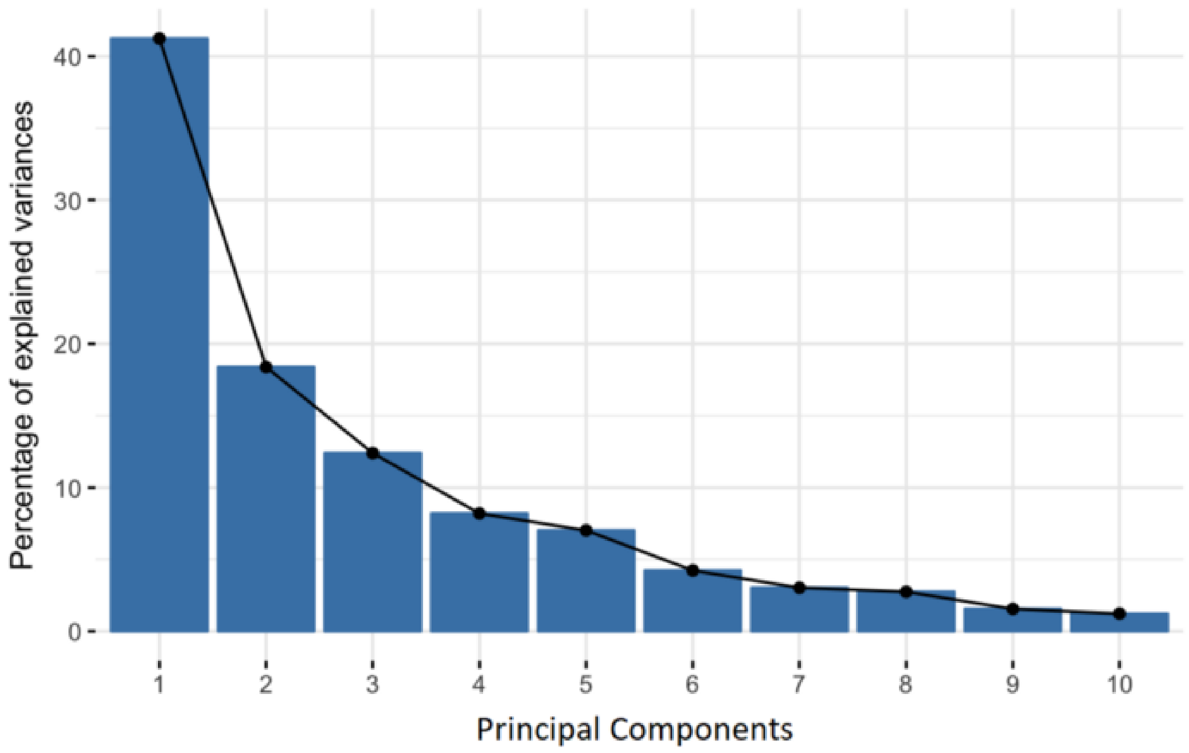
- If positive then: the two variables increase or decrease together (correlated)
- If negative then: one increases when the other decreases (Inversely correlated)

Now that we know that the covariance matrix is not more than a table that summarizes the correlations between all the possible pairs of variables, let's move to the next step.

STEP 3: COMPUTE THE EIGENVECTORS AND EIGENVALUES OF THE COVARIANCE MATRIX TO IDENTIFY THE PRINCIPAL COMPONENTS

Eigenvectors and eigenvalues are the linear algebra concepts that we need to compute from the covariance matrix in order to determine the ***principal components*** of the data. Before getting to the explanation of these concepts, let's first understand what do we mean by principal components.

Principal components are new variables that are constructed as linear combinations or mixtures of the initial variables. These combinations are done in such a way that the new variables (i.e., principal components) are uncorrelated and most of the information within the initial variables is squeezed or compressed into the first components. So, the idea is 10-dimensional data gives you 10 principal components, but PCA tries to put maximum possible information in the first component, then maximum remaining information in the second and so on, until having something like shown in the scree plot below.



Percentage of Variance (Information) for each by PC.

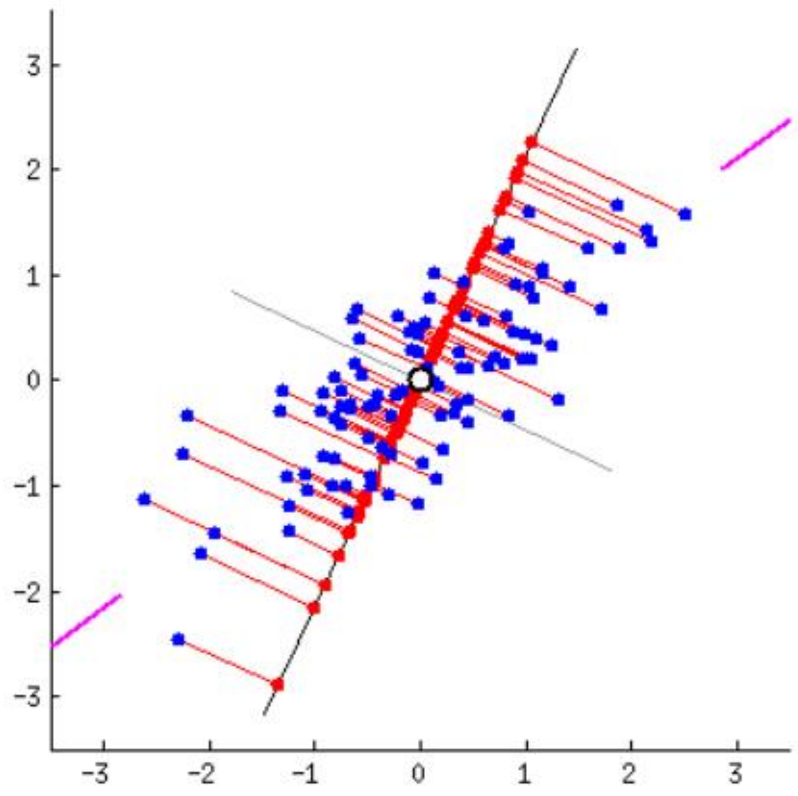
Organizing information in principal components this way, will allow you to reduce dimensionality without losing much information, and this by discarding the components with low information and considering the remaining components as your new variables.

An important thing to realize here is that the principal components are less interpretable and don’t have any real meaning since they are constructed as linear combinations of the initial variables.

Geometrically speaking, principal components represent the directions of the data that explain a **maximal amount of variance**, that is to say, the lines that capture most information of the data. The relationship between variance and information here, is that, the larger the variance carried by a line, the larger the dispersion of the data points along it, and the larger the dispersion along a line, the more information it has. To put all this simply, just think of principal components as new axes that provide the best angle to see and evaluate the data, so that the differences between the observations are better visible.

How PCA Constructs the Principal Components

As there are as many principal components as there are variables in the data, principal components are constructed in such a manner that the first principal component accounts for the **largest possible variance** in the data set. For example, let’s assume that the scatter plot of our data set is as shown below, the first principal component is approximately the line that matches the purple marks because it goes through the origin and it’s the line in which the projection of the points (red dots) is the most spread out. Or mathematically speaking, it’s the line that maximizes the variance (the average of the squared distances from the projected points (red dots) to the origin).



The second principal component is calculated in the same way, with the condition that it is uncorrelated with (i.e., perpendicular to) the first principal component and that it accounts for the next highest variance.

This continues until a total of p principal components have been calculated, equal to the original number of variables.

Now that we understand what we mean by principal components, let's go back to eigenvectors and eigenvalues. What you first need to know about them is that they always come in pairs, so that every eigenvector has an eigenvalue. And their number is equal to the number of dimensions of the data. For example, for a 3-dimensional data set, there are 3 variables, therefore there are 3 eigenvectors with 3 corresponding eigenvalues.

Without further ado, it is eigenvectors and eigenvalues who are behind all the magic explained above, because the eigenvectors of the Covariance matrix are actually *the directions of the axes where there is the most* *variance* (most information) and that we call Principal Components. And eigenvalues are simply the coefficients attached to eigenvectors, which give the *amount of variance carried in each Principal Component*.

By ranking your eigenvectors in order of their eigenvalues, highest to lowest, you get the principal components in order of significance.

Principal Component Analysis Example:

Let's suppose that our data set is 2-dimensional with 2 variables ***x,y*** and that the eigenvectors and eigenvalues of the covariance matrix are as follows:

$$v1 = \begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix} \qquad \lambda_1 = 1.284028$$

$$v2 = \begin{bmatrix} -0.7351785 \\ 0.6778736 \end{bmatrix} \qquad \lambda_2 = 0.04908323$$

If we rank the eigenvalues in descending order, we get λ1>λ2, which means that the eigenvector that corresponds to the first principal component (PC1) is *v1* and the one that corresponds to the second principal component (PC2) is *v2*.

After having the principal components, to compute the percentage of variance (information) accounted for by each component, we divide the eigenvalue of each component by the sum of eigenvalues. If we apply this on the example above, we find that PC1 and PC2 carry respectively 96 percent and 4 percent of the variance of the data.

STEP 4: FEATURE VECTOR

As we saw in the previous step, computing the eigenvectors and ordering them by their eigenvalues in descending order, allow us to find the principal components in order of significance. In this step, what we do is, to choose whether to keep all these components or discard those of lesser significance (of low eigenvalues), and form with the remaining ones a matrix of vectors that we call *Feature vector*.

So, the feature vector is simply a matrix that has as columns the eigenvectors of the components that we decide to keep. This makes it the first step towards dimensionality reduction, because if we choose to keep only ***p*** eigenvectors (components) out of ***n***, the final data set will have only ***p*** dimensions.

Principal Component Analysis Example:

Continuing with the example from the previous step, we can either form a feature vector with both of the eigenvectors *v*1 and *v*2:

$$\begin{bmatrix} 0.6778736 & -0.7351785 \\ 0.7351785 & 0.6778736 \end{bmatrix}$$

Or discard the eigenvector *v*2, which is the one of lesser significance, and form a feature vector with *v*1 only:

$$\begin{bmatrix} 0.6778736 \\ 0.7351785 \end{bmatrix}$$

Discarding the eigenvector *v2* will reduce dimensionality by 1, and will consequently cause a loss of information in the final data set. But given that *v*2 was carrying only 4 percent of the information, the loss will be therefore not important and we will still have 96 percent of the information that is carried by *v*1.

So, as we saw in the example, it’s up to you to choose whether to keep all the components or discard the ones of lesser significance, depending on what you are looking for. Because if you just want to describe your data in terms of new variables (principal components) that are uncorrelated without seeking to reduce dimensionality, leaving out lesser significant components is not needed.

STEP 5: RECAST THE DATA ALONG THE PRINCIPAL COMPONENTS AXES

In the previous steps, apart from standardization, you do not make any changes on the data, you just select the principal components and form the feature vector, but the input data set remains always in terms of the original axes (i.e, in terms of the initial variables).

In this step, which is the last one, the aim is to use the feature vector formed using the eigenvectors of the covariance matrix, to reorient the data from the original axes to the ones represented by the principal components (hence the name Principal Components Analysis). This can be done by multiplying the transpose of the original data set by the transpose of the feature vector.

$$FinalDataSet = FeatureVector^T * StandardizedOriginalDataSet^T$$

While you can speed up the fitting of a machine learning algorithm by changing the optimization algorithm, a more common way to speed up the algorithm is to use principal component analysis (PCA). If your learning algorithm is too slow because the input dimension is too high, then using PCA to speed it up can be a reasonable choice. This is probably the most common application of PCA. Another common application of PCA is for data visualization.

WHAT IS A PCA?

Principal component analysis (PCA) is a method of reducing the dimensionality of data and is used to improve data visualization and speed up machine learning model training.

To understand the value of using PCA for data visualization, the first part of this tutorial post goes over a basic visualization of the Iris data set after applying PCA. The second part, explores how to use PCA to speed up a machine learning algorithm (logistic regression) on the Modified National Institute of Standards and Technology (MNIST) data set.

PCA for Data Visualization

For a lot of machine learning applications, it helps to visualize your data. Visualizing two- or three-dimensional data is not that challenging. However, even the Iris data set used in this part of the tutorial is four-dimensional. You can use PCA to reduce that four-dimensional data into two or three dimensions so that you can plot, and hopefully, understand the data better.

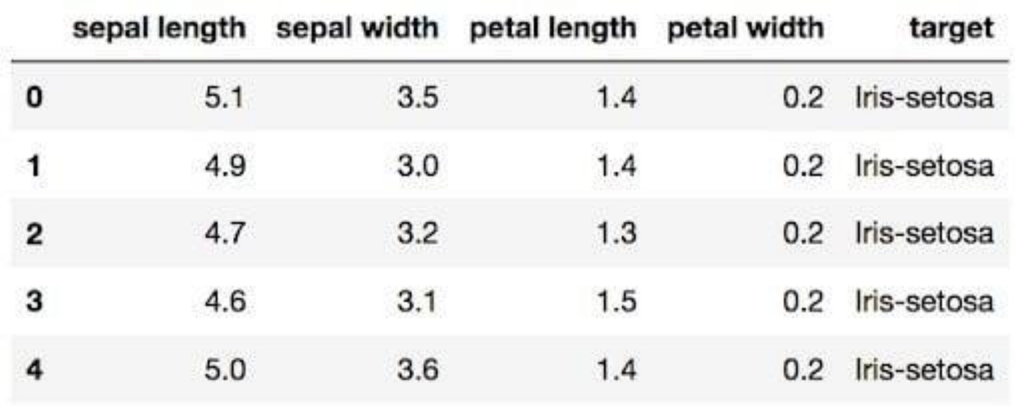
STEP 1: LOAD THE IRIS DATA SET

The iris data set comes with scikit-learn and doesn't require you to download any files from some external websites. The code below will load the Iris data set.

```
import pandas as pd
```

```
url = "https://archive.ics.uci.edu/ml/machine-learning-
databases/iris/iris.data"
```

```
# load dataset into Pandas DataFrame
df = pd.read_csv(url, names=['sepal length','sepal width','petal
length','petal width','target'])
```



	sepal length	sepal width	petal length	petal width	target
0	5.1	3.5	1.4	0.2	Iris-setosa
1	4.9	3.0	1.4	0.2	Iris-setosa
2	4.7	3.2	1.3	0.2	Iris-setosa
3	4.6	3.1	1.5	0.2	Iris-setosa
4	5.0	3.6	1.4	0.2	Iris-setosa

Original Pandas df (features and target). | Image: Michael Galarnyk

STEP 2: STANDARDIZE THE DATA

PCA is affected by scale, so you need to scale the features in your data before applying PCA. Use `StandardScaler` to help you standardize the data set's features onto unit scale (`mean = 0` and `variance = 1`), which is a requirement for the optimal performance of many machine learning algorithms. If you don't scale your data, it can have a negative effect on your algorithm.

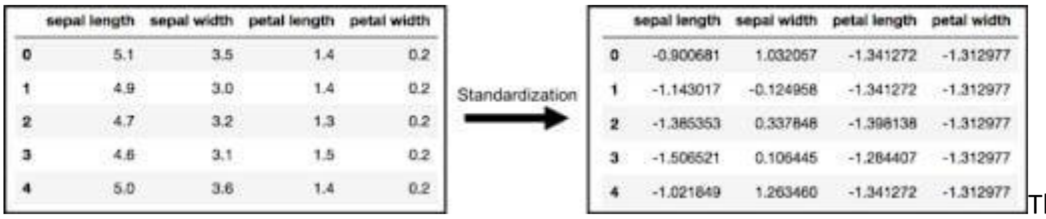
```
from sklearn.preprocessing import StandardScaler
```

```
features = ['sepal length', 'sepal width', 'petal length', 'petal
width']
```

```
# Separating out the features
x = df.loc[:, features].values
```

```
# Separating out the target
y = df.loc[:,['target']].values
```

```
# Standardizing the features
x = StandardScaler().fit_transform(x)
```


Standardization

	sepal length	sepal width	petal length	petal width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

	sepal length	sepal width	petal length	petal width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

The array x (visualized by a pandas dataframe) before and after standardization. | Image: Michael Galarnyk.

STEP 3: PCA PROJECTION TO 2D

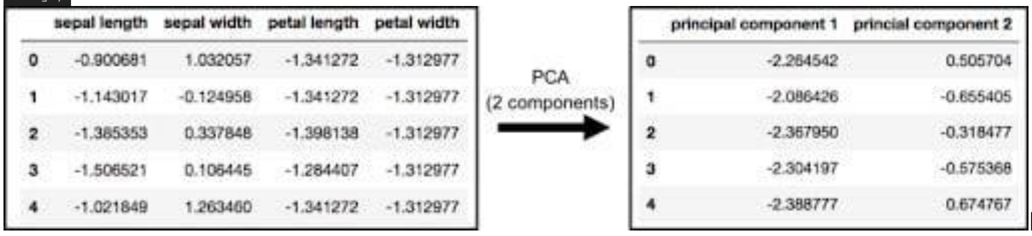
The original data has four columns (sepal length, sepal width, petal length and petal width). In this section, the code projects the original data, which is four-dimensional, into two dimensions. After dimensionality reduction, there usually isn't a particular meaning assigned to each principal component. The new components are just the two main dimensions of variation.

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components=2)
```

```
principalComponents = pca.fit_transform(x)
```

```
principalDf = pd.DataFrame(data = principalComponents
             , columns = ['principal component 1', 'principal component 2'])
```


PCA
(2 components)

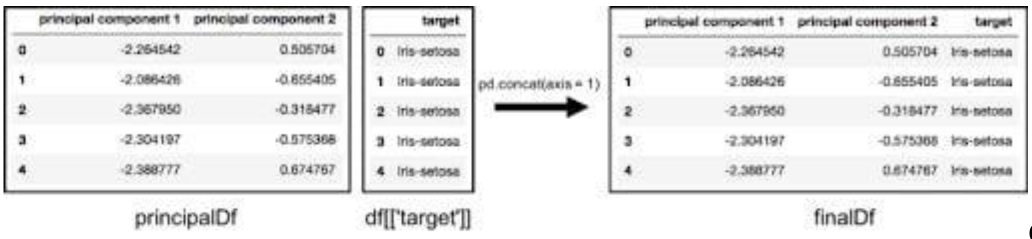
	sepal length	sepal width	petal length	petal width
0	-0.900681	1.032057	-1.341272	-1.312977
1	-1.143017	-0.124958	-1.341272	-1.312977
2	-1.385353	0.337848	-1.398138	-1.312977
3	-1.506521	0.106445	-1.284407	-1.312977
4	-1.021849	1.263460	-1.341272	-1.312977

	principal component 1	princial component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

PCA and keeping the top two principal components

```
finalDf = pd.concat([principalDf, df[['target']]], axis = 1)
```

Concatenating DataFrame along `axis = 1`. `finalDf` is the final DataFrame before plotting the data.



	principal component 1	principal component 2
0	-2.264542	0.505704
1	-2.086426	-0.655405
2	-2.367950	-0.318477
3	-2.304197	-0.575368
4	-2.388777	0.674767

principalDf

	target
0	Iris-setosa
1	Iris-setosa
2	Iris-setosa
3	Iris-setosa
4	Iris-setosa

df[['target']]

	principal component 1	principal component 2	target
0	-2.264542	0.505704	Iris-setosa
1	-2.086426	-0.655405	Iris-setosa
2	-2.367950	-0.318477	Iris-setosa
3	-2.304197	-0.575368	Iris-setosa
4	-2.388777	0.674767	Iris-setosa

finalDf

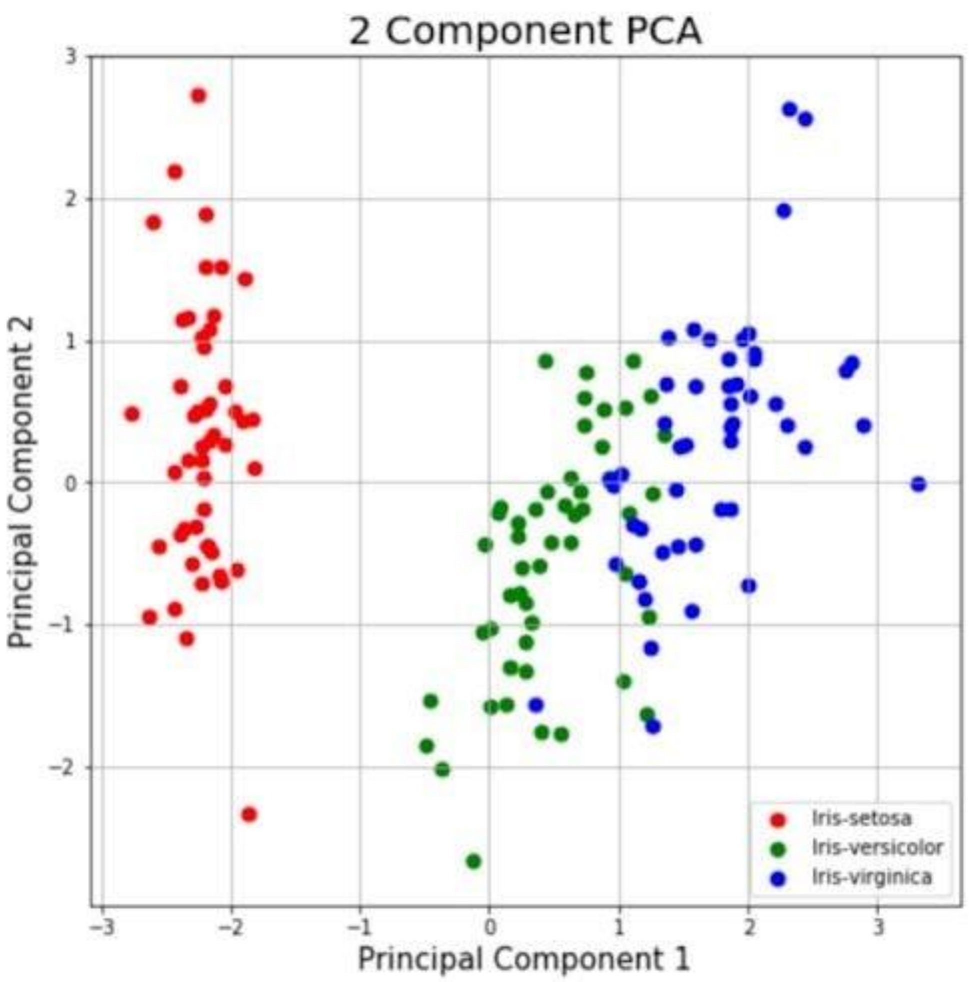
Concatenating DataFrames along columns to make finalDf before graphing. | Image: Michael Galarnyk

STEP 4: VISUALIZE 2D PROJECTION

This section is just plotting two-dimensional data. Notice on the graph below that the classes seem well separated from each other.

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(1,1,1)
ax.set_xlabel('Principal Component 1', fontsize = 15)
ax.set_ylabel('Principal Component 2', fontsize = 15)
ax.set_title('2 component PCA', fontsize = 20)

targets = ['Iris-setosa', 'Iris-versicolor', 'Iris-virginica']
colors = ['r', 'g', 'b']
for target, color in zip(targets,colors):
    indicesToKeep = finalDf['target'] == target
    ax.scatter(finalDf.loc[indicesToKeep, 'principal component 1']
               , finalDf.loc[indicesToKeep, 'principal component 2']
               , c = color
               , s = 50)
ax.legend(targets)
ax.grid()
```



A two component PCA graph. | Image: Michael Galarnyk

EXPLAINED VARIANCE

The explained variance tells you how much information (variance) can be attributed to each of the principal components. This is important because while you can convert four-dimensional space to a two-dimensional space, you lose some of the variance (information) when you do this. By using the attribute `explained_variance_ratio_`, you can see that the first principal component contains 72.77 percent of the variance, and the second principal component contains 23.03 percent of the variance. Together, the two components contain 95.80 percent of the information.

```
pca.explained_variance_ratio_
```

PCA to Speed-Up Machine Learning Algorithms

While there are other ways to speed up machine learning algorithms, one less commonly known way is to use PCA. For this section, we aren't using the Iris data set, as it only has 150 rows and four feature columns. The MNIST database of handwritten digits is more suitable, as it has 784 feature columns (784 dimensions), a training set of 60,000 examples and a test set of 10,000 examples.

STEP 1: DOWNLOAD AND LOAD THE DATA

You can also add a `data_home` parameter to `fetch_mldata` to change where you download the data.

```
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

The images that you downloaded are contained in `mnist.data` and has a shape of (70000, 784) meaning there are 70,000 images with 784 dimensions (784 features).

The labels (the integers 0–9) are contained in `mnist.target`. The features are 784 dimensional (28 x 28 images), and the labels are numbers from 0–9.

STEP 2: SPLIT DATA INTO TRAINING AND TEST SETS

The code below performs a train test split which puts 6/7th of the data into a training set and 1/7 of the data into a test set.

```
from sklearn.model_selection import train_test_split
```

```
# test_size: what proportion of original data is used for test set
train_img, test_img, train_lbl, test_lbl = train_test_split(
mnist.data, mnist.target, test_size=1/7.0, random_state=0)
```

STEP 3: STANDARDIZE THE DATA

The text in this paragraph is almost an exact copy of what was written earlier. PCA is affected by scale, so you need to scale the features in the data before applying PCA. You can transform the data onto unit scale (`mean = 0` and `variance = 1`), which is a requirement for the optimal performance of many machine learning algorithms. `StandardScaler` helps standardize the data set's features. You fit on the training set and transform on the training and test set.

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
```

```
# Fit on training set only.
scaler.fit(train_img)
```

```
# Apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

STEP 4: IMPORT AND APPLY PCA

Notice the code below has .95 for the number of components parameter. It means that scikit-learn chooses the minimum number of principal components such that 95 percent of the variance is retained.

```
from sklearn.decomposition import PCA
```

```
# Make an instance of the Model
```

```
pca = PCA(.95)
```

Fit PCA on the training set. You are only fitting PCA on the training set.

```
pca.fit(train_img)
```

You can find out how many components PCA has after fitting the model using `pca.n_components_`. In this case, 95 percent of the variance amounts to 330 principal components.

STEP 5: APPLY THE MAPPING (TRANSFORM) TO THE TRAINING SET AND THE TEST SET.

```
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

STEP 6: APPLY LOGISTIC REGRESSION TO THE TRANSFORMED DATA

1. Import the model you want to use.

In sklearn, all machine learning models are implemented as Python classes.

```
from sklearn.linear_model import LogisticRegression
```

2. Make an instance of the model.

```
# all parameters not specified are set to their defaults
# default solver is incredibly slow which is why it was changed to
'lbfgs'
logisticRegr = LogisticRegression(solver = 'lbfgs')
```

3. Train the model on the data, storing the information learned from the data.

The model is learning the relationship between digits and labels.

```
logisticRegr.fit(train_img, train_lbl)
```

4. Predict the labels of new data (new images).

This part uses the information the model learned during the model training process. The code below predicts for one observation.

```
# Predict for One Observation (image)
logisticRegr.predict(test_img[0].reshape(1,-1))
```

The code below predicts for multiple observations at once.

```
# Predict for One Observation (image)
logisticRegr.predict(test_img[0:10])
```

STEP 7: MEASURING MODEL PERFORMANCE

While accuracy is not always the best metric for machine learning algorithms (precision, recall, F1 score, ROC curve, etc., would be better), it is used here for simplicity.

```
logisticRegr.score(test_img, test_lbl)
```

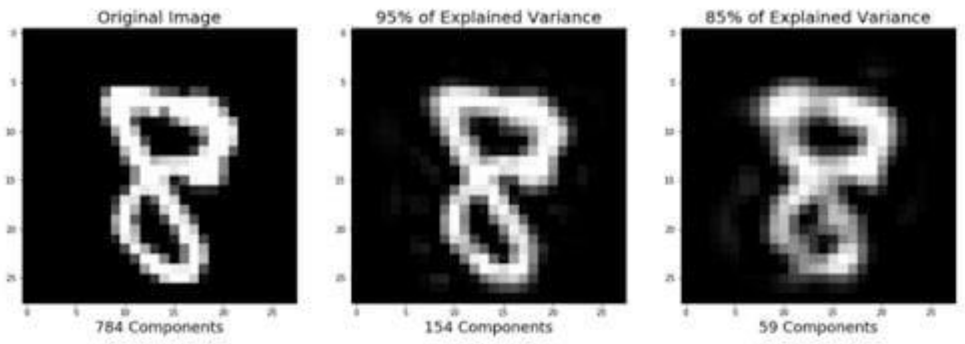
TESTING THE TIME TO FIT LOGISTIC REGRESSION AFTER PCA

The whole purpose of this section of the tutorial was to show that you can use PCA to speed up the fitting of machine learning algorithms. The table below shows how long it took to fit logistic regression on my MacBook after using PCA (retaining different amounts of variance each time).

Variance Retained	Number of Components	Time (seconds)	Accuracy
1.00	784	48.94	0.9158
0.99	541	34.69	0.9169
0.95	330	13.89	0.9200
0.90	236	10.56	0.9168
0.85	184	8.85	0.9156

Image Reconstruction From Compressed Representation

Earlier sections of this tutorial demonstrated how to use PCA to compress high dimensional data to lower dimensional data. But PCA can also take the compressed representation of the data (lower dimensional data) and return it to an approximation of the original high dimensional data. If you are interested, you can use this code to reproduce the image below.


Original Image
784 Components
95% of Explained Variance
154 Components
85% of Explained Variance
59 Components