Alvin
Chung
Student who enjoys analysing data, and trying to 'predict the future'.—Currently in Sydney
Jan 15 · 10 min read

Part 2

Supervised Machine Learning—Dimensional Reduction and Principal Component Analysis

Special thanks to a dear friend of mine, Katharine, for inspiring this series. May the lessons learned in this series guide even the warmest of hearts.

This article is part of a series. Check out Part 1 here.

Introduction

The motivation of this series is to enable anyone who is interested in the field of machine learning to be able to develop, understand and implement their own machine learning algorithms.

In machine learning problems there often involves tens of thousands of features for each training instance. This can be a problem as it makes our training extremely slow and prone to *overfitting (refer to overfitting section)*. This problem is commonly referred to as **the curse of dimensionality**.

Because of the issues associated with **the curse of dimensionality**, it is necessary to reduce the number of features/dimensions considerably to help increase our model's performance and enables us to arrive at an optimal solution for our machine learning model. Fortunately, in most real life problems, it is often possible to reduce the dimensions of our training set, without losing too much of the variance within our data.

For instance, some of our data points may be completely meaningless in explaining our desired target variable. As such, we may prefer to drop them from our analysis. Moreover, it is often that two data points may be highly correlated with each other; therefore by merging them into a single data point, you would not lose much information.

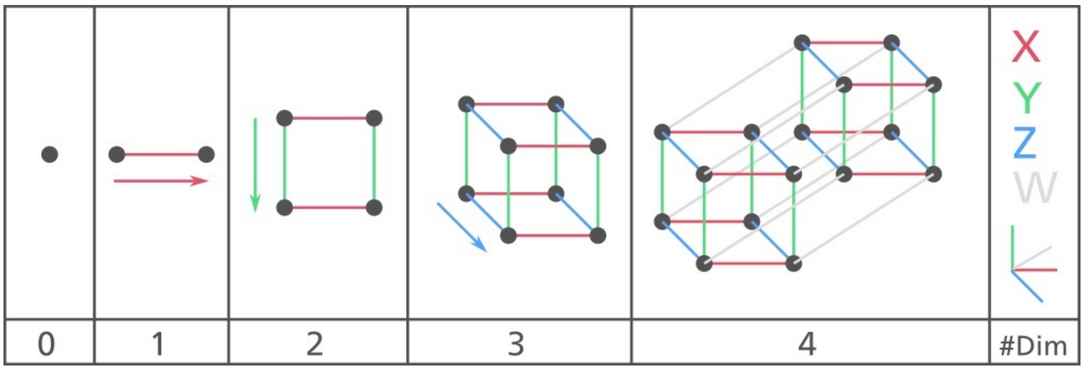
> *By reducing the dimensions of our training set, we can increase the speed of our training, and reduce our dataset down to two or three dimensions, making it easier to perform data visualisations (clustering, patterns).*

In this chapter, we will explore the issues associated with **the *curse of dimensionality***, the various methods in performing Dimensionality Reduction (***Projection & Manifold Learning***) and a Dimensional Reduction algorithm known as ***Principal Component Analysis (PCA)***.

. . .

The Curse of Dimensionality

A fool once said "help help! I can't seem to see anyone!", the wise one replied "It's cause you're stuck in 4-dimensions" Get it? Nor do I



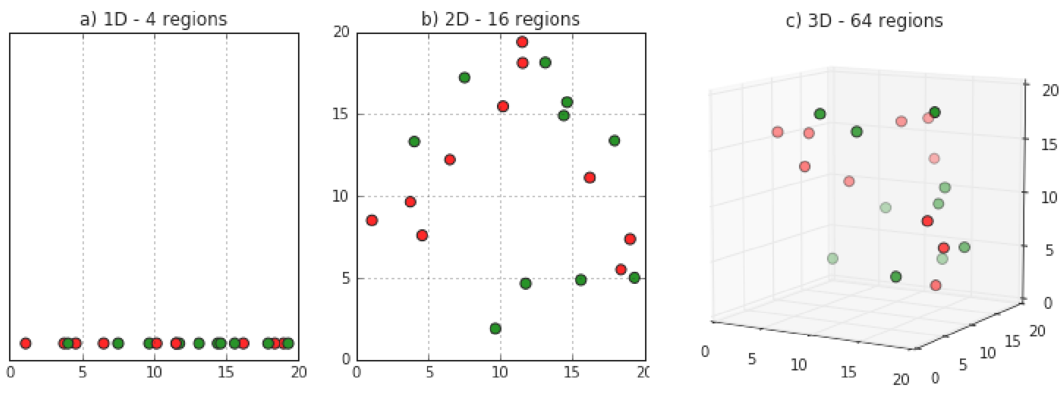
Source: Wikipedia

It turns out, when in high dimensional space, various phenomenas arise that do not occur in low-dimensional space.

> *Most noticeable, as the dimensions of our data set increases,*

the volume of the space increases so fast that the available data becomes extremely sparse.

As noted by Aurelien Geron, if you were to pick a random point in a unit space (1 x 1 square), it would only have about a 0.4% chance of being located less than 0.001 from the border (in other words, it is extremely unlikely that a random point will be 'extreme along any dimensions'). But in a 10,000 dimensional unit hyper cube (a 1 x 1 x … x 1 cube, with ten thousand 1s), this probability is greater than 99.9999%.



Source: Clever Owl

To further highlight this issue, imagine if you picked two random points in a unit square, the distance between these two points is on average, roughly 0.52. If you were then asked to pick two random points, but this time in a unit 3D cube, the average distance would be roughly 0.66. And for a 1,000,000- dimensional hypercube, this distance would be roughly 408.25—Credits to Aurlien Geron. As such, high dimensional data sets are often at risk of being extremely sparse, making our model prone towards overfitting, and much less reliable than in lower dimensions. Thereby, negatively impacting the performance of our machine learning model.

In short, the more dimensions the training set has, the greater the risk of overfitting our model.

Note: One solution to the problem of *the curse of dimensionality* could be to increase the size of the training set such that it reaches a sufficient density of training instances. However, this solution is extremely impractical as the number of training instances required to reach a given density grows exponentially with the number of dimensions.

. . .

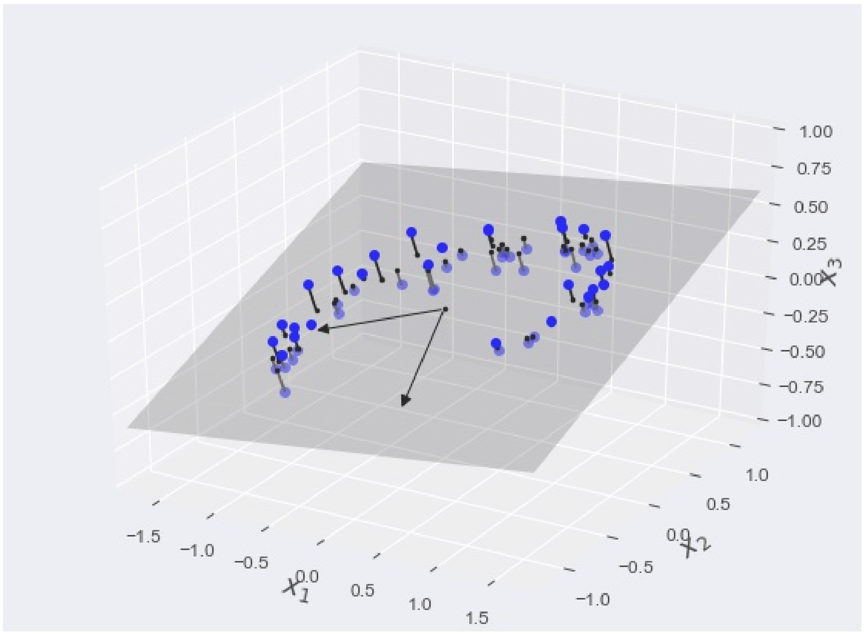
Methods to Dimensional Reduction

There are two primary approaches towards dimensional reduction, *Projection* and *Manifold Learning*. We will explore both methods in depth below.

Projection

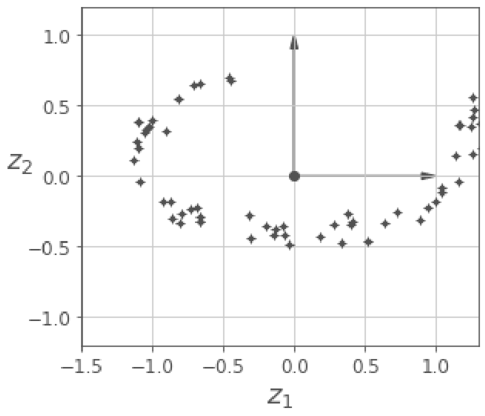
Grab a piece of paper and place it in between two objects. That's it. You've learned Projection!

Thankfully in most real life problems, our training instances are not separated out uniformly in all dimensions. Most features we observe follow some sort of pattern, which commonly results to our features being either constant or highly correlated. As such, all training instances can actually be mapped to a much lower-dimensional subspace of a high-dimension. Take for example, this three dimensional space.



Source: handson-ml

As you can see nearly all the training instances lie close to a plane: this plane is a lower-dimensional (2D)subspace of a high-dimensional (3D) space. Now if we were to ***project*** every training instance perpendicular onto this subspace (as seen from the small lines), we can get a new 2D data set. That's it! We have just reduced a higher-dimensional (3D) space into a lower 2-dimensional space. The projection of the axes would now correspond to the new features, commonly denoted as *z*1 and *z*2 (the coordinates of the projections of the plane).



Source: handson-ml

Projection

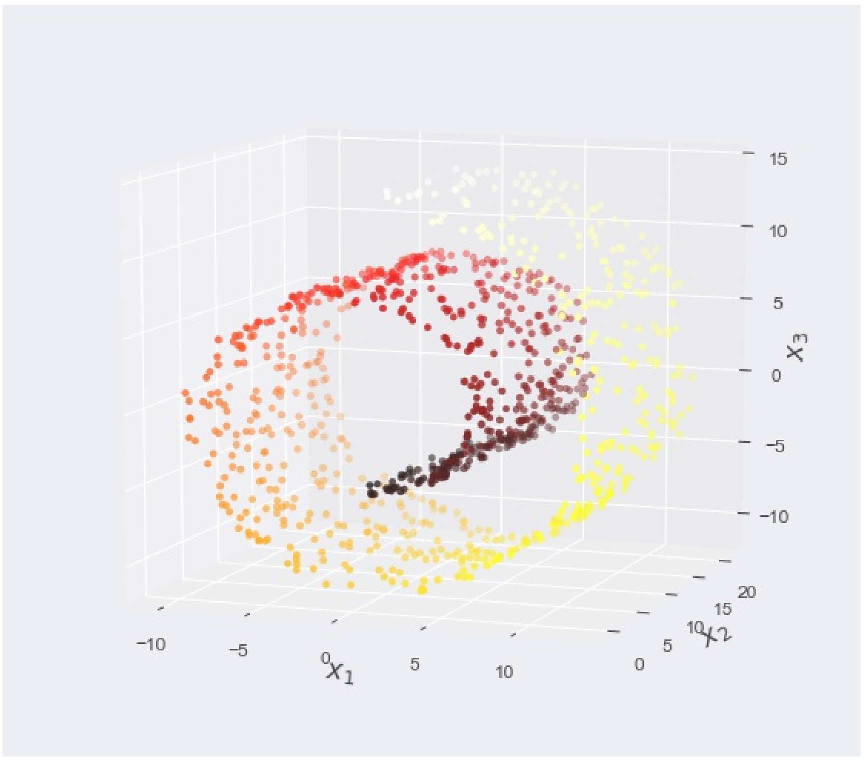
Original Dimensions --> [x1, x2, x3] (*3-Dimensions*)

Projection Dimensions --> [z1, z2] (*2-Dimensions*)

Note: This various depending on the dimensions of your data set. For our example, we have used a 3D data set and reduced the dimensions to 2D, as it is easier to visualise. However, if you have more dimensions in your training and after reduction, you have k-dimensions. Your coordinates would be denoted as (z1, z2...zn) for all n dimensions in k

Weakness

While, the projection method is simple and effective in performing dimensionality reduction. If we have overlapping instances in our dimensional subspace, simply projecting our instances towards a hyperplane may result in the loss of important information. Take for instance, this Swiss Roll imported from Sci-kit Learn.

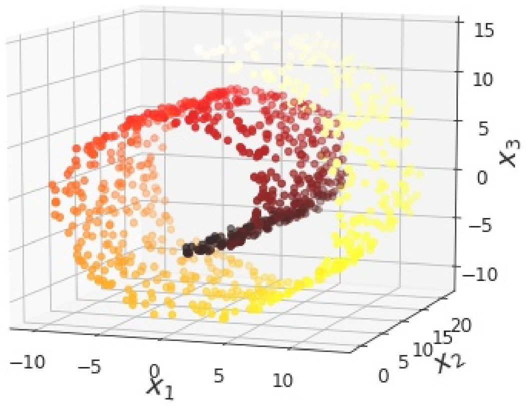


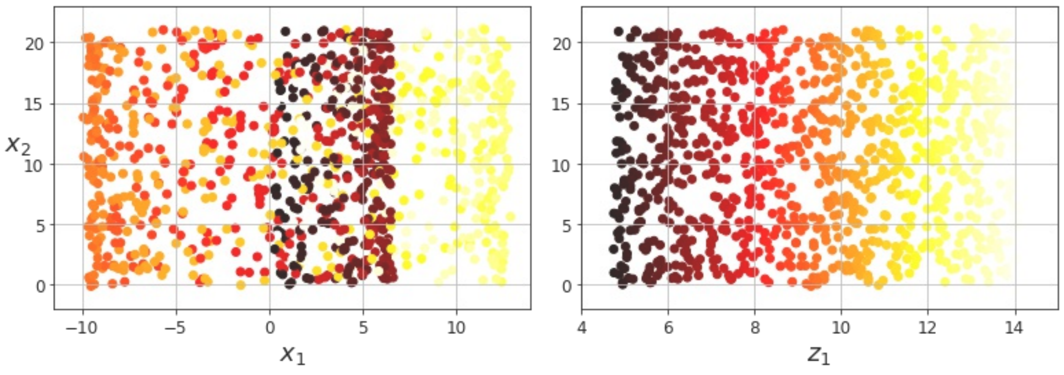
If we just projected a 2D plane onto this 3D space (*e.g. by dropping x3*), it would simply squash the various layers together and lose all the information we wanted to capture. Instead, what we want is to unroll the Swiss roll, to obtain a 2D data set, were we able to capture nearly all the variance without much loss of information. This technique is called Manifold Learning.

Manifold Learning

A man once said "How do I travel a city in a day?" The director replied "By hiring a CGI personal, and getting him to unroll the city"—Thus Manifold Learning was born.—haha……

Simply put, Manifold Learning is when we bend and twist a high-dimensional space such that it can be mapped to a lower-dimensional space, where the high-dimensional space can locally be resembled in a lower-dimensional space. This technique is often the most popular method employed in Machine Learning, as it enables the algorithm to find the optimal manifold in reducing the dimensionality of our data set. For instance, in our Swiss Roll example, the objective of the algorithm would be to learn the optimal way to unfold our Swiss Roll in enabling us to capture as much of the information as possible, relative to it’s original dimensional space.





Source: handson-ml

Weakness

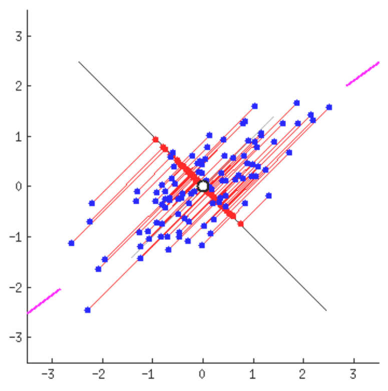
Where Manifold Learning fails to increase the accuracy of our model is when the task at hand is not simpler if expressed in a lower-dimensional subspace. Such that, if we were to reduce the dimensionality of our training set, the explanatory power of our data set diminishes relative to it’s original dimensions.

> *As a general rule, if you are given a data set which has more than 3 dimensions, it is often the case that dimensionality reduction algorithms will be beneficial in enhancing the performance of your model.*

. . .

Principal Component Analysis

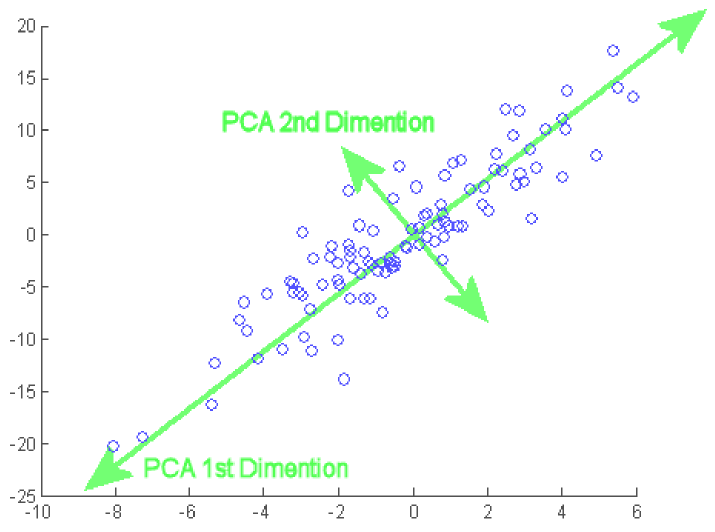
Principle Component Analysis **(PCA)** one of the most important algorithms in the field of Data Science and is by far the most popular dimensionality reduction method currently used today. The objective of Principle Component Analysis is simple, identify a used today. The objective of Principle Component Analysis is simple, identify a hyperplane that lies closest to the data points, and project the data onto it.



Source: leonardoaraujosantos

Preservation our Variance

When mapping our original data set in a lower-dimensional hyperplane, we first must determine the right hyperplane. In Principal Component Analysis, the hyperplane, in most cases is determined by the axis through our data set, which preserves the maximum amount of variance.



Source: prachimjoshi

We want to choose the axis that retains the maximum amount of variance in our data set, as it will most likely lose less information than if we were to consider other projections. In this case, the axis we choose is referred to as **Principal Components**, also known as **PCA1** as it captures the maximum amount of variance within our data set, with the second Principal Component, orthogonal to the first, as it accounts for the largest remaining variance in our dataset—denoted as **PCA2**.

Note: In real life we are more likely to work with higher dimensional data sets. As a result, we may need to include addition Principal Components in order to capture the more of the variance in our data set. Thereby, increasing the performance of our Machine Learning algorithm. However, for the purpose of visualisation, only necessary to take two Principle Components for visualisation; and in some cases three.

Data Visualisation: [PCA1, PCA2 and/or PCA3] 2 or 3 Dimensions

Machine Learning: It is preferred in most machine learning problems to capture at least 95% of the training set's variance. Thus, it is not necessary to stick with 2 or 3 Principal Components.

. . .

Principal Components

The are various techniques we can use to find our principle components. However, the most common method in finding our principle components is known as **Singular Value Decomposition (SVD)**. It turns out that if you perform a basic matrix factorisation technique known as Singular Value Decomposition (**SVD**), where you decompose the training set matrix X, into the dot product of three matrices, our third matrix V* actually contains all the principal components we are looking for.

$$M = U \cdot \Sigma \cdot V^*$$

Where:

M = m × n matrix whose entries come from a field, where either field consists of real numbers or complex numbers.
U = a *m* × *m* unitary matrix. (left singular vector)
Σ = *m* × *n* diagonal matrix with non-negative real numbers.
V = *n* × *n* unitary matrix . (right singular vector)
V* = conjugate transpose of the *n* × *n* unitary matrix.

For those whom are interested in the full mathematics in Singular Value Decomposition(SVD). Here is a good link

Implementation

It's extremely simple to perform a Principle Component Analysis on our data set.We simply invoke the function **PCA** from Sci-Kit Learn.

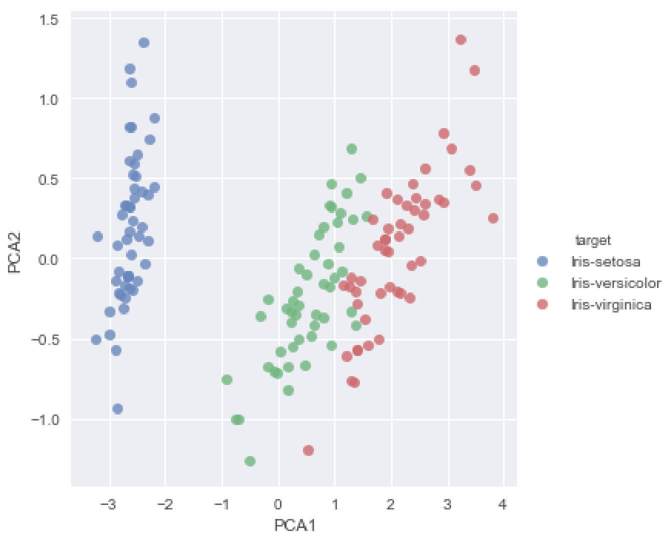
```
from sklearn.decomposition import PCA

pca = PCA(n_components = 2)
X2D = pca.fit(X).transform(X)
```

After fitting the dataset, we can transform our data set into a **Pandas DataFrame Object**, with the respective labels **(PCA1, PCA2).**

```
df = pd.DataFrame(X2D,columns = ['PCA1','PCA2'])
```

We can them proceed to plot our Principal Components onto a scatterplot,. For simplicity I have used a library known as **Seaborn** simplicity I have used a library known as **Seaborn**.



From the scatterplot, it is evident that we can see clustering in our principal components. This indicates that there is variance in our dataset that can explain our target variables. Most notably, if we wanted to examine the explained variance of each Principal Component, Sci-Kit Learn has a handy function called **explained_variance_ratio**, which gives us information on the explained variance of each principal component. Let's invoke it!

```
pca.explained_variance_ratio
array([0.82,0.15])
```

> *This tells us that 82% of the variance lies along the first axis,*
>
> *and 12% lies along the second axis,*
>
> *with the remaining variance captured in the other Principal Components.*

This works well, when we are looking to perform data visualisation tasks in our high dimensional dataset. However, when we are modelling, we often want to capture a certain proportion of the variance within our training set.

> *As a general rule: It is often preferred that we capture at least 95% of the variance within our training set.*

To achieve this, we simply change our n_components parameter from an integer to a float between 0.0 to 1.0, indicative of the ratio of variance we would like to capture:

```
pca = PCA(n_components = 0.95)
X = pca.fit_transform(X_train)
```

. . .

Congratulations! You've just covered all the

Congratulations! You've just covered all the essential principles of Dimensionality Reduction and Principal Component Analysis!!

> *Wow, That was a lot to learn for just one day! We've just covered some crucial concepts for Machine Learning. I hope you've gained a greater understanding on Dimensionality Reduction, and see the strength of it in your next adventure.*

Here's a brief summary of what we've covered:

- What **the Curse of Dimensionality** is
- Methods which we can use to **reduce the affects** of the **curse of dimensionality**. Notably: **Projection & Manifold Learning**
- The **Weaknesses** of various **Dimensionality Reduction methods**
- An introduction to **Principal Component Analysis** and **how it works**
- How to **Implement Principle Component Analysis**

If you liked this article, please do and share it with your friends. Remember, you can clap up to 50 times—it really makes a big difference for me.

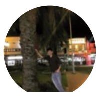
Thanks to Rebecca.

Machine Learning | Dimensionality Reduction | Data Science | Artificial Intelligence | Python

One clap, two clap, three clap, forty?
By clapping more or less, you can signal to us which stories really stand out.

174

1



Alvin Chung
Student who enjoys analysing data, and trying to 'predict the future'.—

Follow

Currently in Sydney

Hacker Noon

how hackers start their afternoons.

Follow

Never miss a story from **Hacker Noon**

GET UPDATES