Unsupervised Learning

Table of Contents

[Principal Component Analysis 3](#_Toc106495172)

[Principal Components 5](#_Toc106495173)

[3D Feature Spaces 6](#_Toc106495174)

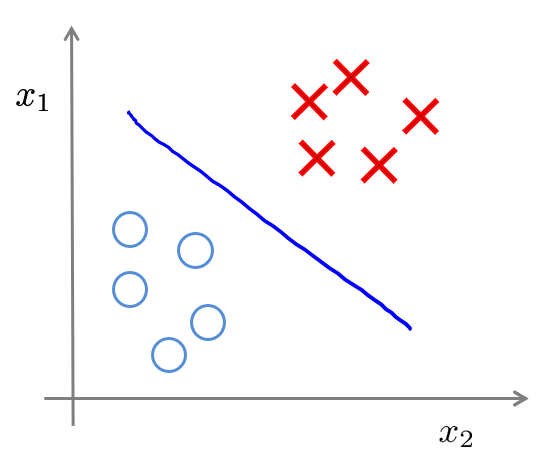
[Algorithm 7](#_Toc106495175)

[Data Reconstruction 12](#_Toc106495176)

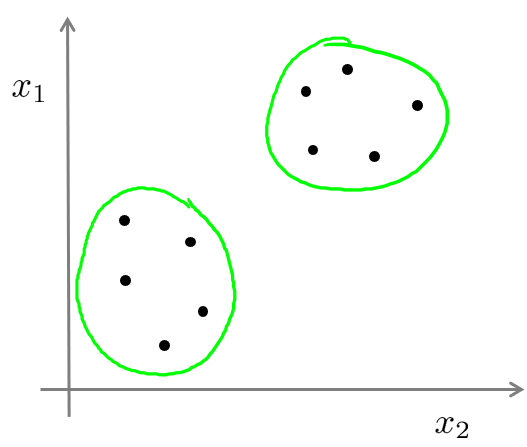
[Choosing the Number of Principle Components 13](#_Toc106495177)

[Applying PCA 14](#_Toc106495178)

In **supervised learning**, we had data points that were labelled to be in specific classes and our goal was to find the threshold to divide the feature space into sections based on those classes.



In **unsupervised learning**, the class labels are not provided. As a result, we end up just grouping the data points based on shared attributes. These groups are called **clusters**.



Essentially, this is what unsupervised learning does. It finds patterns in data that humans cannot identify without using any pre-existing labels and with minimal human supervision.

There are mainly two methods for unsupervised learning, Cluster Analysis and Principal Component Analysis.

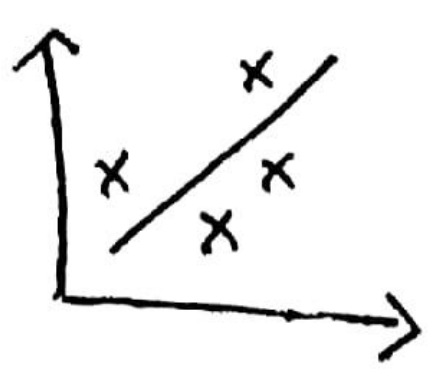
**Cluster Analysis** is essentially what we saw above, where the algorithm learns to group data points with shared attributes.

**Principal Component Analysis** (PCA) attempts to learn a new feature space which captures the characteristics of the original feature space but has fewer features.

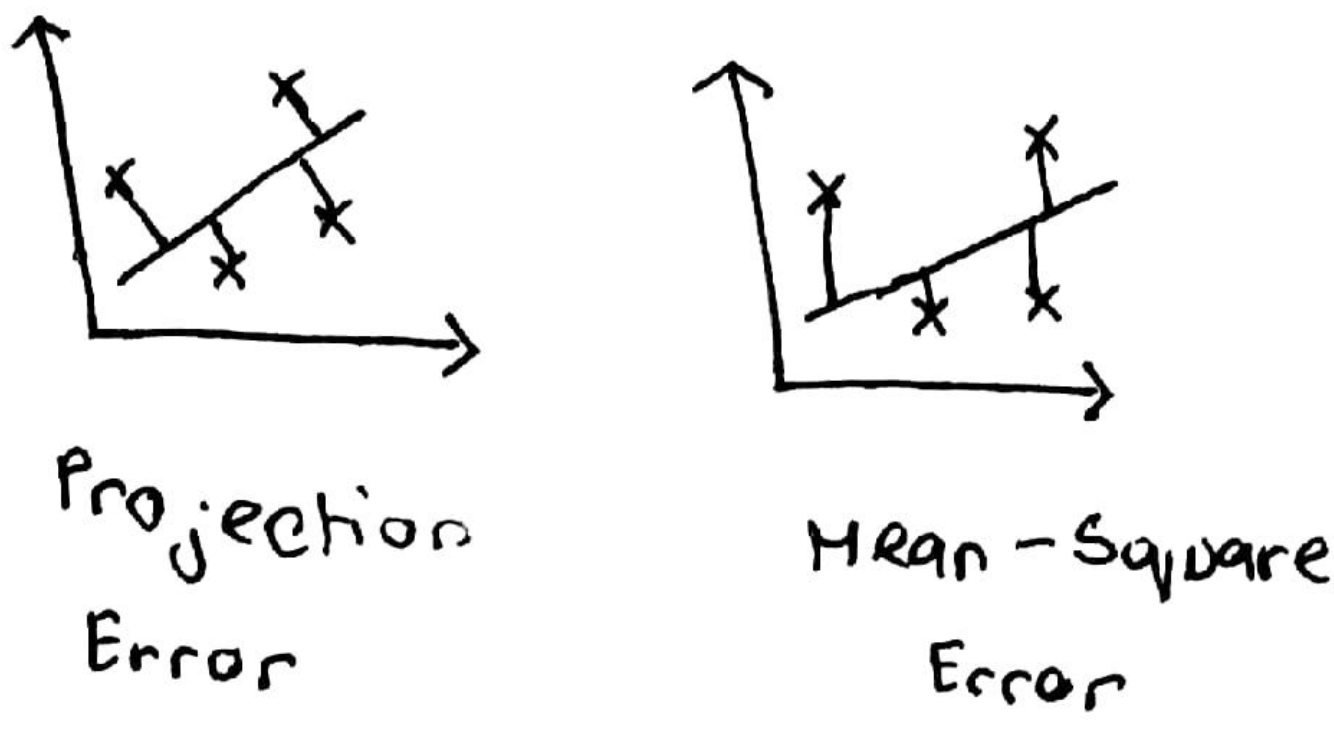
## Principal Component Analysis

In **Principal Component Analysis** (PCA), our goal is to maximize some component function while minimizing some loss function. This helps us to reduce the **feature space**. For example, if we have 10,000 features, we can bring that down to suppose 10 features. Such a huge reduction improves performance. The trick however, is that we do not lose (much) information either, i.e. the loss of features does not affect our accuracy (much).

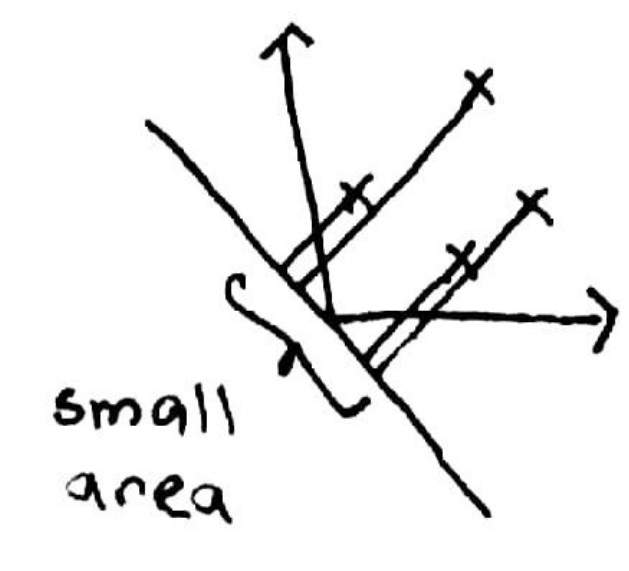
The **principal component** is a series of vectors in a -dimensional space. The th vector best fits the data while also being **orthogonal** to the previous vectors.



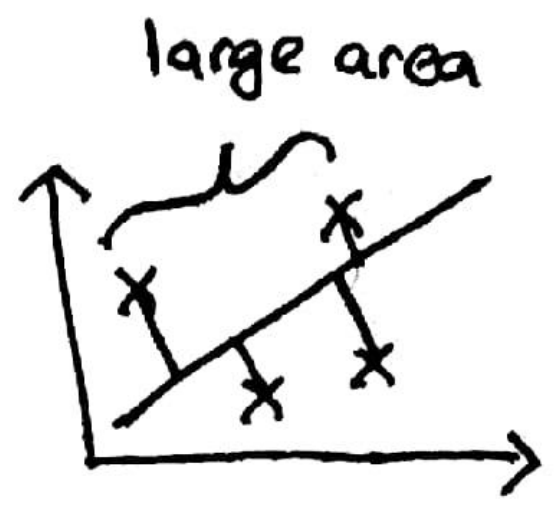
Consider that we have a 2D feature space with some data points. The **minimum error** will be found from the **line of best fit**. From each data point, we get a **projection error**, , , . The actual line is called the **projection line**. Notice that we are minimizing the error, which was one of the requirements of PCA.



On a side note, notice that the **projection error** is not the same as the **mean square error**, which means PCA is not the same as Linear Regression.



Alternatively, we could have chosen some line like the one above. However, not only does this line give us a larger area, the length of the line is also smaller. Essentially, since we are compressing the data into a smaller space, we are losing more information. The **variance** between the data points is important since it gives us some information. By using the line of best fit, we are capturing more of the variance and thus **maximizing** the content, which was the other requirement of PCA.

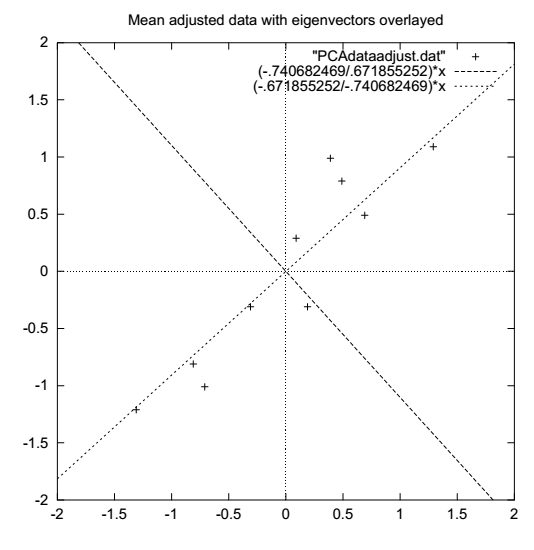


It is also possible to find a line which maximizes the content but also has high error. We do not want that either.

By using the line of best fit, we have reduced the feature space from 2D to 1D.

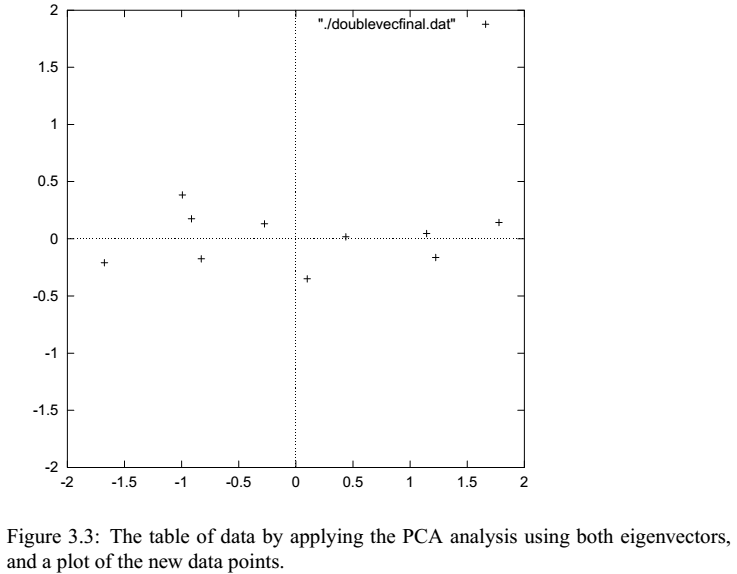
### Principal Components

For a 2D feature space, the two principal components might look like this:



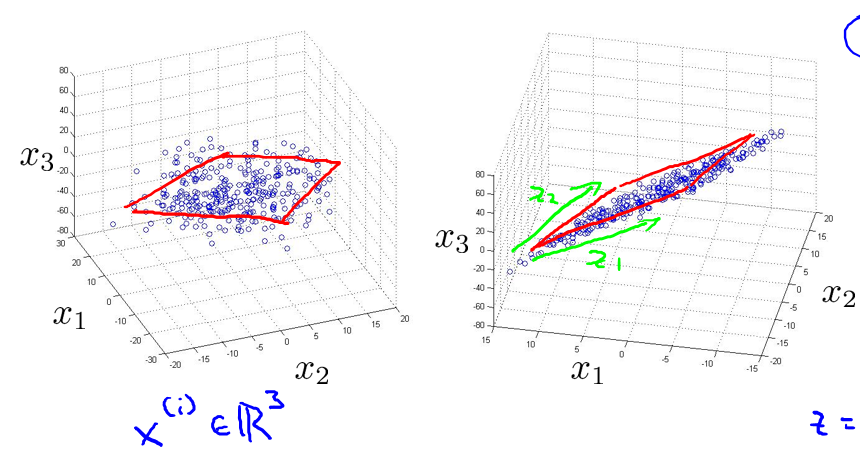
Essentially, these are the two vectors which give us the least error and highest variance coverage while also being orthogonal to each other. We will later see that these are just the eigenvectors, so for now, do not worry about how we got these vectors.

If we rotate the feature space so that the two vectors are horizontally and vertically aligned respectively, we will see that the variance of the data along the second vector (the vertical one here) is very little. Thus, if we discard the second dimension, we are not losing too much information. This will be the basis for the rest of the topics under PCA.



### 3D Feature Spaces

We have yet to see how multiple vectors are made to be orthogonal to each other to create the principal component. We know that a 2D plane has axes which are orthogonal to each other. If we have a 3D feature space, we are essentially finding the **plane** of best fit.



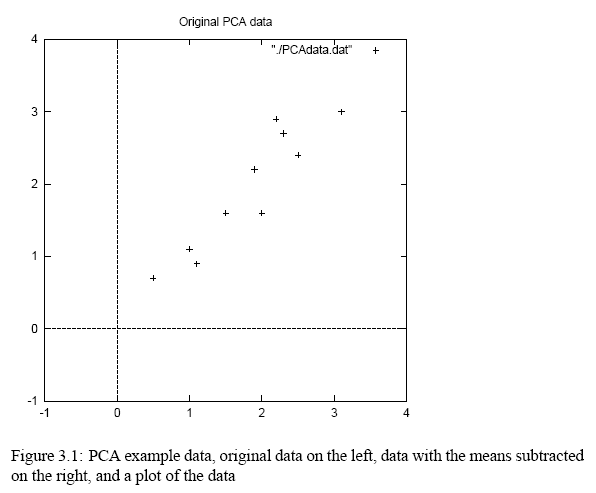
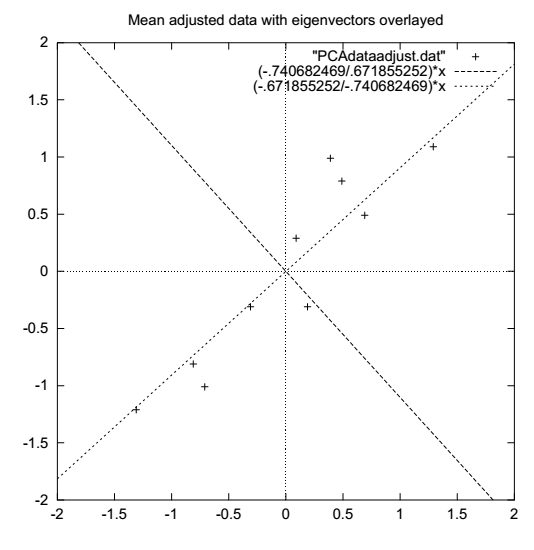
Remember that this is not as simple as flatting the data points along one axis. We are having to find two separate lines which best fit the data, under the condition that the first line is the actual line of best fit and the second line is the line of best fit while also being orthogonal to the first line. These two lines, and , form the principal component .

### Algorithm

The PCA algorithm has 4 steps:

1. Subtract the mean
2. Calculate the covariance matrix
3. Calculate the eigenvectors and eigenvalues
4. Choose components and form a feature vector

Firstly, we **subtract the mean**. We do this to bring the data to the centre, which makes further calculation relatively easier.

We can also perform **feature scaling** at this stage if required.

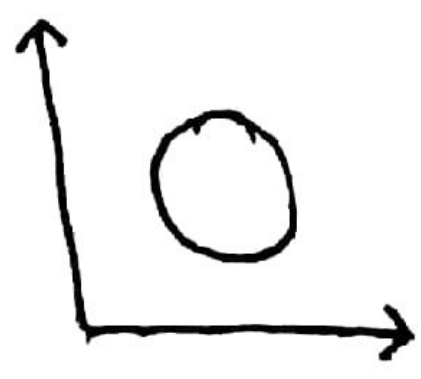
Next, we calculate the **covariance matrix**.

For features, this gives us an matrix.

This formula can also be vectorized in the format:

The covariance matrix informs us about the **shape** of the data distribution. For example, suppose we have two features, and . The covariance matrix will thus be

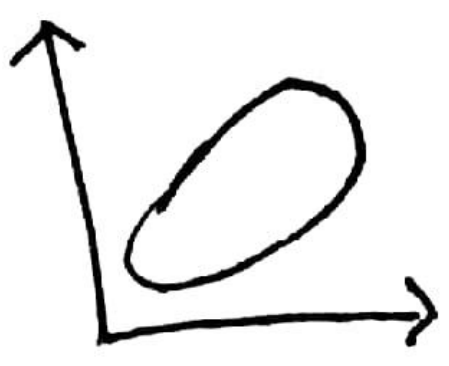
Here, each cell tells us about the variance of the first feature with respect to the second. Thus, if we have , it means the change in and are equal, but they are independent. This is a circle.



On the other hand, if we have , it means the change in is twice the change in , but they are still independent. This is an oval.



Finally, if we have , then there is dependence. The shape is somewhat like this:



The third step is to find the **eigenvalues** and **eigenvectors** of the covariance matrix. As a reminder, the eigenvector of a matrix is given by

where is the corresponding eigenvalue.

We can first calculate the eigenvalue from

and then find from

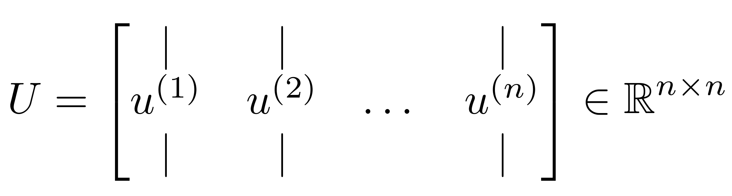
For an matrix, we will find eigenvalues and eigenvectors, each of which are orthogonal to each other.

The final step is to choose the components to keep. The values represent the variance of the data in the direction of the corresponding vector. Thus, since we want to keep the maximum variance vectors, we can just sort the values and keep the corresponding vectors. If we have eigenvectors, we can choose to keep of them to reduce the feature space from to .



From the diagram above, we can see that the values of are decreasing. Essentially, this is telling us that getting rid of eigenvectors corresponding to the latter values does not cause us to lose much information. In fact, it is even possible to get rid of some values without losing any information at all.

Practically, we calculate eigenvectors and eigenvalues using a process called **Singular Value Decomposition**. We do not need to know the details of this. In Python, there is a method, np.lingalg.svd, which performs this for us. This returns three values, the first of which is the eigenvectors arranged in columns.



## Data Reconstruction

We now have the eigenvectors as a series of column vectors in the variable . For a particular value of in the original feature space, we can find the value in the new feature space as:

Vectorizing this, we get

This gives us a series of column vectors, .

We had to use since is a series of column vectors. However, if we change so that each **row** represents an eigenvector instead, we can use this equation:

This representation is about to make things way easier for us, since we will now work on **reconstructing** the original data in the original feature space from the new data in the new feature space.

To find from , we can do this:

is **orthonormal**. One of the properties of orthonormal matrices is that their inverse is the same as their transpose. Thus,

We originally had an feature space, which we reduced to a feature space. Thus, is a matrix, is an matrix and is an matrix.

The value of we get is also **mean adjusted**, i.e. the mean was subtracted from the original data. Thus, we need to add it back.

We should have the original back now, but we won’t. This is the **reconstructed version**. The only scenarios in which we will get exactly the original data back is if , meaning no features were removed, or if the features that were removed did not have any variance.

## Choosing the Number of Principle Components

The reason we went through the whole reconstruction process is to get the final . We can use this to find the **error** caused by the feature reduction.

For example, if we get a value of here, we can say that variance was retained. Depending on what percentage of the variance we want to retain, we can choose a value for , the number of principle components to keep.

An easier way to do this is to use the **eigenvalues**. When using the SVD method, we get back three values, The second value is , which is a **diagonal matrix** in which the diagonal values are the values. If we take of the eigenvectors, the total error is thus

## Applying PCA

If we have a series of inputs, , , , we can map these to a series of new inputs, , , . This gives us a few benefits:

* Compressing the data speeds up the learning process
* Less memory is needed to store the data
* Visualization

We should choose a value of which gives us a **significant increase in speedup** while also not losing too much variance. If this cannot be achieved, then it is not recommended to use PCA, since the process of calculating eigenvalues and eigenvectors is extremely computationally expensive.

For the visualization benefit in particular, if we have a value of or , we can actually see what the data looks like graphically.

However, we should not use PCA to reduce **overfitting**. The entire goal of PCA is to keep all of the information. If a particular feature is not contributing to the variance, it will be discarded by PCA. However, the fact that it was not contributing to the variance in the first place also means it was not contributing to overfitting. Thus, even if we use PCA, we will still have an overfitting problem.