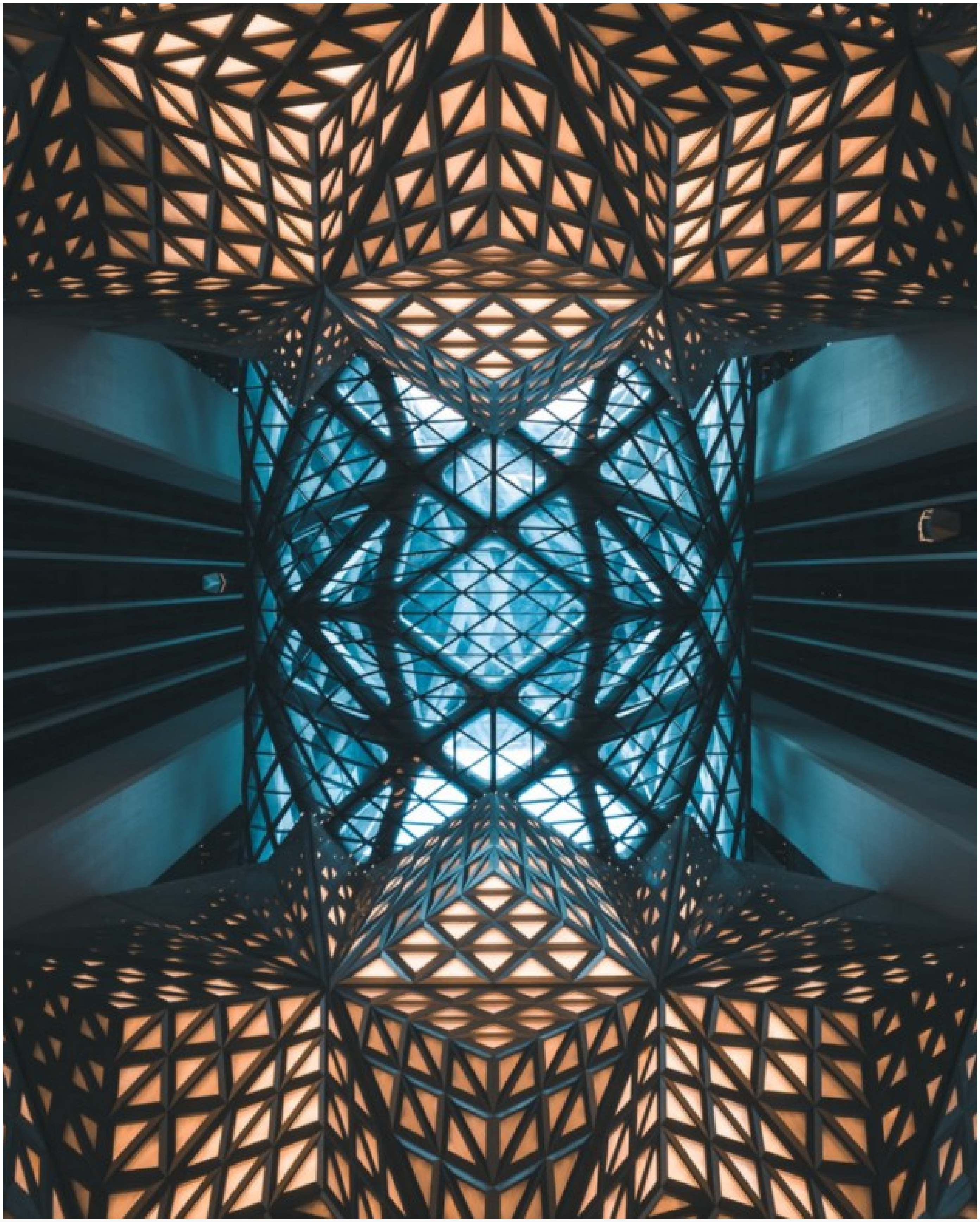
Victor Roman Follow

Apr 17, 2019 · 9 min read · · Listen

Unsupervised Learning: Dimensionality Reduction

Compress features, reduce overfitting and noise and increase eficciency and performance

Open in app Get started



As stated in previous articles, unsupervised learning refers to a kind of machine learning algorithms and techniques that are trained and fed with unlabeled data. In other words, we do not know the correct solutions or the values of the target variable beforehand.

The main goal of these types of algorithms is to study the intrinsic and hidden structure of the data in order to get meaningful insights, segment the datasets in similar groups or to simplify them.

Throughout this article, we are going to explore some of the algorithms and techniques most commonly used to reduce the dimensionality of datasets.

Basics of Dimensionality Reduction

Dimensionality is the number of variables, characteristics or features present in the dataset. This dimensions are represented as columns, and the goal is to reduce the number of them.

In most cases, those columns are correlated and, therefore, there is some information that is redundant which increase the dataset's noise. This redundant information impacts negatively in Machine Learning model's training and performance and that is why using dimensionality reduction methods becomes of paramount importance. It is a very useful way to reduce model's complexity and avoid overfitting.

There are two main categories of dimensionality reduction:

- Feature Selection → we select a subset of features of the original dataset.
- Feature Extraction → we derive information from the orginal set to build a new feature subspace.

Feature Selection

Feature Selection stands for a family of greedy algorithms used to reduce the dimensional feature space of a given dataset. The aim is to obtain a model capable of automatically

> *algorithms, which evaluate the whole set of combinations and yield the overall optimal solution. The benefits of greedy algorithms is that they are computationally much more efficient, in expenses of precision, but, most of the times they yield sufficiently good solutions.*

In that way, we will improve the computation efficiency of the training process, reduce dataset's noise, avoid overfitting and reduce model's complexity.

We will study two of the main techniques of feature selection:

- Sequential Backward Selection or SBS
- Random Forests Feature Importance

Sequential Backward Selection (SBS)

The idea behind the SBS algorithm is the following:

- We will set the final number of features *d* that we want in our dataset
- Then, we will set a criterion function that will be in charge of minimizing the loss of performance caused by removing one of the dataset's features.
- At each iteration, the algorithm will calculate this reduction of performance, by simply calculating the performance, before and after the removal of each of the current features.
- Then, the feature that results in the least performance reduction, will be removed from the dataset.
- If the number of features present in the dataset is equal to the *d* number (set at the beginning) the algorithms stops. Otherwise, it will complete another iteration.

Feature Importance With Random Forests

We have studied Random Forests algorithms in a previous article. These are a kind of ensemble algorithms that combine a set of weak Decision Tree models in order to build a more robust and precise one.

computed from all decision trees present in the forest. This will be done without making any assumptions of whether the data is linearly separable or not.

This is a quite simple method of obtaining feature importance, as we are going to use the random forest implementation of the scikit-learn library, which already collects feature importance by using the *feature_importance_* attribute after fitting a *RandomForesClassifier.*

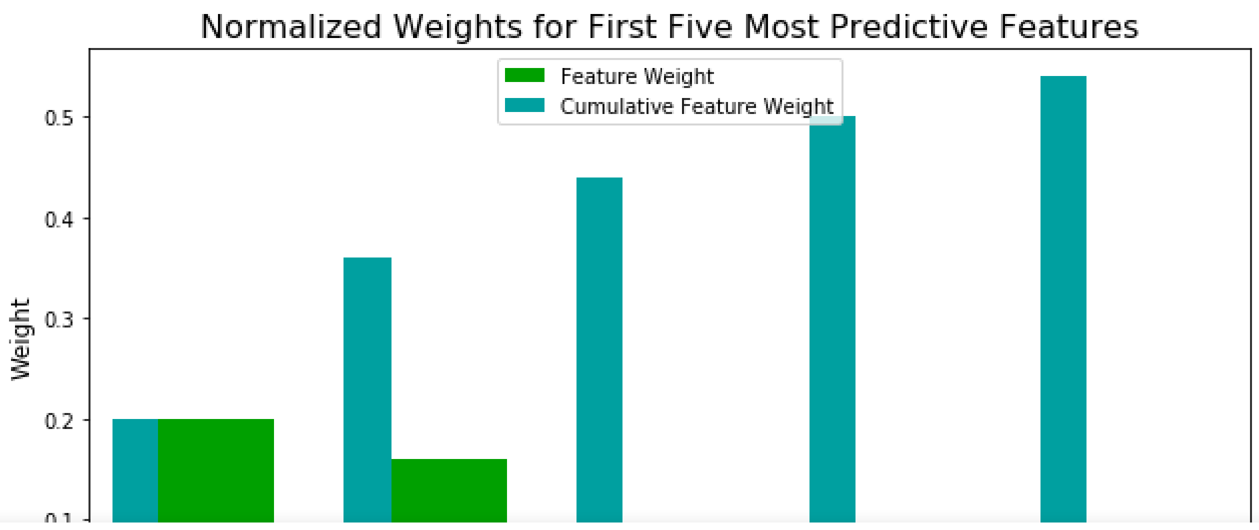
We already have studied an example of this in a previous article where we wanted to identify those features that most strongly help to predict wheter a specific individual made at most or more than $50.000.

```
# Import Ada Boost Classifier
from sklearn.ensemble import AdaBoostClassifier

# Train the supervised model on the training
model = AdaBoostClassifier().fit(X_train, y_train)

# Extract the feature importances using .feature_importances_
importances = model.feature_importances_

# Plot
vs.feature_plot(importances, X_train, y_train)
```



Open in app Get started

Feature Extraction

Feature extraction is also used to reduce the number of features of a certain dataset, but in contrast to feature selection, the output features will not be the same as the originals.

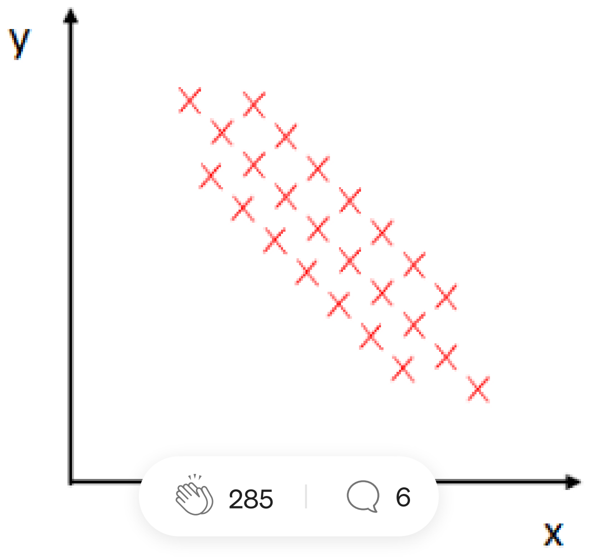
When using feature extraction, we project the data into a new feature space, so the new features will be combinations of the original features, compressed in a way that they will retain the most relevant information.

Some of the most used algorithms for unsupervised feature extraction are:

- Principal Component Analysis
- Random Projection
- Independant Component Analysis

Principal Component Analysis (PCA)

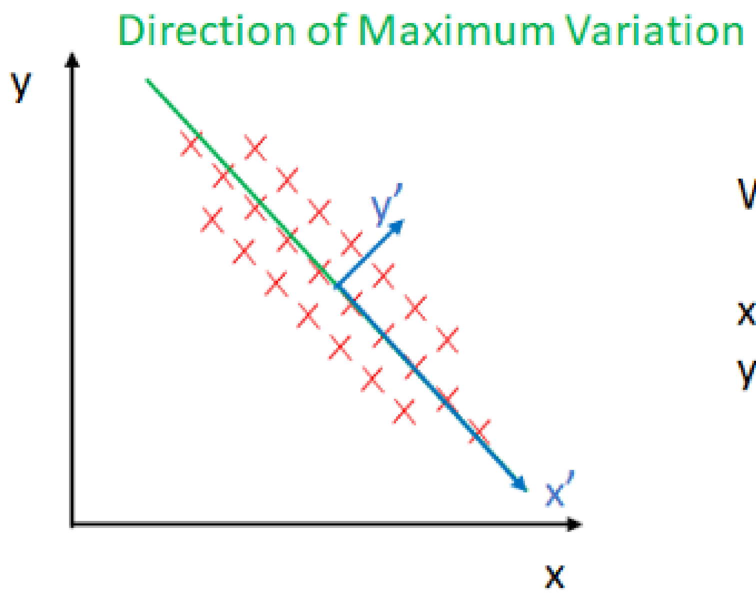
In order to understand how the PCA algorithm works, let's consider the following distribution of data:



285 6

PCA finds a new quadrant system (y' and x' axis) that it is obtained from the old one by

- It will then move the x-axis into the principal axis of variation, which is the one with most variation relative to data points (the direction of maximum spread).
- Then it moves other axis orthogonally to the principal one, into less important directions of variation.



With:

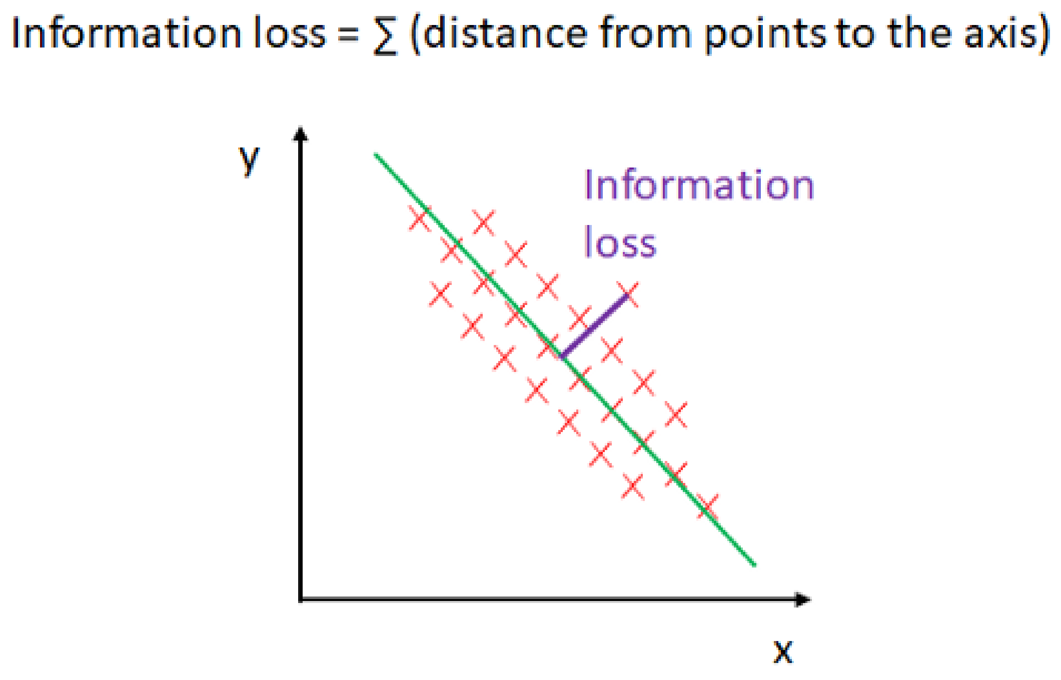
x' → Principal Component 1
y' → Principal Component 2

Basically, PCA finds the directions of maximum variance in high-dimensional data and projects this data into a new subspace with the same or fewer dimensions than the original one.

These new directions that contain the maximum variance are called Principal Components, they have the constraint of being orthogonall to each other.

Maximal Variance and Information Loss

Data points will be projected in the direction of maximal variance to form a new axis. The further the points are to the axis, the biggest the information loss.



It is a mathematical fact that when we project points onto a direction of maximal variance, it minimized the distance from old and higher-dimensional data points to its new transformed value. In other words, it minimizes the information loss.

Core ideas of PCA for Feature Transformation

As a summary, what PCA do is to combine every feature and to extract the top more relevant ones automatically. It is a systemized way to transform input features into principal components, and uses them as the new features.

Principal components are directions in data that maximize variance (minimize information loss) when projecting or compressing them down.

The more the variance of data along a Principal Component, the more information that direction contains and the higher that principal component is ranked.

The number of Principal Components will be less or equall the number of input features.

Scikit-Learn PCA Implementation

```
# Import PCA Algorithm
from sklearn.decomposition import PCA

# Initialize the algorithm and set the number of PC's
pca = PCA(n_components=2)

# Fit the model to data
pca.fit(data)

# Get list of PC's
pca.components_

# Transform the model to data
pca.transform(data)

# Get the eigenvalues
pca.explained_variance_ratio
```

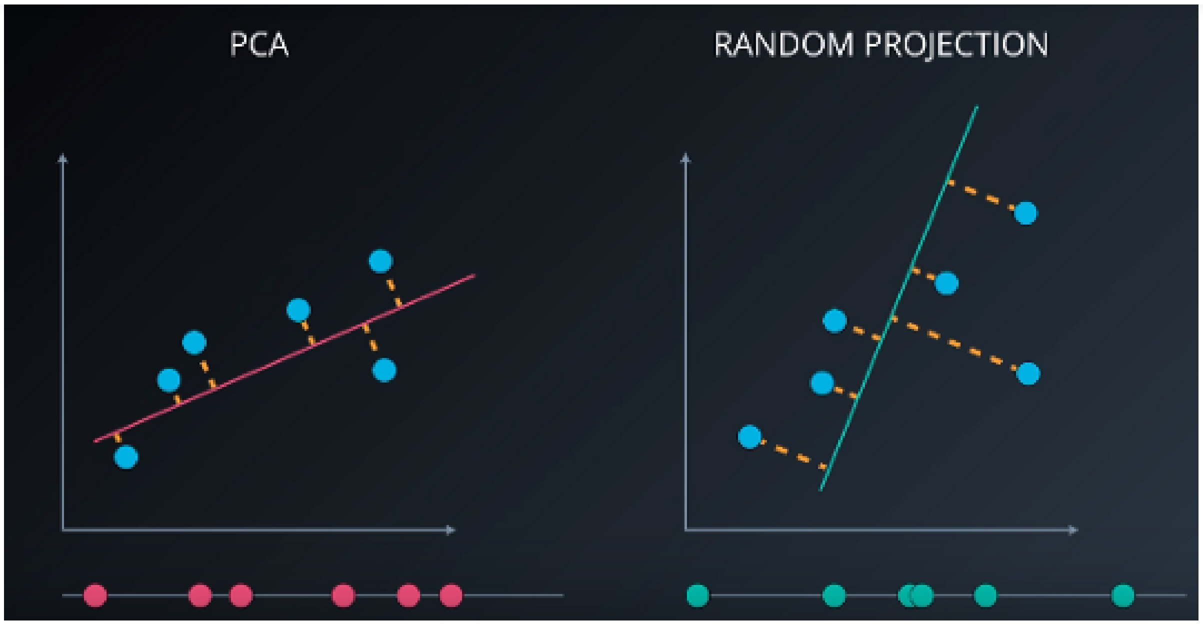
When To Use PCA

- When having latent features driving the patterns in data.
- For Dimensionality reduction.
- To visualize high-dimensional data.
- To reduce the noise.
- As a preprocessing step to improve the performance of other algorithms.

Random Projection

Random projection is a powerful dimensionality reduction method that is computationally more efficient tha PCA. It is commonly used in datasets that have too many dimensionsfor PCA to be directly computed.

Like PCA, it takes a dataset with d dimensions and n samples and produces a transformation of the dataset with k dimensions, being k much smaller than d ($k << d$).



Method

The basic premise is to reduce the number of dimension of our dataset by multiplying it to a random matrix. Which will project the dataset into a new subspace of features.

Theoreticall Approach: Johnson — Lindenstrauss Lemma

> *A datset with* N *samples in high-dimensional Euclidean space can be mapped down to a space in much lower dimension in a way that preserves the distance to the points to a large degree.*

In other words, the distance squared between two points in the dataset is calculated and the distance of those two points in the new dataset must be either:

- Smaller than the squared distance multiplied by (1-ε)
- Greater than the squared distance multiplied by (1+ε)

$$(1 - \varepsilon)\,||\,u - v\,||^2 < ||\,p(u) - p(v)\,||^2 < (1 + \varepsilon)\,||\,u - v\,||^2$$

```
# Import Random Projection Algorithm
from sklearn.random_projection import SparseRandomProjection

# Initialize the algorithm and set the number of PC's
randprojection = SparseRandomProjection()

# Fit and transformthe model to data
randprojection.fit_transform(data)
```

The epsilon value ε, is the level of error we are allowing between the points of the dataset. The default value of epsilon is 0.1.

We can use random projection by either setting a number of components or by specifying a value for epsilon and having the algorithm automatically calculating a conservative value for the number of dimensions.

Independant Component Analysis or ICA

ICA is a method for dimensionality reduction similar to PCA or Random Projection in the sense that it takes a set of features and produces a different set that is useful in some way.

But while PCA tries to maximize variance, ICA assumes that the features are mixtures of independent sources and it tries to isolate these independent sources that are mixed in the dataset.

The motivation behind ICA would be to take the original set of features and try to identify those of them that contribute independently to the dataset, in other words, those with the leat correlation to the other features. So it will isolate those most important components. This problem is called Blind Source Isolation.

High Level Algorithm

- X: Is our original Dataset.
- A: Mixing matrix

These variables are related as follows:

$$X = A*S$$
$$S = W*X$$
$$W = A^{-1}$$

So the goal is to calculate W, to be able to obtain S, the source matrix of the independant features.

To do so, the algorithm will perform the following steps:

1. Having X as the dataset, it will center and whiten it.
2. Choose an initial random weight matrix W1,W2,…,Wn.
3. Estimate W, containing vectors.
4. Decorrelate W.
5. Repeat from step 3 until convergence

ICA assumes that the components are statistical independent. They must have non Gaussian distributions, as we would not be able to restore the original signals if they were Gaussian.

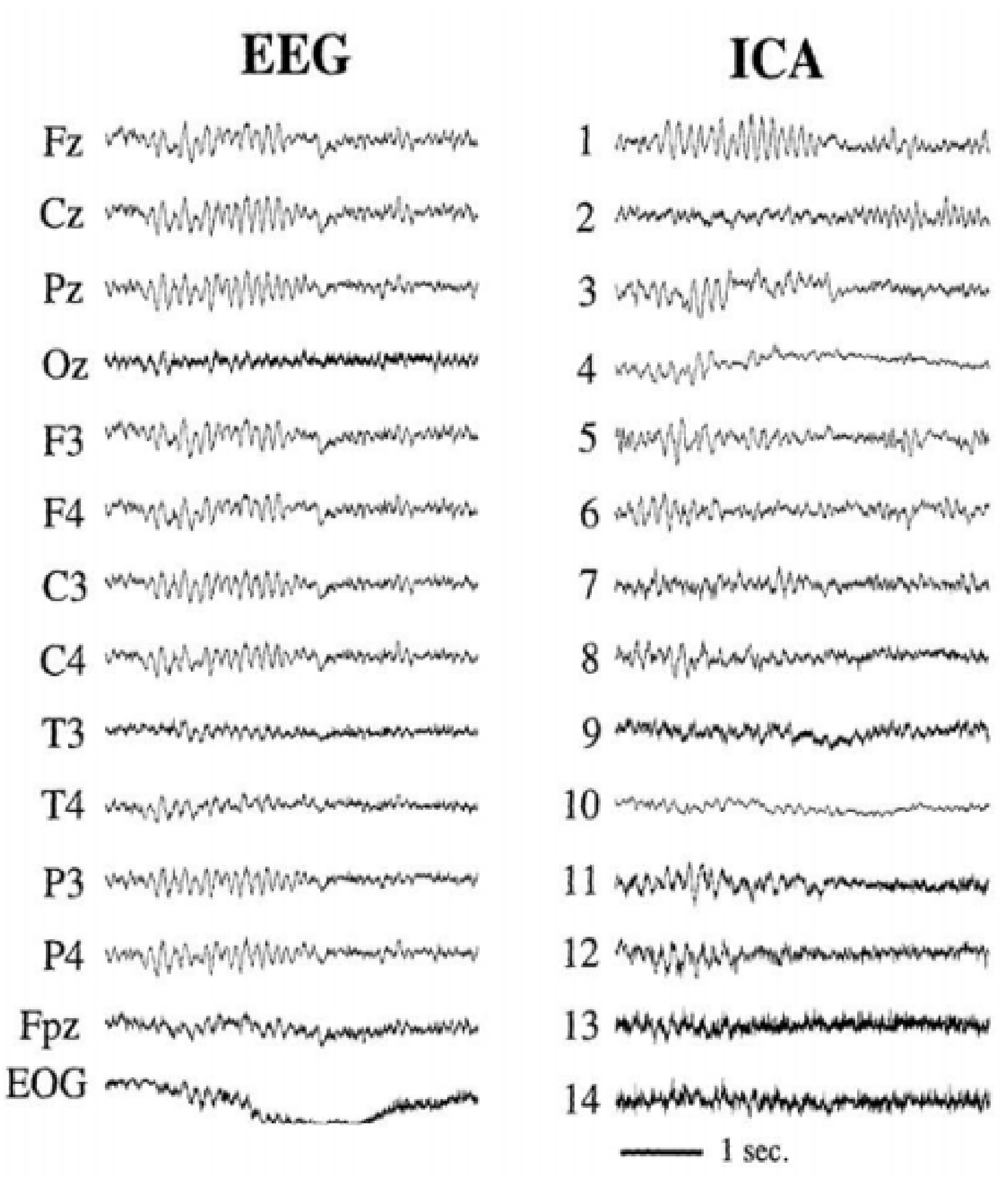
From this point, the central limit theorem says that the distribution of a sum of independent variables tends towards a Gaussian distribution.

ICA Implementation in Sci-kit learn

```
# Import Independent Component Analysis Algorithm
from sklearn.decomposition import FastICA

# Initialize the algorithm and set the number of PC's
```

One interesting applicaiton of ICA is the analysis of Electroenphalographic data. The following is an example of the readings of 14 channels from an EEG scan that lasted 4.5 seconds and the independent components extracted from the dataset.



Original Paper: https://www.semanticscholar.org/paper/Applying-Independent-Component-Analysis-to-Factor-Cha-Chan/a34be08a20eba7523600203a32abb026a8dd85a3

Final Words

If you want to learn more about Machine Learning, Data Science and Artificial Intelligence ***follow me on Medium****, and stay tuned for my next posts!*