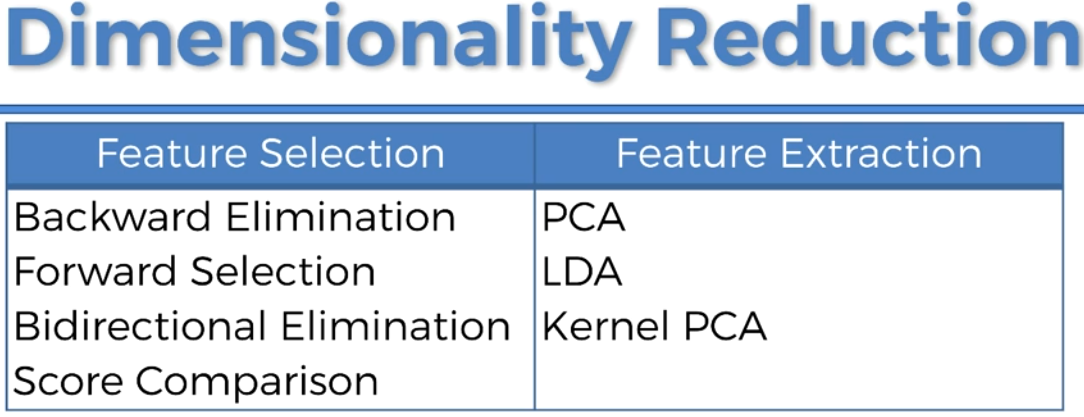
# DIMENSIONALITY REDUCTION TECHNIQUES

Dimensionality reduction or dimension reduction is the process of reducing the number of random variables under consideration. by obtaining a set of principal variables. It can be divided into feature selection and feature extraction.



I wanted to review three similar but different tasks:

* **FEATURE EXTRACTION AND FEATURE ENGINEERING**: transformation of raw data into features suitable for modeling;
* **FEATURE TRANSFORMATION:** transformation of data to improve the accuracy of the algorithm;
* **FEATURE SELECTION:** removing unnecessary features.

# FEATURE SELECTION

Feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features for use in model construction.

The central premise when using a feature selection technique is that the data contains some features that are either redundant or irrelevant, and can thus be removed without incurring much loss of information

A feature selection algorithm search technique proposes a new feature subset along with an evaluation measure which scores the different feature subsets. The simplest algorithm is to test each possible subset of features finding the one which minimizes the error rate.

The choice of evaluation metric heavily influences the algorithm, and it is these evaluation metrics which distinguish between the three main categories of feature selection algorithms:

1. Wrappers methods
2. Filters methods
3. Embedded methods

## WRAPPER METHODS

Wrapper methods use a predictive model to score feature subsets. Each new subset is used to train a model, which is tested on a hold-out set. Counting the number of mistakes made on that hold-out set (the error rate of the model) gives the score for that subset.

As wrapper methods train a new model for each subset, they are very computationally intensive, but usually provide the best performing feature set for that particular type of model.

We have following techniques in wrapper methods:

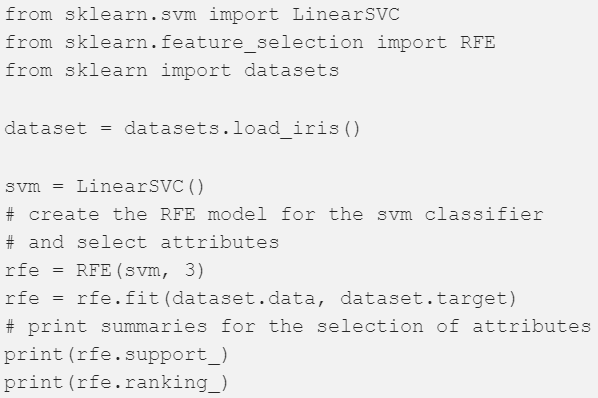
* Stepwise selection
* Forward selection
* Backward selection
* Recursive feature engineering (usually used with SVM)

We don’t have forwards selection and backward selection methods in Python but we have other equivalent methods which can be used instead or we can implement using custom code.

Below are few techniques that we can use for feature selection. Recursive feature elimination technique in Python is equivalent to Forward selection method. It can be used both for regression and classification.

### RECURSIVE FEATURE ELIMINATION

One other popular approach is the Recursive Feature Elimination algorithm, commonly used with Support Vector Machines to repeatedly construct a model and remove features with low weights.



We can use SelectFromModel Meta-transformer for selecting features based on importance weights. We can use this meta transformer with Lasso and Tree regressor or classifier models.

## FILTER METHODS

Filter methods use a proxy measure (like relation between IV and DV) instead of the error rate to score a feature subset. This measure is chosen to be fast to compute, while still capturing the usefulness of the feature set. Common measures include the MI (mutual information), Pearson Correlation Coefficient (r) and scores of significance tests for each class/feature combinations. We have following methods:

* F Test
* Mutual Information
* Variance threshold

Filters are usually less computationally intensive than wrappers, but they produce a feature set which is not tuned to a specific type of predictive model. This lack of tuning means a feature set from a filter is more general than the set from a wrapper, usually giving lower prediction performance than a wrapper. However, the feature set doesn't contain the assumptions of a prediction model, and so is more useful for exposing the relationships between the features.

Filter methods have also been used as a preprocessing step for wrapper methods, allowing a wrapper to be used on larger problems.

One of the most popular form of feature selection is stepwise regression, which is a wrapper technique. It is a greedy algorithm that adds the best feature (or deletes the worst feature) at each round. The main control issue is deciding when to stop the algorithm. In machine learning, this is typically done by cross-validation.

Filter Methods considers the relationship between features and the target variable to compute the importance of features. Below are the techniques used in filter methods

### F – Test

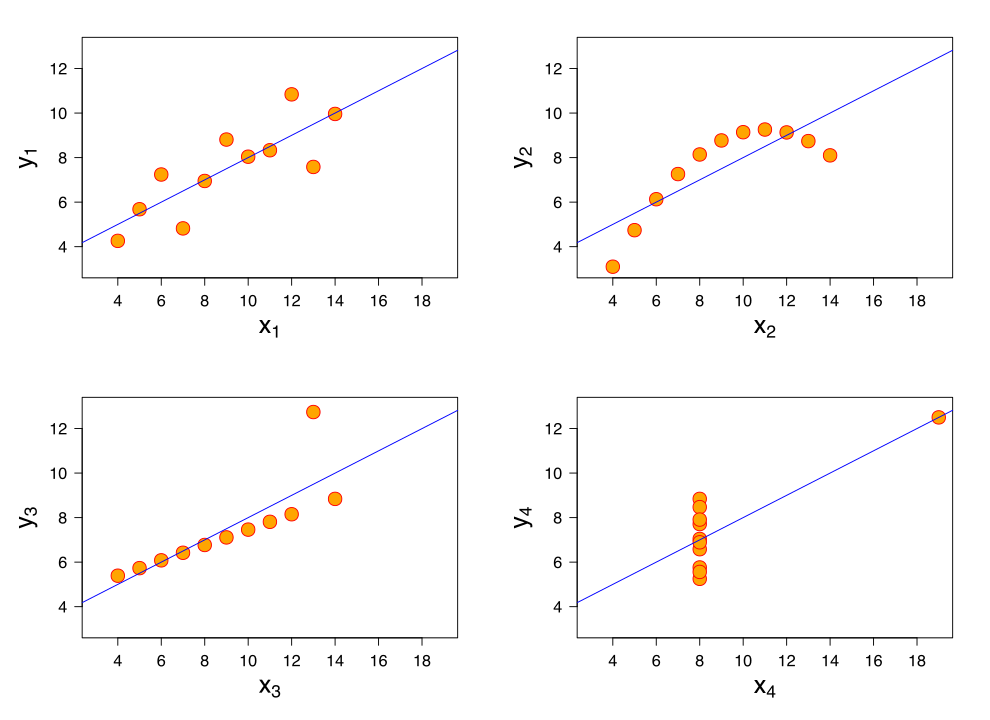
F-Test does a hypothesis testing between model X and Y where X is a model created by just a constant and Y is the model created by a constant and a feature. The least square errors in both the models are compared and checks if the difference in errors between model X and Y are significant or introduced by chance.

F-Test is useful in feature selection as we get to know the significance of each feature in improving the model.

Below are the drawbacks of using F-Test

* F-Test captures only linear relationship between features and labels. If your label and features are not linearly related then this measure will not give accurate results.
* A highly correlated feature is given higher score and less correlated features are given lower score.

Using Correlation for feature selection could be highly deceptive if DV and IV are not linearly correlated as it doesn’t capture strong non-linear relationships. Hence using summary statistics like correlation may be a bad idea, as illustrated by Anscombe’s quartet.



Francis Anscombe illustrates how four distinct datasets have same mean, variance and correlation to emphasize ‘summary statistics’ does not completely describe the datasets and can be quite deceptive.

We can perform F – TEST in python using below functions:



### MUTUAL INFORMATION

Mutual Information between two variables measures the dependence of one variable to another. If X and Y are two variables, and

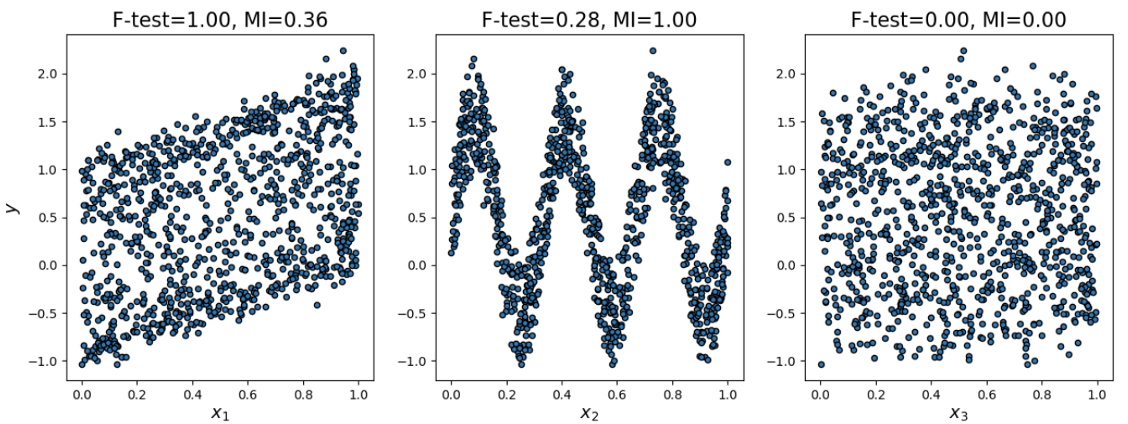
* If X and Y are independent, then no information about Y can be obtained by knowing X or vice versa. Hence their mutual information is 0.
* If X is a deterministic function of Y, then we can determine X from Y and Y from X with mutual information 1.
* When we have Y = f (X, Z, M, N), 0 < mutual information < 1

We can select our features from feature space by ranking their mutual information with the target variable.

Sklearn offers feature selection with Mutual Information for regression and classification tasks.



F-Test captures the linear relationship well. Mutual Information captures any kind of relationship between two variables.



### VARIANCE THRESHOLD

This method removes features with variation below a certain cutoff. The idea is when a feature doesn’t vary much within itself, it generally has very little predictive power.



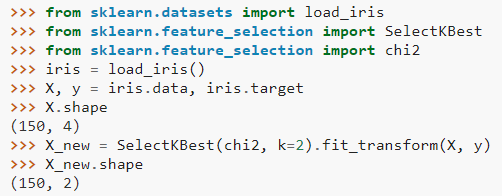
Variance Threshold doesn’t consider the relationship of features with the target variable.

### SELECTKBEST

Using SelectKBest object from feature\_selection class in sklearn we can perform above mentioned test. These objects take as input a scoring function that returns univariate scores and p-values:

* For regression: f\_regression, mutual\_info\_regression
* For classification: chi2, f\_classif, mutual\_info\_classif

The methods based on F-test estimate the degree of linear dependency between two random variables. On the other hand, mutual information methods can capture any kind of statistical dependency, but being nonparametric (does not assume any distribution assumptions) they require more samples for accurate estimation.



We can use fit\_transform to create a new set of X with only best features as required.

## EMBEDDED METHODS

Embedded methods are a catch-all group of techniques which perform feature selection as part of the model construction process. The example of this approach is the LASSO method for constructing a linear model, which penalizes the regression coefficients with an L1 penalty, shrinking many of them to zero.

Any features which have non-zero regression coefficients are 'selected' by the LASSO algorithm. Improvements to the LASSO include:

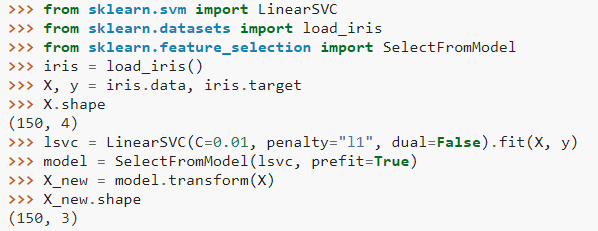
* Bolasso which bootstraps (random sampling with replacement) samples.
* Elastic net regularization, which combines the L1 penalty of LASSO with the L2 penalty of Ridge Regression.
* FeaLect which scores all the features based on combinatorial analysis of regression coefficients.

These approaches tend to be between filters and wrappers in terms of computational complexity.

### PENALTY BASED FEATURE SELECTION

Linear models penalized with the L1 norm have sparse solutions, many of their estimated coefficients are zero. When the goal is to reduce the dimensionality of the data to use with another classifier, they can be used along with feature\_selection.SelectFromModel to select the non-zero coefficients.

In particular, sparse estimators useful for this purpose are the linear\_model.Lasso for regression, and of linear\_model.LogisticRegression and svm.LinearSVC for classification.

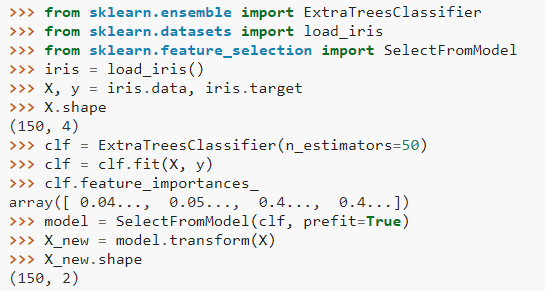


With SVMs and logistic-regression, the parameter C controls the sparsity: the smaller C the fewer features selected. With Lasso, the higher the alpha parameter, the fewer features selected.

There is no general rule to select an alpha parameter for recovery of non-zero coefficients. It can be set by cross-validation (LassoCV or LassoLarsCV), though this may lead to under-penalized models: including a small number of non-relevant variables is not detrimental to prediction score. BIC (LassoLarsIC) tends, on the opposite, to set high values of alpha.

### TREE-BASED FEATURE SELECTION

Tree-based estimators can be used to compute feature importance, which in turn can be used to discard irrelevant features



## WHY USE FEATURE SELECTION

Feature selection techniques are used for below reasons:

* Simplification of models to make them easier to interpret by researchers/users
* Shorter training times,
* To avoid the curse of dimensionality,
* Enhanced generalization by reducing overfitting (formally, reduction of variance)
* In some cases, data analysis such as regression or classification can be done in the reduced space more accurately than in the original space.

# FEATURE SACLING

Monotonic feature transformation is critical for some algorithms and has no effect on others. This is one of the reasons for the increased popularity of decision trees and all its derivative algorithms (random forest, gradient boosting).

Parametric methods usually require a minimum of symmetric and unimodal distribution of data, which is not always given in real data. However, data requirements are imposed not only by parametric methods but also by some methods like KNN.

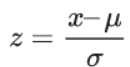
KNN will predict complete nonsense if features are not normalized e.g. when one distribution is located in the vicinity of zero and does not go beyond (-1, 1) while the other’s range is on the order of hundreds of thousands.

A simple example: suppose that the task is to predict the cost of an apartment from two variables — the distance from city center and the number of rooms. The number of rooms rarely exceeds 5 whereas the distance from city center can easily be in the thousands of meters.

## STANDARD SCALING

Feature scaling through standardization (or Z-score normalization) can be an important preprocessing step for many machine learning algorithms. Standardization involves rescaling the features such that they have the properties of a standard normal distribution with a mean of zero and a standard deviation of one.

The simplest transformation is Standard Scaling (or Z-score normalization):

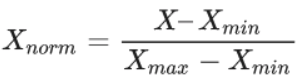




This will transform the feature to have mean 0 and variance 1. Note that Standard Scaling does not make the distribution normal in the strict sense.

## MINMAX SCALING

Another fairly popular option is MinMax Scaling, which brings all the points within a predetermined interval (typically (0, 1)).





StandardScaling and MinMax Scaling have similar applications and are often more or less interchangeable. However, if the algorithm involves the calculation of distances between points or vectors, the default choice is StandardScaling. But MinMax Scaling is useful for visualization by bringing features within the interval (0, 255).

## LOG NORMAL SCALING

If we assume that some data is not normally distributed but is described by the log-normal distribution, it can easily be transformed to a normal distribution.



The lognormal distribution is suitable for describing salaries, price of securities, urban population, number of comments on articles on the internet, etc. However, to apply this procedure, the underlying distribution does not necessarily have to be lognormal; you can try to apply this transformation to any distribution with a heavy right tail.

One can also try to use other similar transformations, formulating their own hypotheses on how to approximate the available distribution to a normal. Examples of such transformations are Box-Cox transformation (logarithm is a special case of the Box-Cox transformation) or Yeo-Johnson transformation (extends the range of applicability to negative numbers). In addition, you can also try adding a constant to the feature — np.log (x + const).

# FEATURE EXTRACTION

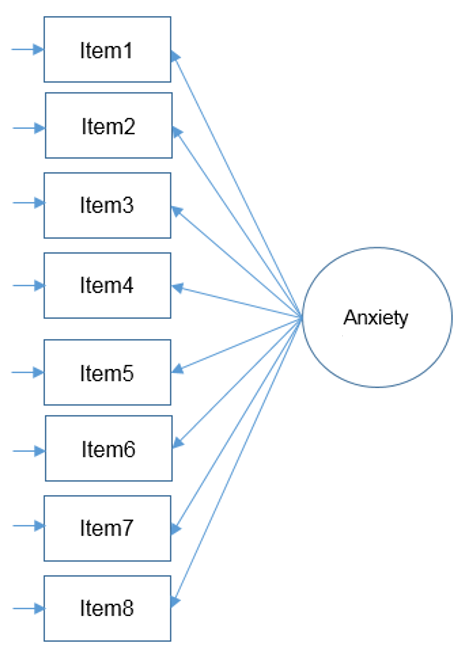
Feature extraction module can be used to extract features in a format supported by machine learning algorithms from datasets consisting of formats such as text and image. For example, transforming arbitrary data, such as text or images, into numerical features usable for machine learning.

Feature extraction creates new features from functions of the original features, whereas feature selection returns a subset of the features.

# FACTOR ANALYSIS

It is an unsupervised statistical learning technique used to examine the interrelation among a set of variables in order to identify the underlying structure of those variables.

The basic assumption of factor analysis is that for a collection of observed variables there are a set of underlying variables called factors (smaller than the observed variables), that can explain the interrelationships among those variables. Let’s say you conduct a survey and collect responses about people’s anxiety.



Do all these items actually contribute to “Anxiety”?

Check the correlation between the features and if its significant then we can go ahead and perform factor analysis. Recall that the goal of factor analysis is to model the interrelationships between items with fewer (latent) variables. These interrelationships can be broken up into multiple components.

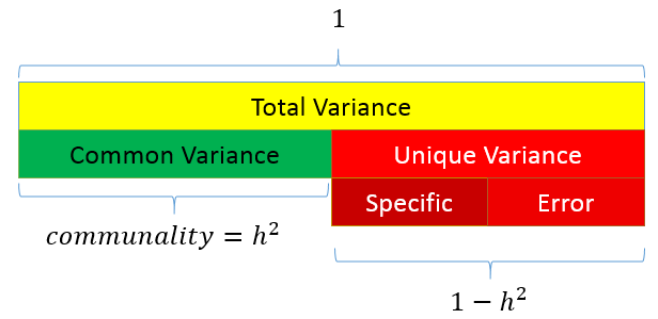
## PARTITIONING THE VARIANCE IN FACTOR ANALYSIS

Since the goal of factor analysis is to model the interrelationships among items, we focus primarily on the variance and covariance rather than the mean. Factor analysis assumes that variance can be partitioned into two types of variance, common and unique

* Common variance is the amount of variance that is shared among a set of items. Items that are highly correlated will share a lot of variance.
* Unique variance is any portion of variance that’s not common. There are two types:

Specific variance: is variance that is specific to a particular item (e.g., Item 4 “All computers hate me” may have variance that is attributable to anxiety about computers in addition to anxiety about SPSS).

Error variance: comes from errors of measurement and basically anything unexplained by common or specific variance (e.g., the person got a call from her babysitter that her two-year old son ate her favourite lipstick).



## PERFORMING FACTOR ANALYSIS

As a data analyst, the goal of a factor analysis is to reduce the number of variables to explain and to interpret the results. This can be accomplished in two steps:

* Factor extraction
* Factor rotation

Factor extraction involves making a choice about the type of model as well the number of factors to extract. Factor rotation comes after the factors are extracted, with the goal of achieving simple structure in order to improve interpretability.

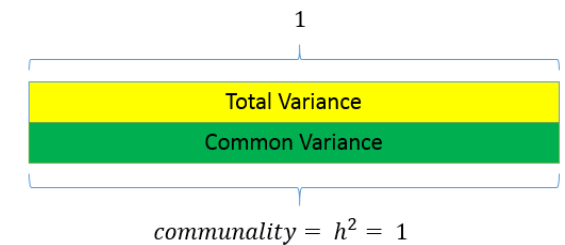
## EXTRACTING FACTORS

There are two approaches to factor extraction which stems from different approaches to variance partitioning:

* Principal components analysis and
* Common factor analysis.

## PRINCIPAL COMPONENTS ANALYSIS

Unlike factor analysis, principal components analysis or PCA makes the assumption that there is no unique variance, the total variance is equal to common variance. Recall that variance can be partitioned into common and unique variance. If there is no unique variance then common variance takes up total variance (see figure below). Additionally, if the total variance is 1, then the common variance is equal to the communality.



## EIGENVALUES

Eigenvalues represent the total amount of variance that can be explained by a given principal component. They can be positive or negative in theory, but in practice they explain variance which is always positive.

If eigenvalues are greater than zero, then it’s a good sign.

Since variance cannot be negative, negative eigenvalues imply the model is ill-conditioned.

Eigenvalues close to zero imply there is item multicollinearity, since all the variance can be taken up by the first component.

Eigenvalues are also the sum of squared component loadings across all items for each component, which represent the amount of variance in each item that can be explained by the principal component.

Eigenvectors represent a weight for each eigenvalue. The eigenvector times the square root of the eigenvalue gives the component loadings which can be interpreted as the correlation of each item with the principal component.

## COMMON FACTOR ANALYSIS

The partitioning of variance differentiates a principal components analysis from what we call common factor analysis. Both methods try to reduce the dimensionality of the dataset down to fewer unobserved variables, but whereas PCA assumes that there common variances takes up all of total variance, common factor analysis assumes that total variance can be partitioned into common and unique variance. It is usually more reasonable to assume that you have not measured your set of items perfectly. The unobserved or latent variable that makes up common variance is called a factor, hence the name factor analysis. The other main difference between PCA and factor analysis lies in the goal of your analysis. If your goal is to simply reduce your variable list down into a linear combination of smaller components then PCA is the way to go. However, if you believe there is some latent construct that defines the interrelationship among items, then factor analysis may be more appropriate.

## ROTATION METHODS

After deciding on the number of factors to extract and with analysis model to use, the next step is to interpret the factor loadings. Factor rotations help us interpret factor loadings. There are two general types of rotations, orthogonal and oblique.

* Orthogonal rotation assume factors are independent or uncorrelated with each other.
* Oblique rotation factors are not independent and are correlated.

The goal of factor rotation is to improve the interpretability of the factor solution by reaching simple structure.

## SIMPLE STRUCTURE

Without rotation, the first factor is the most general factor onto which most items load and explains the largest amount of variance. This may not be desired in all cases. Suppose you wanted to know how well a set of items load on each factor; simple structure helps us to achieve this.

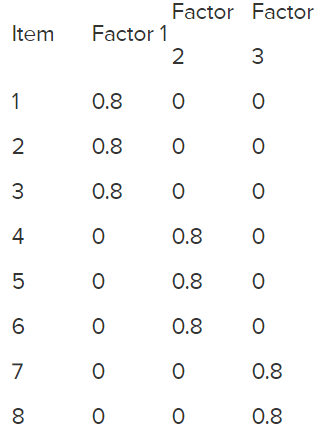
The definition of simple structure is that in a factor loading matrix:

* Each row should contain at least one zero.
* For m factors, each column should have at least m zeroes (e.g., three factors, at least 3 zeroes per factor).

For every pair of factors (columns),

* There should be several items for which entries approach zero in one column but large loadings on the other.
* A large proportion of items should have entries approaching zero.
* Only a small number of items have two non-zero entries.

The following table is an example of simple structure with three factors:



An easier set of criteria states that:

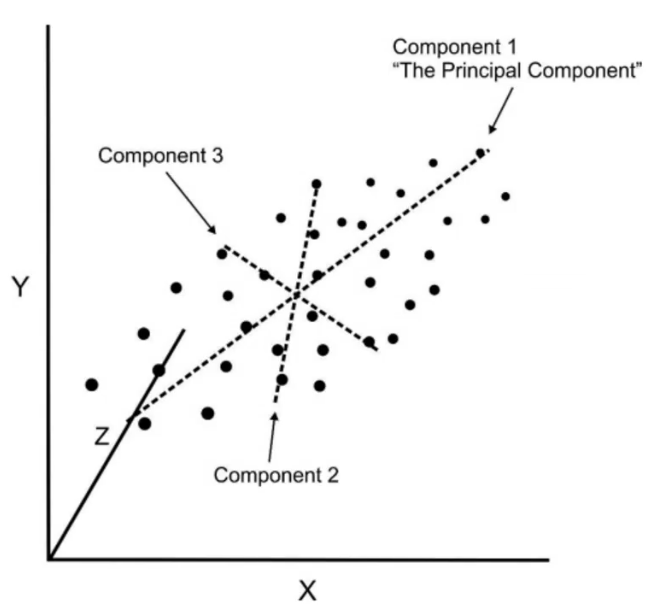
* Each item has high loadings on one factor only
* Each factor has high loadings for only some of the items.

# PCA (PRINCIPLE COMPONENT ANALYSIS)

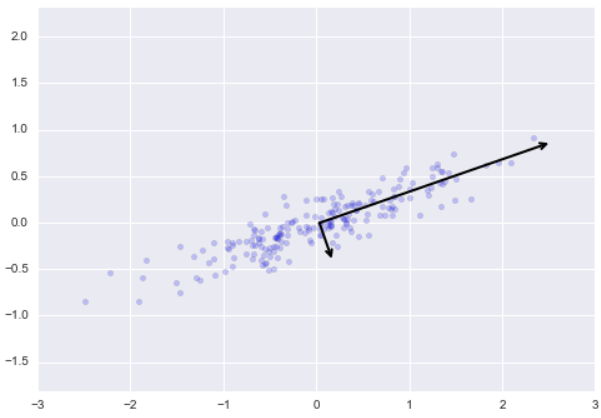
PCA is fundamentally a dimensionality reduction algorithm, but it can also be useful as a tool for visualization, for noise filtering, for feature extraction and engineering, and much more.

In principal component analysis, this relationship is quantified by finding a list of the principal axes in the data, and using those axes to describe the dataset.

* The PCA fit learns some quantities from the data, most importantly the "components" and "explained variance".
* Components define the direction of the vector (these components are technically known as "eigenvectors").
* Explained variance defines the squared-length of the vector.
* PCA is affected by scale so we need to scale features in our dataset before applying PCA.
* When dealing with large dataset, we can use RandomizedPCA. It contains a randomized method to approximate the first N principal components much more quickly than the standard PCA estimator, and thus is very useful for high-dimensional data (example dimensionality of nearly 3,000).
* Component are uncorrelated since in the sample space they are orthogonal to each other.

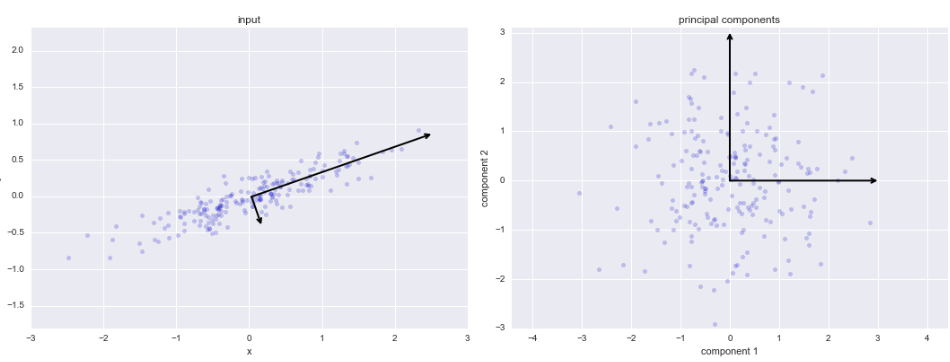


While many algorithms (such as SVM, K-nearest neighbors, and logistic regression) require features to be normalized, intuitively we can think of Principle Component Analysis (PCA) as being a prime example of when normalization is important. In PCA we are interested in the components that maximize the variance. If one component (e.g. human height) varies less than another (e.g. weight) because of their respective scales (meters vs. kilos), PCA might determine that the direction of maximal variance more closely corresponds with the ‘weight’ axis, if those features are not scaled. As a change in height of one meter can be considered much more important than the change in weight of one kilogram, this is clearly incorrect.



The vectors represent the principal axes of the data, and the length of the vector is an indication of how "important" that axis is in describing the distribution of the data—more precisely, it is a measure of the variance of the data when projected onto that axis. The projection of each data point onto the principal axes are the "principal components" of the data.

If we plot these principal components beside the original data, we see the plots shown here:

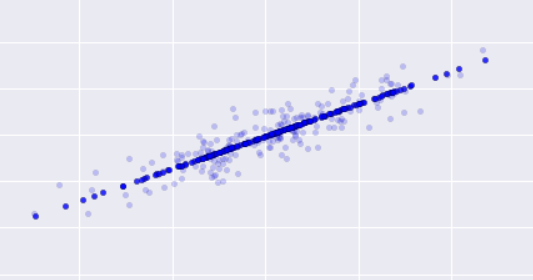


This transformation from data axes to principal axes is an affine transformation, which basically means it is composed of a translation, rotation, and uniform scaling.

## PCA AS DIMENSIONALITY REDUCTION

Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

Here is an example of using PCA as a dimensionality reduction transform:



The original dataset consists of 2 features and 200 observations where as transformed dataset with PCA contains 1 component and 200 observations.

The light points are the original data, while the dark points are the projected version. This makes clear what a PCA dimensionality reduction means: the information along the least important principal axis or axes is removed, leaving only the component(s) of the data with the highest variance. The fraction of variance that is cut out (proportional to the spread of points about the line formed in this figure) is roughly a measure of how much "information" is discarded in this reduction of dimensionality.

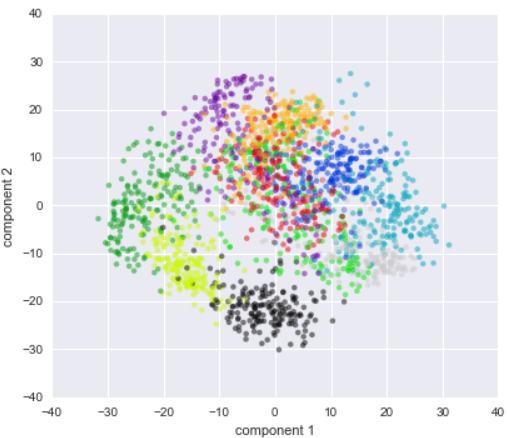
This reduced-dimension dataset is in some senses "good enough" to encode the most important relationships between the points: despite reducing the dimension of the data by 50%, the overall relationship between the data points are mostly preserved.

## PCA FOR VISUALIZATION

The usefulness of the dimensionality reduction may not be entirely apparent in only two dimensions, but becomes much clearer when looking at high-dimensional data.

Suppose we have image dataset with shape (1797, 64). To gain some intuition into the relationships between these points, we can use PCA to project them to a more manageable number of dimensions, say two (1797, 2).

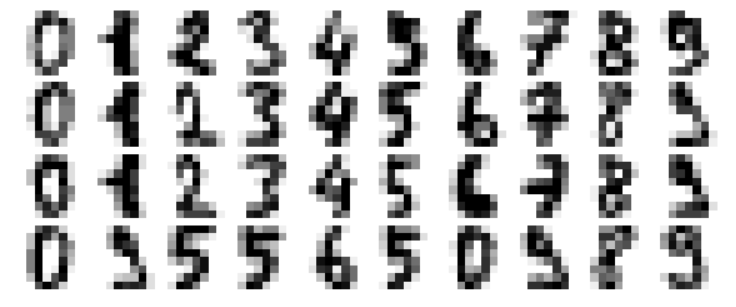
Now we can plot them and visualize the relationship.



## PCA AS NOISE FILTERING

PCA can also be used as a filtering approach for noisy data. The idea is this: any components with variance much larger than the effect of the noise should be relatively unaffected by the noise. So, if you reconstruct the data using just the largest subset of principal components, you should be preferentially keeping the signal and throwing out the noise.

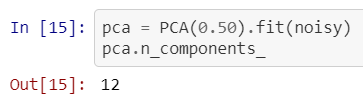
Let's see how this looks with the digits data. First, we will plot several of the input noise-free data:



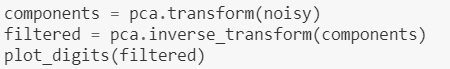
Now let’s add some random noise to create a noisy dataset, and re-plot it:

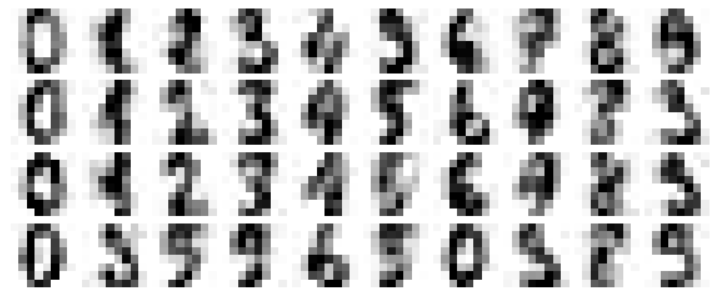


It's clear by eye that the images are noisy, and contain spurious pixels. Let's train a PCA on the noisy data, requesting that the projection preserve 50% of the variance:



Here 50% of the variance amounts to 12 principal components. Now we compute these components, and then use the inverse of the transform to reconstruct the filtered digits:

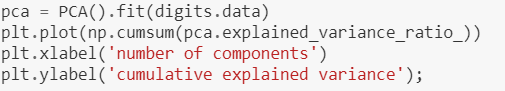


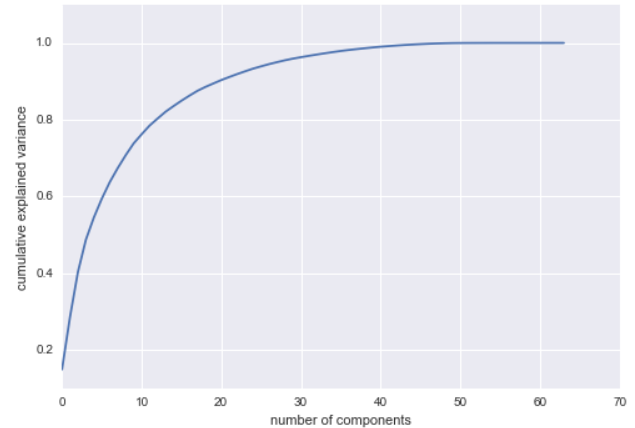


This signal preserving/noise filtering property makes PCA a very useful feature selection routine—for example, rather than training a classifier on very high-dimensional data, you might instead train the classifier on the lower-dimensional representation, which will automatically serve to filter out random noise in the inputs.

## CHOOSING THE NUMBER OF COMPONENTS

A vital part of using PCA in practice is the ability to estimate how many components are needed to describe the data. This can be determined by looking at the cumulative explained variance ratio as a function of the number of components:





This curve quantifies how much of the total, 64-dimensional variance is contained within the first $N$ components. For example, we see that with the digits the first 10 components contain approximately 75% of the variance, while you need around 50 components to describe close to 100% of the variance.

Here we see that our two-dimensional projection loses a lot of information (as measured by the explained variance) and that we'd need about 20 components to retain 90% of the variance. Looking at this plot for a high-dimensional dataset can help you understand the level of redundancy present in multiple observations.

# KERNEL PCA

Used for non linear features.

<https://sebastianraschka.com/Articles/2014_kernel_pca.html>

# PRINCE

Prince is a library for doing factor analysis. This includes a variety of methods including principal component analysis (PCA) and correspondence analysis (CA).

The fit method is actually an alias for the row\_principal\_components method which returns the row principal components. However, you can also access the column principal components with the column\_principal\_components.

Prince uses a randomised version of SVD. The randomised version of SVD is an iterative method. Because each of Prince's algorithms use SVD, they all possess a n\_iter parameter which controls the number of iterations used for computing the SVD. On the one hand the higher n\_iter is the more precise the results will be. On the other hand, increasing n\_iter increases the computation time. In general, the algorithm converges very quickly so using a low n\_iter (which is the default behaviour) is recommended.

You are supposed to use each method depending on your situation:

* All your variables are numeric: use principal component analysis (prince.PCA)
* You have a contingency table: use correspondence analysis (prince.CA)
* You have more than 2 variables and they are all categorical: use multiple correspondence analysis (prince.MCA)
* You have groups of categorical or numerical variables: use multiple factor analysis (prince.MFA)
* You have both categorical and numerical variables: use factor analysis of mixed data (prince.FAMD)

Both are data reduction techniques—they allow you to capture the variance in variables in a smaller set.

# LDA (LINEAR DISCRIMINANT ANALYSIS)

Linear Discriminant Analysis (LDA) is most commonly used as dimensionality reduction technique in the pre-processing step for pattern-classification and machine learning applications. The goal is to project a dataset onto a lower-dimensional space with good class-separability in order avoid overfitting (“curse of dimensionality”) and also reduce computational costs.

The general LDA approach is very similar to a Principal Component Analysis but in addition to finding the component axes that maximize the variance of our data (PCA), we are additionally interested in the axes that maximize the separation between multiple classes (LDA).

It is important to know that LDA assumes a normal distribution for each class, a class-specific mean, and a common variance.

In general, dimensionality reduction does not only help reducing computational costs for a given classification task, but it can also be helpful to avoid overfitting by minimizing the error in parameter estimation (“curse of dimensionality”).

## PREPARING DATA FOR LDA

This section lists some suggestions you may consider when preparing your data for use with LDA.

* **Classification Problems:** This might go without saying, but LDA is intended for classification problems where the output variable is categorical. LDA supports both binary and multi-class classification.
* **Gaussian Distribution:** The standard implementation of the model assumes a Gaussian distribution of the input variables. Consider reviewing the univariate distributions of each attribute and using transforms to make them more Gaussian-looking (e.g. log and root for exponential distributions and Box-Cox for skewed distributions).
* **Remove Outliers:** Consider removing outliers from your data. These can skew the basic statistics used to separate classes in LDA such the mean and the standard deviation.
* **Same Variance:** LDA assumes that each input variable has the same variance. It is almost always a good idea to standardize your data before using LDA so that it has a mean of 0 and a standard deviation of 1.

## EXTENSIONS TO LDA

Linear Discriminant Analysis is a simple and effective method for classification. Because it is simple and so well understood, there are many extensions and variations to the method. Some popular extensions include:

* **Quadratic Discriminant Analysis (QDA):** Each class uses its own estimate of variance (or covariance when there are multiple input variables).
* **Flexible Discriminant Analysis (FDA):** Where non-linear combinations of inputs is used such as splines.
* **Regularized Discriminant Analysis (RDA):** Introduces regularization into the estimate of the variance (actually covariance), moderating the influence of different variables on LDA.

## FEATURE SUBSPACE

Let’s assume that our goal is to reduce the dimensions of a d-dimensional dataset by projecting it onto a (k)-dimensional subspace (where k<d). So, how do we know what size we should choose for k (k = the number of dimensions of the new feature subspace), and how do we know if we have a feature space that represents our data “well”?

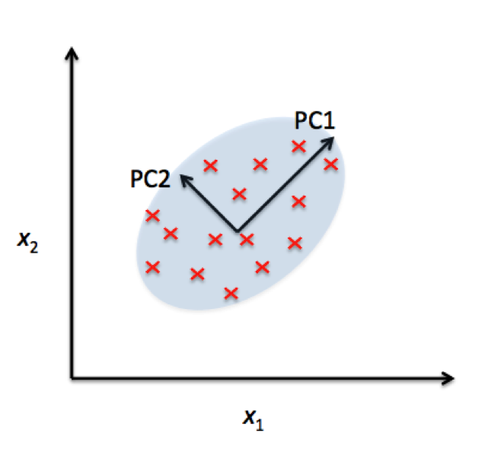
If all eigenvalues have a similar magnitude, then this may be a good indicator that our data is already projected on a “good” feature space.

And in the other scenario, if some of the eigenvalues are much larger than others, we might be interested in keeping only those eigenvectors with the highest eigenvalues, since they contain more information about our data distribution. Vice versa, eigenvalues that are close to 0 are less informative and we might consider dropping those for constructing the new feature subspace.

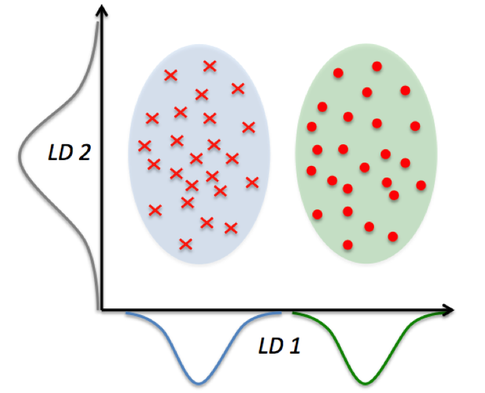
# LDA VS PCA VS FA

Both LDA and PCA are linear transformation techniques: LDA is a supervised whereas PCA is unsupervised – PCA ignores class labels.

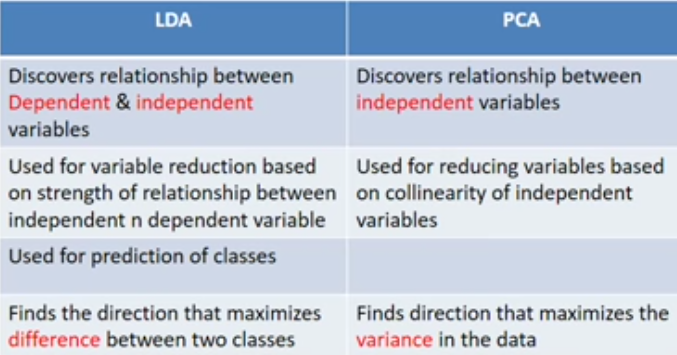
We can picture PCA as a technique that finds the directions of maximal variance:



In contrast LDA attempts to find a feature subspace that maximizes class separability.



Remember that LDA makes assumptions about normally distributed classes and equal class covariances.



* If your aim is to reduce features using correlation between IVs then perform PCA. It discovers relationship between independent variables.
* If your aim is to reduce features using covariance between IVs then perform FA.
* If your aim is to reduce features using correlation between IVS and dependent variable then perform LDA. It discovers relationship between independent and dependent variable.
* For features more than 30, both PCA and FA will yield same results.

Correlation is when the change in one item may result in the change in another item. On the other hand, covariance is when two items vary together.

# REFERENCES

Classification Feature selection using SVM

<https://medium.com/@aneesha/visualising-top-features-in-linear-svm-with-scikit-learn-and-matplotlib-3454ab18a14d>

<https://medium.com/open-machine-learning-course/open-machine-learning-course-topic-6-feature-engineering-and-feature-selection-8b94f870706a>

<https://github.com/MaxHalford/prince>

https://stats.idre.ucla.edu/spss/seminars/efa-spss/

https://sebastianraschka.com/Articles/2014\_python\_lda.html