Contents

[Topic 1: Association rule mining 4](#_Toc96369138)

[Association rules 4](#_Toc96369139)

[1.3 Introduction to association rules 4](#_Toc96369140)

[1.4 Rules and metrics definitions 4](#_Toc96369141)

[Definitions 5](#_Toc96369142)

[Transaction Database T 5](#_Toc96369143)

[Support 6](#_Toc96369144)

[Confidence 6](#_Toc96369145)

[Lift 6](#_Toc96369146)

[Conviction 6](#_Toc96369147)

[1.5 Brute force approach 7](#_Toc96369148)

[1.6 Apriori algorithm 8](#_Toc96369149)

[1.7 Apriori example 12](#_Toc96369150)

[1.8 MS-Apriori algorithm 13](#_Toc96369151)

[Benefits of MS-Apriori 14](#_Toc96369152)

[1.9 MS-Apriori example 14](#_Toc96369153)

[1.10 Mining class association rules 15](#_Toc96369154)

[Class Association Rule (CAR) 15](#_Toc96369155)

[Quiz 16](#_Toc96369156)

[Topic 2: Feature Engineering I 18](#_Toc96369157)

[Multicollinearity 18](#_Toc96369158)

[2.2 Correlation and multicollinearity 18](#_Toc96369159)

[2.3 Measuring multicollinearity 19](#_Toc96369160)

[Principal component analysis 20](#_Toc96369161)

[2.4 Principal component analysis 20](#_Toc96369162)

[Summary 22](#_Toc96369163)

[2.5 Implementing a PCA analysis from first principles 23](#_Toc96369164)

[Independent component analysis 23](#_Toc96369165)

[2.6 Independent component analysis 23](#_Toc96369166)

[PCA Vs ICA 24](#_Toc96369167)

[Multiple Correspondence Analysis 25](#_Toc96369168)

[2.7 Multiple correspondence analysis 25](#_Toc96369169)

[Multiple Correspondence Analysis (MCA) 27](#_Toc96369170)

[2.8 Multiple Correspondence Analysis Example 27](#_Toc96369171)

[Factor analysis of mixed data 27](#_Toc96369172)

[2.9 Factor analysis of mixed data 27](#_Toc96369173)

[Quiz 28](#_Toc96369174)

[Topic 3: Feature Engineering II 29](#_Toc96369175)

[Random Forest 29](#_Toc96369176)

[3.2 Random forest feature selection 29](#_Toc96369177)

[Neural Networks 30](#_Toc96369178)

[3.3 Neural Networks 30](#_Toc96369179)

[Autoencoders 31](#_Toc96369180)

[3.4 Introduction to autoencoders 31](#_Toc96369181)

[3.5 Sparse autoencoders 32](#_Toc96369182)

[3.6 Convolutional autoencoders 33](#_Toc96369183)

[3.7 Variational autoencoding 34](#_Toc96369184)

[Quiz 37](#_Toc96369185)

[Topic 4: Cluster analysis 40](#_Toc96369186)

[Clustering, distance and similarity measures 40](#_Toc96369187)

[4.2 What is clustering? 40](#_Toc96369188)

[4.3 Similarity metrics 42](#_Toc96369189)

[Similarity 43](#_Toc96369190)

[Euclidean Distance 43](#_Toc96369191)

[4.4 Manhattan distance 43](#_Toc96369192)

[4.5 Cosine similarity 44](#_Toc96369193)

[4.6 Jacardian Index 45](#_Toc96369194)

[Similarity of asymmetric binary attributes 46](#_Toc96369195)

[Multinomial data 48](#_Toc96369196)

[Ordinal Data 48](#_Toc96369197)

[Ratio-Scaled Data 49](#_Toc96369198)

[Uses of Jaccard 51](#_Toc96369199)

[Example 51](#_Toc96369200)

[4.7 Gower's similarity coefficient 52](#_Toc96369201)

[What is gower distance? 52](#_Toc96369202)

[Ordinal and continuous variables 52](#_Toc96369203)

[Nominal Variables 52](#_Toc96369204)

[4.8 Clustering approaches 53](#_Toc96369205)

[4.9 K-means (Lloyd's algorithm) 54](#_Toc96369206)

[4.10 K-Medoids clustering 54](#_Toc96369207)

[4.11 DBSCAN 55](#_Toc96369208)

Topic 1 – Data integration

# Topic 1: Association rule mining

# Association rules

## 1.3 Introduction to association rules

Association rule mining is a rule-based machine learning technique that assists in the discovery of patterns in large scale databases.

The subject was proposed by Agrawal and Shim in 1993 and has been used extensively by the database and data mining community. **It assumes all the data are categorical and attempts to find relationships between categories or groups of categories**. It was initially used for market basket analysis with the intention of understanding customer buying patterns. So for example, if someone buys {*Potatoes*, *Onions*} then they are likely to buy Burgers. This is typically written as follows:

{*Potatoes*, *Onions*}  {*Burgers*}

and there is no importance to the order. So in this example, the order of Onions and Potatoes has no significance.

The idea behind this concept was to identify shopping patterns. Understanding these patterns allows supermarkets to control their sales and even gives them a handle on products they need to buy. This may seem strange, but you should all be familiar with supermarket pricing promotions. So, for example, you may be offered soft drinks at a knockdown price, but when you are in the supermarket the price of potato chips has remained the same. Supermarkets will have asked the suppliers of soft drinks and potato chips for a set schedule of promotions and nine out of ten times they will coordinate these items together. They do this so they can boost their profits on potato chips and increase their margin on high volumes. The suppliers of both products are happy as they will increase their sales and will only take a minor hit on their margin.

The concept has not just been applied to market basket analysis but also to web usage mining, intrusion detection, continuous production and bioinformatics.

Can you think of another example? Post these in the comments section below. I will point out that since the advent of online selling these assumptions are being seriously tested, so I suspect the high street retailers will have to change their sales mechanism to compete.

In the next step, we are going to look at a number of definitions and metrics that are used in association rule mining.

## 1.4 Rules and metrics definitions

In order to get used to the terminology and jargon that is used in association rule mining, we are going to have to outline a number of definitions that are typically used. We will then outline a few examples of each definition. Get used to these terms as quickly as you can because they will be used throughout this Topic.

### Definitions

#### Items I

* Items ***I***. For each transaction database, there is always a set of items that are finite (though they may be very large).

***I*** = { ***i1***, ***i2***,…, ***im***} where ***im*** is **an individual item** and ***I*** is the complete set of items.

#### Transaction t

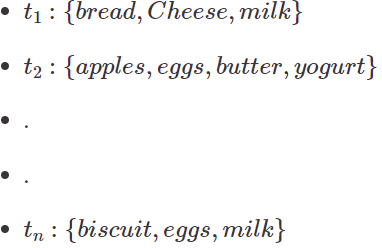
* ***t*** is **a set of items** and a subset of ***I*** 

### Transaction Database T

***T*** is a set of transactions or a Transaction Database.

* ***T*** = { ***t1***, ***t2***,…, ***tn***}

So if we are to relate these definitions to a supermarket’s data we would have the following market basket transactions:

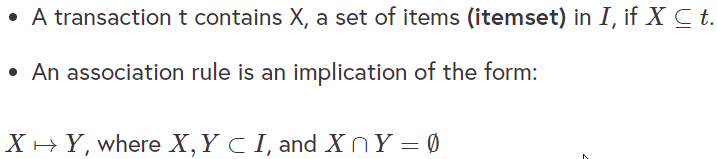


So we are now going to define some concepts.

Recap:

* ***i =*** An **item** ***i*** is an item or article in a basket, e.g. bread.
* ***I*** = the **set of all items** sold in the store.
* ***t*** = A transaction, a set of items purchased in the basket. It will most likely have a transaction ID (TID).
* ***T*** = A **transactional dataset** is a set of transactions.
* ***X*** = An **itemset**, e.g. {***bread*, *milk*, *cereal*}**, contained in a transaction ***t***.
* ***Y*** = An **itemset** we may associate with ***X***.
* ***X is a subset t is a subset of T***

So in association rule mining set theory definitions are used to describe the rule structure.



* An itemset is a set of items:

E.g. ***X*** = {***milk***, ***bread***, ***cereal***} is an itemset.

So we have now defined what a rule is but we need some metrics to assess a proposed rule:

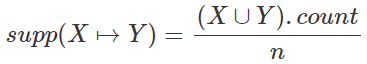
### Support

**IMPORTANT!** In association rule mining rules, the standard union symbol  really means intersection!

The support of X with respect to T is defined as **the proportion of transactions *t* in the dataset which contains the itemset X**:

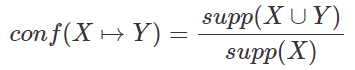


**Support is an indication of how frequently an item is bought**. The support count of an itemset ***X***, denoted by ***X.count*** in a dataset ***T*** is the number of transactions in ***T*** that contain ***X***, If we assume has n transactions then you can have support for a rule X Y and it can be defined as:



### Confidence

We can also define the confidence of a rule ***X***  ***Y***, with respect to a set of transactions ***T***, is the **proportion of transactions that contains *X* which also contains Y**. It can be defined as follows:

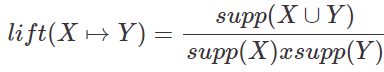


Note that ***supp(X ∪ Y)*** means the support of the union of the items in ***X*** and ***Y***. This is somewhat confusing since we normally think in terms of probabilities of events and not sets of items. We can rewrite ***supp(X ∪ Y)*** as the probability ***P(Ex ꓵ Ey)*** where ***Ex*** and ***Ey*** are the events that a transaction contains itemset ***X*** and ***Y***.

Thus confidence can be interpreted as an estimate of the conditional probability ***P(Ey*** \ ***Ex)*** the probability of finding the RHS of the rule in transactions under the condition that these transactions also contain the LHS, Hahsler(2005).

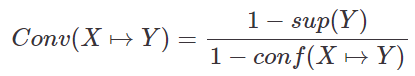
### Lift

Lift can be defined as the observed support in ratio to that if the events were statistically independent, i.e. **ratio of support Vs independence**.



### Conviction

Conviction can be interpreted as the ratio of the ***expected frequency that X occurs without Y***.

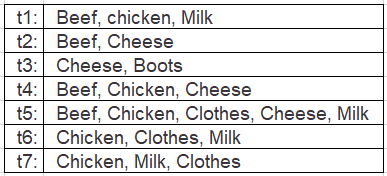


**Recap**

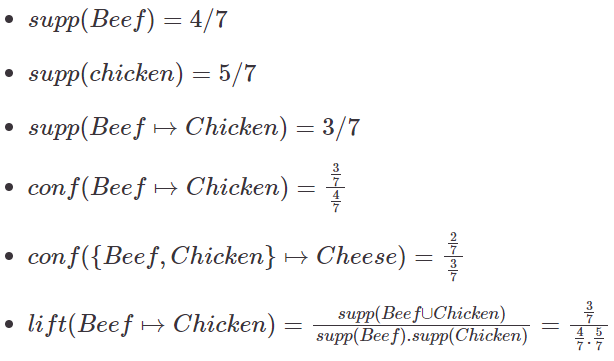
* **Support**  *How frequent is X?*
* **Confidence**  *What proportion of transactions contain both X and Y?*
* **Lift**  *Ratio of X and Y together Vs X and Y separate.*
* **Conviction**  *How often does X occur without Y?*

**Example**

We have now had 4 metrics that will allow us to make decisions about the relationship between rules. The following is a small transactional database:



Let’s do a few calculations to make sure it all makes sense:



Interpreting these results is a little tricky. **A lift > 1 indicates a positive dependence or substitution**. In other words, if we have *Beef* in our basket is there a likelihood that there will be *Chicken*. This term can be very useful as it vouches if there is a high(>1) or low (< 1) association between products. In our case, the Lift is 1.05 which implies there is an association between the 2 products. This type of metric will help store managers to make promotions and product placements.

Now can you calculate the following:



**Answer:** (2/7) / (2/7) = **1**

Towards Data Science have a nice link that will help you understand these terms a little better:

<https://towardsdatascience.com/association-rules-2-aa9a77241654>

## 1.5 Brute force approach

So now you should understand the background and some basic definitions to association rule mining. The next question is: how do we find the rules that are most useful to us?

Brute force approach

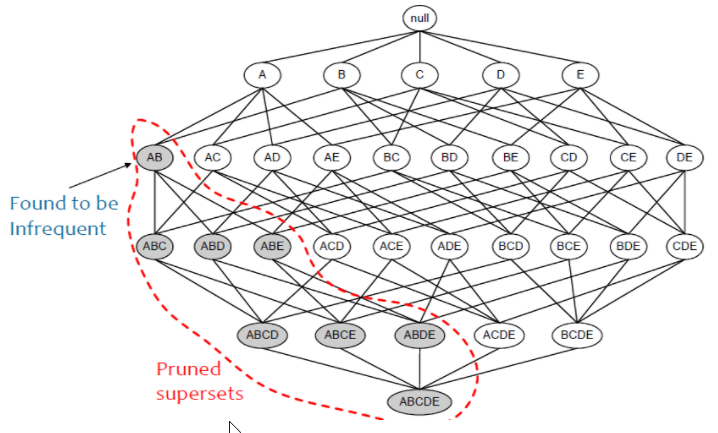
So now you should understand the background and some basic definitions to association rule mining. The next question is: how do we find the rules that are most useful to us?

There are a number of techniques that we will examine in this topic but to understand them we really need to examine the most basic approach which is known as the brute force approach. It is as simple as it sounds, we are going to extract every possible rule that exists. Imagine you have 5 products {A,B,C,D,E} in a virtual store and that your customers can take any combination of the 5 products. They could buy {A,B} or {B,D,E} for example. So if you wanted to find out every rule possible we would simply generate all the combinations of the list of products. In this case it would would come to:



combinations. Graphically this would look like the lattice laid out below:

We would have a total of 31 possible combinations of rules or list of potential baskets. There is a small piece of code which generates every possible combination of the virtual stores’ product list.



Follow this link to go to the Colab for this step:

<https://colab.research.google.com/drive/182GymbVjA9X5CZqQ6XdYz_q9JnpP9MsO#scrollTo=qk6_WWV5qtw->

## 1.6 Apriori algorithm

In the previous step, we examined the brute force algorithm and saw how even with a small number of products the potential number of baskets was very high.

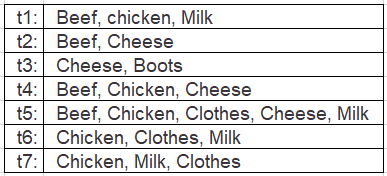
Processing the transaction database for each basket would take excessive amounts of time if we were to do so. The Ariori algorithm, Algrawal and Srikant (1984), is probably one of the best-known algorithms in association rule mining:

<http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.88.8047&rep=rep1&type=pdf>

The algorithm proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database, using effectively two steps. These steps are as follows:

1. Find all itemsets that have minimum support.
2. Use frequent itemsets to generate rules.

In the example below, we have seven transactions.

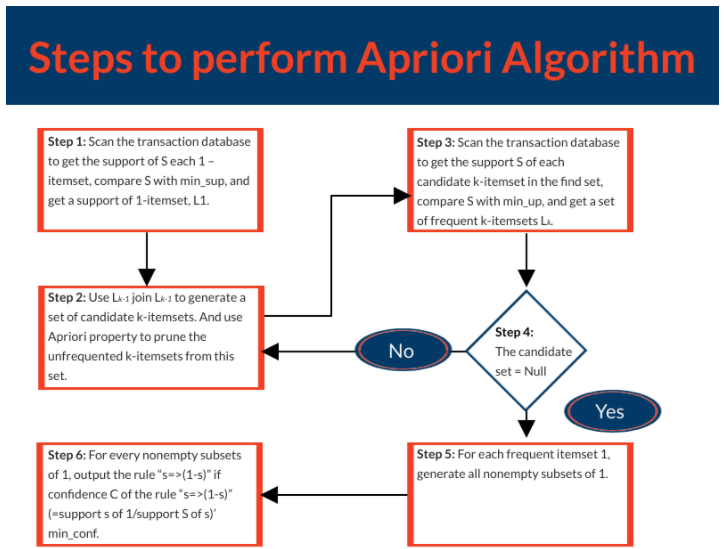


So we can see from this transactional database that:

*supp(Clothes  {Milk, Chicken}) = 3/7*

*conf(Clothes  {Milk, Chicken}) = (3/7) / (3/7) = 1*

which is a frequent itemset. But if we started with *Boots*, we would get support of 1\7 which means that we could not generate frequent itemsets from it. So there wouldn’t be any point starting with boots or in other words going down this part of the tree if we were looking for frequent itemsets. The Apriori algorithm can be explained in the with the following pseudo-code:



Let’s look at another example and examine how it works. See the table below.

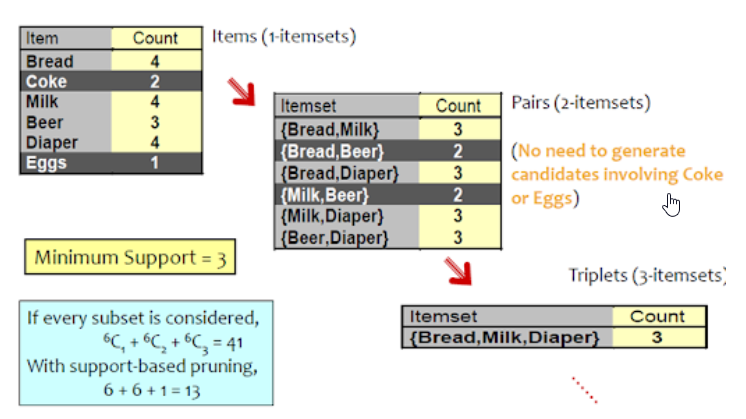


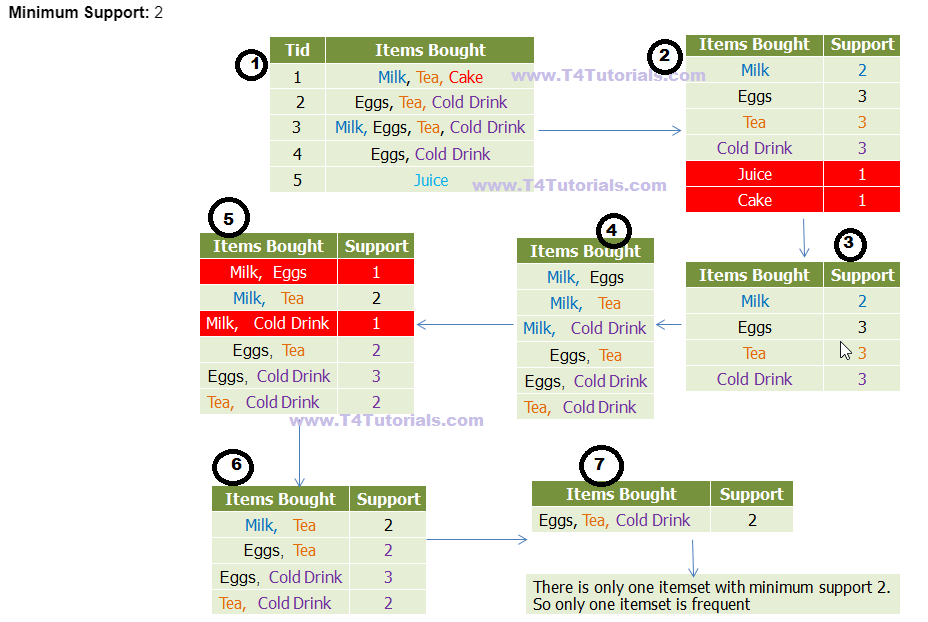
* So we have the following unique items in our database and they are:

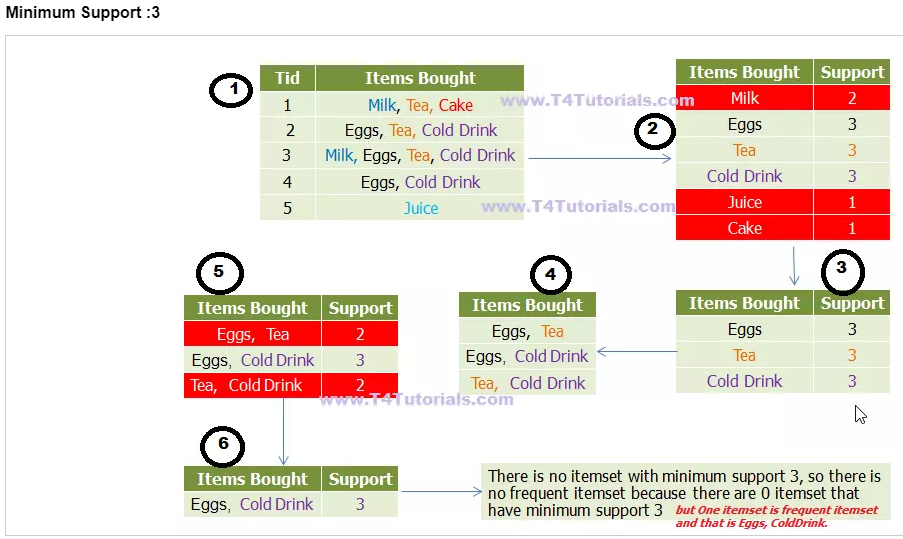
{Bread, Milk, Diapers, Beer, Eggs, Cola}

4 4 4 3 1 2

* We can see from the database that Milk occurs four times and Eggs only occurs once.
* If we were to examine each individual item and the items in couples, triples and so on we would get a pattern such as that shown below.
* Now notice how we have a term called Minimum support.







## 1.7 Apriori example

This step will provide a working example of the Apriori algorithm using Python.

The example we are going to use to demonstrate the Apriori algorithm comes from GeeksforGeeks:

<https://www.geeksforgeeks.org/implementing-apriori-algorithm-in-python/>

In it, we will pull down an online retail dataset and attempt to find the most frequent itemsets. I have concentrated the following code on UK sales as these possess the most transactions. Now you will notice that when I am importing the data into Google Colabs that I have used “ISO-8859-1” encoding.

You should also look at how the Apriori algorithm needs to be set up. Each product must be set up as a hot encoded column for each invoice (basket). This link should help you here:

<http://rasbt.github.io/mlxtend/api_subpackages/mlxtend.frequent_patterns/>

In the final rules dataset, you will see two columns: **antecedents** and **consequents**. That might need some explanation. In the previous steps, we referred to them as ***X  Y***.

* Antecedent = ***X***
* Consequent = ***Y***

The remaining columns describe the metrics that we have explained in Step 1.4. Experiment with the data and see if it all makes sense.

You can access the Google Colab file here:

<https://colab.research.google.com/drive/1_0I2s5-69QwOVnakzPoWm1D5xcWdnq6J#scrollTo=YOuknZJZMliT>

This video explains the above code:

[VIDEO:] <https://www.youtube.com/watch?v=guVvtZ7ZClw>

## 1.8 MS-Apriori algorithm

The Apriori algorithm that we looked at in Step 1.6 assumed all items had the same minimum support and that all items have a similar frequency.

In reality, this is not the case as some items appear very frequently in our data, while others rarely appear. For example, in a supermarket, people buy food processors and cooking pans much less frequently than they buy bread and milk. If the frequencies of items vary a great deal, we will encounter two problems. These are as follows:

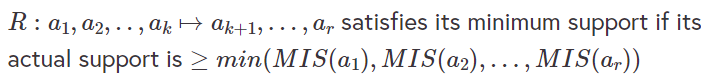
1. If minsup is set too high, those rules that involve rare items will not be found.
2. To find rules that involve both frequent and rare items, minsup has to be set very low. This may cause a combinatorial explosion because those frequent items will be associated with one another in all possible ways.

You may ask yourself why would you care if something is a rare item. Well if any of you shop in Lidl or Aldi then you will have noticed their promotions on rare items such as tents or wet suits. These supermarkets know what items will be sold with these items and this knowledge allows them to plan their marketing strategy. People won’t buy these things every week but the supermarket knows that if they do a promotion on these items then associated items will be sold as well.

So the answer to this problem is to have **multiple minimum supports**. The minimum support of a rule is expressed in terms of minimum item supports (MIS) of the items that appear in the rule. Each item can have a minimum item support, and by providing different MIS values for different items, the user effectively expresses different support requirements for different rules.

So let’s go a little deeper:

Let ***MIS(ai)*** be the the MIS value if item ***i***. The minsup of a rule ***R*** is the lowest value of the items in the rule. In other words a rule



Let’s look at an example; consider the following items:

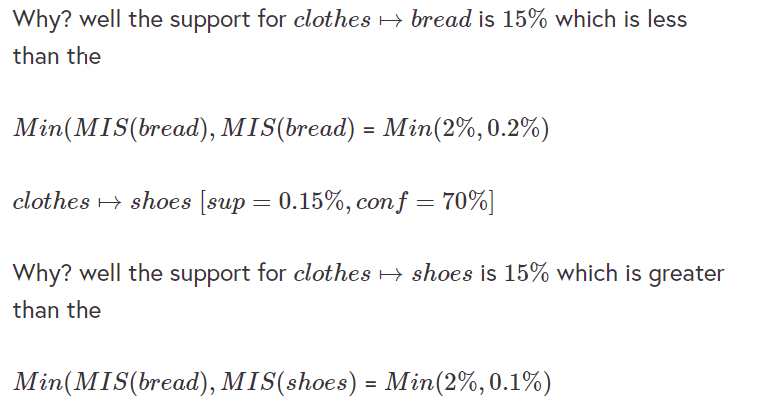
[***bread, shoes, clothes***]

The user-specified MIS values are as follows:

* ***MIS(bread) = 2%***
* ***MIS(shoes) = 0.1%***
* ***MIS(clothes) = 0.2%***

The following rule doesn’t satisfy its minsup:





### Benefits of MS-Apriori

The benefits of MS-Apriori are:

* Multiple minsup model subsumes the single support model.
* It is a more realistic model for practical applications.
* The model enables us to found rare item rules yet without producing a huge number of meaningless rules with frequent items.
* By setting MIS values of some items to 100% (or more), we effectively instruct the algorithms not to generate rules only involving these items.

I am not going to describe the algorithm in detail, however, for those that are interested, you can download my slides for association rule mining here:

<https://www.computing.dcu.ie/~amccarren/mcm_slides/Lecture_6_association_rules.ppt>

Now let’s have a look at a working example in Python in the next step.

## 1.9 MS-Apriori example

The step demonstrates the MS-Apriori algorithm.

The MS-apriori example I am going to show you here was written by Bing Liu and is located on Github. Additionally, this code is not supported by Python 3.7 so you will have to set the runtime to Python 2 in the options above.

<https://github.com/sachinbiradar9/MS-Apriori>

Normally, I would use the Google Drive ID’s to reference files, but in this situation I want you to follow the instructions exactly, so I have mounted Google Drive as a virtual drive in the following Google Colab available here:

<https://github.com/sachinbiradar9/MS-Apriori>

There are other versions of this online and you may find this one more useful.

<https://github.com/pavvu/FindFrequentItemSetWithMultipleItemSupport/blob/master/README.md>

Now you will also have to download the transaction.txt and parameter.txt files and store them on your Google Drive, or where ever you want to run them from.

<https://drive.google.com/open?id=1RyPO44xoEf60CSi1OmhbRSRSAOeA3hEl>

<https://drive.google.com/open?id=1kee1SdYZLVbBqiA3Qz6GvtPa0iJOIBq->

## 1.10 Mining class association rules

Normal association rule mining does not have any target, as it finds all possible rules that exist in data. For example, any item can appear as a consequent or a condition of a rule. However, in some applications, the user is interested in some targets. For example, the user has a set of text documents from some known topics. He/She wants to find out what words are associated with or correlated with each topic.

### Class Association Rule (CAR)

Graphical user interface, text

Description automatically generated

**An example**

Let’s look at a text document data set. In it we have documents that have keywords in them and each set of words is categorised with a label becomes:

Table

Description automatically generated

In this example, we want to examine the support and confidence for a group of words.

Text

Description automatically generated

Unlike normal association rules, CARs can be mined directly in one step.

The key operation is to find all rule items that have support above minsup. A rule item is of the form:



Each rule item basically represents a rule:



The Apriori algorithm can be modified to generate CARs. The multiple minimum support idea can also be applied here. The user can specify different minimum supports to different classes, which effectively assign a different minimum support to the rules of each class. For example, we have a data set with two classes, Yes and No. We may want rules of class Yes to have the minimum support of 5% and rules of class No to have the minimum support of 10%. By setting minimum class supports to 100% (or more for some classes), we tell the algorithm not to generate rules of those classes. This is a very useful trick in applications.

Can you think of any other examples that we could use CARs for? Let us know in the comments section below.

# Quiz

**Question 1 –** In association rule mining I = {i1, i2, …, im} is known as a set of:

* Rules
* Items
* Objects

**Question 2 –** Complete the following statement correctly.

A transaction database is a set of:

* All items.
* All transactions.
* A subset of all transactions.

**Question 3 –** Shopping Baskets

t1: Beef, chicken, Mil t2: Beef, Cheese t3: Cheese, Boots t4: Beef, Chicken, Cheese t5: Beef, Chicken, Clothes, Cheese, Milk t6: Chicken, Clothes, Milk t7: Chicken, Milk, Clothes

Calculate the support for Beef and Chicken. Select the correct answer below:

* 50%
* 28.5%
* 42.85%

**Question 4 -** Shopping Baskets

t1: Beef, chicken, Mil t2: Beef, Cheese t3: Cheese, Boots t4: Beef, Chicken, Cheese t5: Beef, Chicken, Clothes, Cheese, Milk t6: Chicken, Clothes, Milk t7: Chicken, Milk, Clothes

Which item set has a support of greater than 30% on the shopping baskets?

* Beef, Chicken, Cheese
* Chicken, Milk, Cheese
* Chicken, Clothes, Milk

**Question 5 –** What is the brute force technique in association rule mining?

* Include nearly all the leaves.
* Generate all combinations and possible transactions.
* Select those transactions with support greater than minsup.

**Question 6 –** The MS-Apriori has only one MINSUP.

* Is this statement true or false?
* True
* False

Topic 2: Feature Engineering I

# Topic 2: Feature Engineering I

# Multicollinearity

## 2.2 Correlation and multicollinearity

Correlation and multicollinearity

In steps 3.12 In Introduction to Data Analytics and in Step 4.9, in Pre-processing Data and Feature Impact Calculation, we discussed correlation and multi-collinearity.

**Multi-collinearity can be defined as a phenomenon in which one predictor variable in a model can be linearly predicted from the remaining variables.**

I appreciate that most of you will feel we are repeating ourselves. However, the impact and conclusions drawn from proposed algorithms can be extremely important if Multi-Collinearity exists in the data. Now many of you might recap what was said in Step 3.12 (mentioned above) about the effect of Multicollinearity can have on our models:

* In linear regression, it potentially inflates the standard error of some of the parameter estimates.
* When this happens we are inclined to drop variables that are relevant and keep irrelevant variables.

Many of you will say why is this such a big deal, as all we are interested in is making predictions. In theory, this is true. Multi-collinearity doesn’t affect the predictions. But understanding which variables are relevant to our analysis is extremely important. We will now discuss reasons as to why we should deal with it below.

Under perfect multicollinearity, Ordinary Least Squares (OLS) estimates simply don’t exist. In OLS the solution we are trying to find the best estimates of to the following equation:



The solution to this problem can be shown to be:



Please note that we have \((X^TX)^{-1\) in this equation. This means that we have to be able to invert the . If any of the independent variables are perfectly collinear with others, then this matrix will not be invertible as the rank of the matrix will be less than the number of dimensions+1.

Many of you might think that the variables will not be perfectly correlated, and you might be right here. Well **if there is a “strong” but not perfect collinearity between variables then you probably will find that your model will be quite sensitive to minor changes in the data**. This is caused by ill-conditioning and many libraries will struggle to converge. Even if you get your library to work it will most likely come up with the volatile weights which will cause errors with regard to variable importance.

There will be those among you who will think that you are going to use a non-parametric model such as a neural network and this issue won’t matter. Well, I am sorry to burst your bubble, but **neural networks will still have issues with multicollinearity because they use algorithms such as steppest descent to optimize the cost function**. These algorithms struggle to optimize when there are non-orthogonal features in the cost function (perfect orthogonality effectively means we have no multi-collinearity in our input features/weights). The net effect is we may not converge or in some cases, we will end up in a nonlinear “hole”, thus we are less likely to find the global optimum.

Finally, you will come across a lot of material about multi-collinearity on-line. From a data mining perspective, it is very important to understand the amount of multi-collinearity in our data as it will allow us to understand which variables are important in our models and to determine the impact that these variables have with our outcome variables. One should note that in clinical trials and loan approval models authorities such as the FDA or central banks insist on understanding the impact of each variable.

Now were are going to implement the Variance Inflation Factor (VIF) in the next step using the Python “statsmodels” library.

## 2.3 Measuring multicollinearity

In the Google Colab file in Step 3.13 in the first course in this program, we have shown how **Variance Inflation Factor (VIF) can be used to measure multicollinearity** in a feature dataset:

**See Colab M1.3.13**

VIF is calculated by regressing an independent variable against all the other variables and using the following formula:



We then get the from each variables regression model and then calculate the VIF using the following equation:



**Recap:**

**R-squared is a goodness-of-fit measure for linear regression models**. This statistic indicates the percentage of the variance in the dependent variable that the independent variables explain collectively. After fitting a linear regression model, you need to determine how well the model fits the data.

* If your VIF factor is **>10** then you really need to **drop variables** from your model,
* If it is between **5-10** then you need to **consider it**.

Let’s look at the Boston housing data from Scikit Learn in the Google Colab for this step is here.

<https://colab.research.google.com/drive/1EeLbLtxEDobwYJaOWS_-Gdk1DC_ixV1H#scrollTo=cDnQ9YJPz6qo>

# Principal component analysis

## 2.4 Principal component analysis

In Step 4.9, in Preprocessing Data and Feature Impact Calculation, we introduced the concept of principal component analysis (PCA) in order to deal with dimensionality reduction.

In this step, we will expand on our discussion of the working of PCA and explain how it can also be used to create new features. We will also explain the background to PCA and what an eigenvalue or eigenvector is as well as how they relate to principal components.

You might ask what is so important about PCAs. Well, they can help us accomplish the following:

* Data visualization
* Data reduction
* Data classification
* Trend analysis
* Factor analysis
* Noise reduction

**PCA is the most common form of factor analysis. Factor analysis is a process where we try to impute latent variables or variables that are not directly observable from our data.**

So let’s try and explain in simple terms what we are trying to do with PCA**. In PCA we are trying to create a new set of variables that explain the variation in the original dataset subject to the constraint that they are orthogonal (not correlated) to each other.** This is an easy concept to understand but the maths behind it is a little tricky.

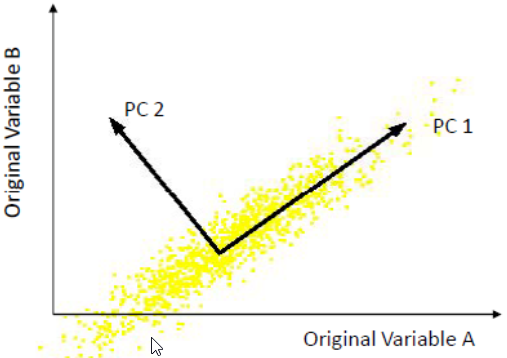
**Recap:**

* **Orthogonal  Uncorrelated**: An orthogonal model means that all independent variables in that model are uncorrelated.
* **Non-orthogonal  Correlated**: If one or more independent variables are correlated, then that model is non-orthogonal.

So imagine we have a set of variables:



Now we are going to find a transformation that creates a new axis that describes the greatest variance in the data subject to the condition that each new variable is orthogonal. We can see in the figure below that we have created a new axis ***PC1*** and ***PC2*** which are orthogonal (perpendicular).

******

So lets go back a little and do a recap on what a covariance matrix is. The covariance between two variables and can be given by the following equation:



now if we center each variable around their means, with the following transformation:

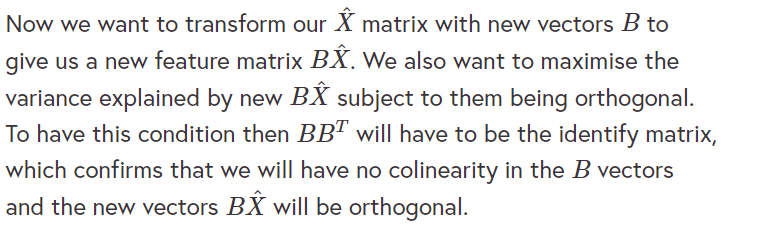


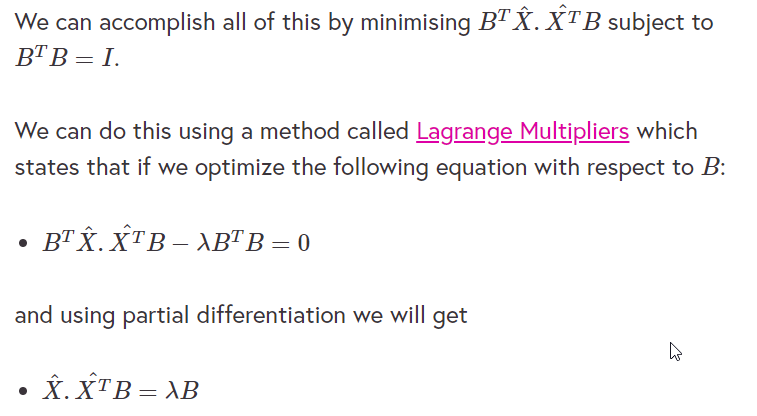
then the covariance between and  can be described as follows:



Now if we center all the variables in ***X*** above we can describe the covariance matrix as follows:

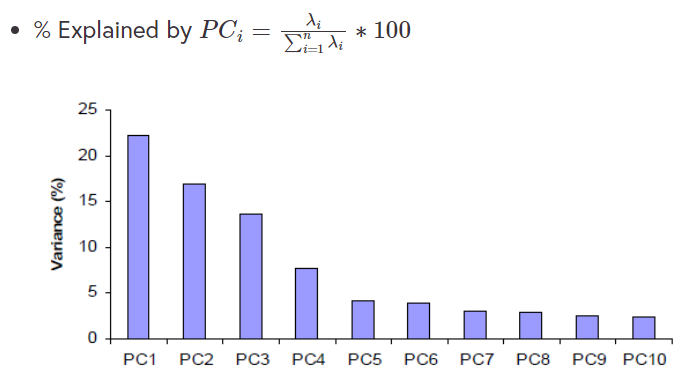






<http://mathonline.wikidot.com/the-method-of-lagrange-multipliers>

Those of you who remember a little linear algebra will notice that this equation is the same as the derivation of two terms known as the **eigenvalue**  and the **eigenvectors** ***B*** of the covariance matrix . Now the great thing about all of this is the largest eigenvalue and the eigenvector corresponding to it multiplied by the  matrix will give you the **first component or the component that describes the most variation and the second component describes the second highest amount of the variation, and so on**. I have regularly found datasets with 50 or more variables where the first component accounts for over 50% of the overall variation within the data. You will also find that the proportion of each eigenvalues to the total sum of the eigenvalues gives you the % explained by each component or:



You can see from the figure above that the amount of variance explained diminishes with the number of components. You do lose some information, but if the eigenvalues are small, you don’t lose much. So if you start with ***n*** dimensions you will probably end up with considerably less as a small number of components can explain the variance of a high dimensional dataset.

We mentioned this in the previous course, Preprocessing Data and Feature Impact Calculation, but I will repeat it again. **Don’t use PCA on categorical data.** It really is not appropriate. You can use a technique called multiple correspondence analysis to do this and we will look at this in Step 2.7 of this topic.

If you are finding this a little difficult, have a look at a tutorial from Lindsay Smith or this one from medium.com by Rishav Kumar:

<http://www-labs.iro.umontreal.ca/~pift6080/H09/documents/papers/pca_tutorial.pdf>

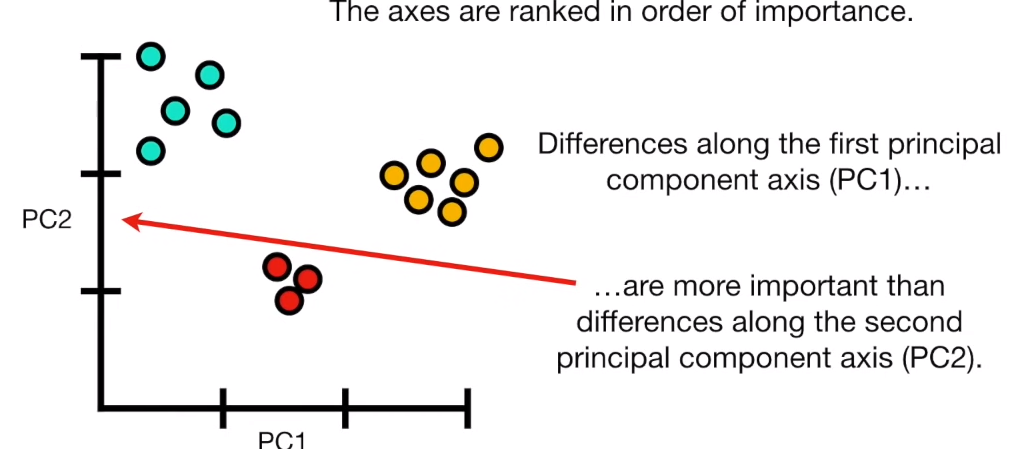
<https://medium.com/@aptrishu/understanding-principle-component-analysis-e32be0253ef0>

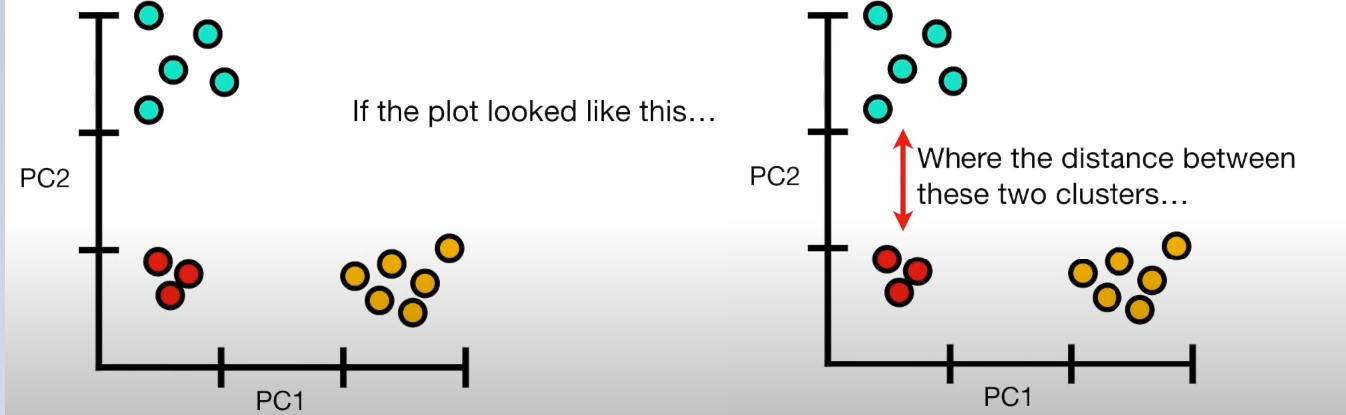
We will now look a very simple example of how we calculate the PCA’s in the next step.

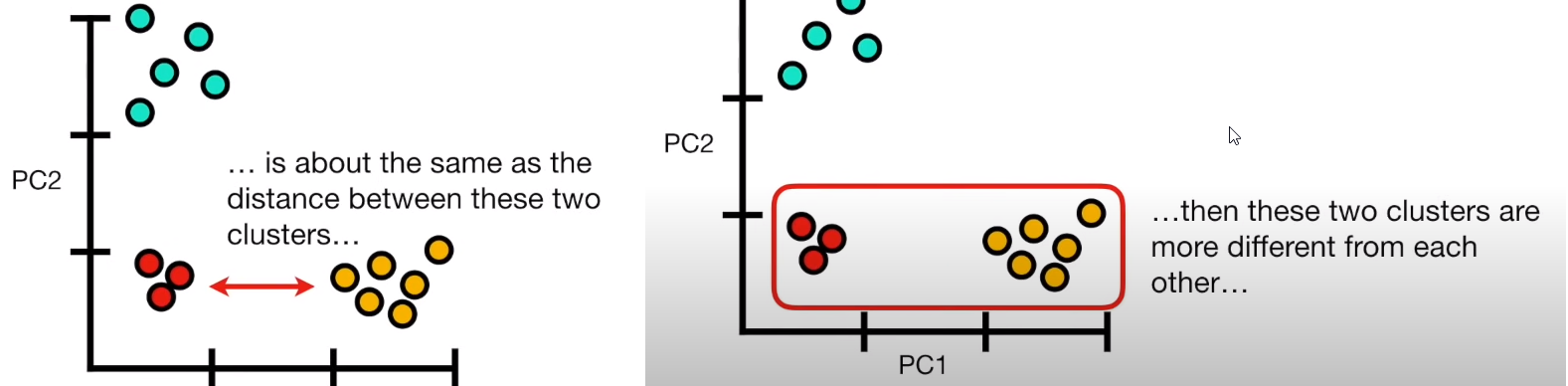
### Summary

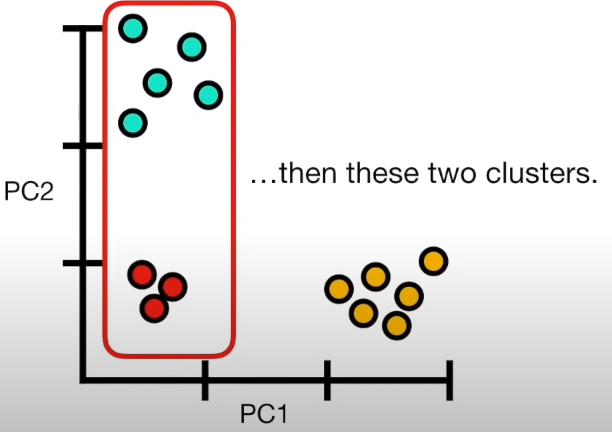
PCA effectively clusters data ranking in descending importance from most significant:

**PC1, PC2, PC3… PCn**









## 2.5 Implementing a PCA analysis from first principles

We are now going to complete an example of how you would implement the steps of a principal component analysis (PCA). There are routines in Python or R that would allow you to do it automatically, but working it out from first principles is a really good exercise.

Let’s get started by following this link to go to the Google Colab for this step:

<https://colab.research.google.com/drive/1VEelHRZwNpDp_1yqbJHQoMUD6lLzt6Hs#scrollTo=dyLOHeFPEYz9>

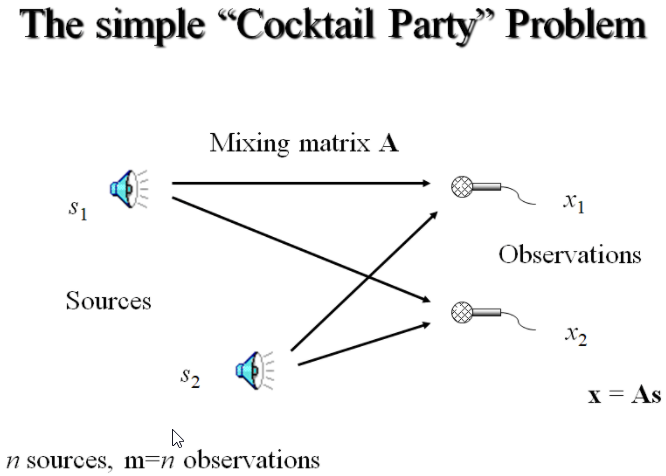
# Independent component analysis

## 2.6 Independent component analysis

Imagine you are a spy at an embassy cocktail party and in attendance are various VIPs and diplomats.

Now, there are two major groups at the party, the first group is the Americans and the second is the Russians. If you were trying to record these conversations with one microphone you would get extremely distorted sound quality. You may think I can just put two microphones in the room and you would be correct, but you will still get levels of noise from each conversation. An additional problem is that you will also not know where to put the microphones exactly.

You can see in Fig 1 below that each source is mixed together and the outputs in ***s1*** and ***s2*** and mixed by some matrix A. So this is where Independent Component Analysis (ICA) comes in. It helps filter conflicting sources into their original source.



**ICA (also known as Blind Signal Separation) allows us to identify & separate a mixtures of sources with little prior information.** There are many applications and these include the following:

* Audio Processing
* Medical data
* Finance
* Array processing (beamforming)
* Coding
* And most applications where Factor Analysis and PCA is currently used

### PCA Vs ICA

* **PCA**  seeks directions that represents data best
* **ICA**  seeks directions that are mostly independent from each other

In this step, We are going to rely on notes written by Klein Carsten:

<https://github.com/akcarsten/Independent_Component_Analysis>

I, however, use Sklearn FastICA instead of Klein’s method, as Klein’s version is quite long. But, I would suggest you implement and compare the results.

As we saw from fig 1, ICA assumes we generate the observed data from an underlying unknown independent process ***x*** with mixing signals ***A***.

You can access the Google Colab for this step by following this link:

<https://colab.research.google.com/drive/1IZcvCMklk2QOQSeNby0Y2fvviF_S4dux#scrollTo=zDECGFm-Jor->

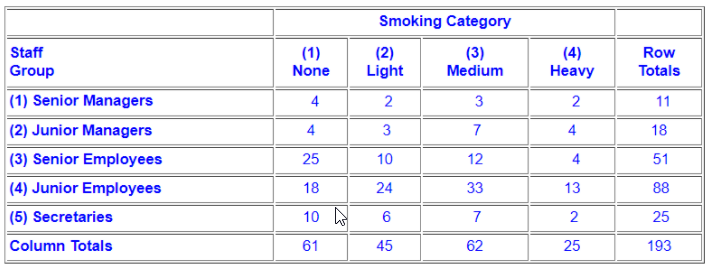
# Multiple Correspondence Analysis

## 2.7 Multiple correspondence analysis

Multiple correspondence analysis

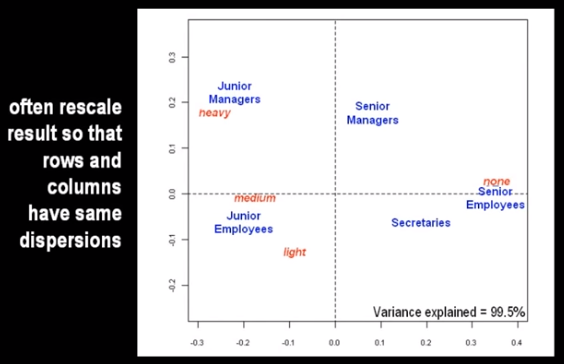
In this step we will discuss correspondence analysis.

When we have contingency tables such as the one shown in Table 1 below we are not just interested in the relationship between the two factors but also the inherent components that exist within it. The values that are shown in a contingency table are cross-tabulations of dummy variables that have been derived from categorical factors.



Now you might remember we used the  distribution on contingency tables to determine if there was a relationship between two factors such as those given in table 1, but it did not tell us what the components are, or the contribution that explained the relationship between rows. Recall that we used principal component analysis on continuous variables to both reduce the dimensionality, handle correlation and tell us the contribution each row had to a factor (eigenvectors). **Correspondence analysis (Multi) can be described as a descriptive/exploratory technique designed to analyze simple two-way and multi-way tables containing some measure of correspondence between the rows and columns**. The results allow one to **explore the structure of categorical variables** included in the table.





**Definition:**

**Correspondence analysis (CA)** is a multivariate statistical technique. It is conceptually similar to principal component analysis, but **applies to categorical rather than continuous data**.

In a typical correspondence analysis, a crosstabulation table of frequencies is first standardized, so that the relative frequencies across all cells sum to 1.0. One way to state the goal of a typical analysis is to represent the entries in the table of relative frequencies in terms of the distances between individual rows and/or columns in a low-dimensional space. This is best illustrated by a simple example, which will be described below. There are several parallels in interpretation between correspondence analysis and factor analysis, and some similar concepts will also be pointed out below.

For a comprehensive description of this method, computational details and its applications refer to the classic by Greenacre (1984). These methods were originally developed primarily in France by Jean-Paul Benzécri in the early 1960s and 1970s (see Benzécri, 1973).

### Multiple Correspondence Analysis (MCA)

**Multiple correspondence analysis (MCA)** originates from correspondence analysis. In correspondence analysis, we predominantly work with two factors but **this is extended to multiple factors** in MCA.

In the next step, we will demonstrate how MCA can be implemented on a toy dataset. Another, more complex example can be found here:

<http://vxy10.github.io/2016/06/10/intro-MCA/>

This example uses a 3-dimensional categorical table and also calculates all the measurements from scratch and uses the Python MCA Library.

## 2.8 Multiple Correspondence Analysis Example

In the example provided in this Google Colab:

<https://colab.research.google.com/drive/13xofqepzkAf7z4GirICWDVubIV8YiIZW#scrollTo=_OEIXGQ4vDC3>

we are using a toy dataset known as the balloons dataset:

<https://drive.google.com/open?id=1HXSdDFY9ZUPZTbMNUrZlrOJLeOV8JvK7>

which was taken from UCI datasets:

<https://archive.ics.uci.edu/ml/index.php>

This dataset follows the most common format for categorical variables.

Please post any questions you may have about the MCA example in the comment section below.

# Factor analysis of mixed data

## 2.9 Factor analysis of mixed data

In the previous four steps, we have examined how we can use principal component and multiple correspondence analysis to help us search for latent factors for both continuous and categorical data respectively. The problem we have is that most of our datasets have both categorical and quantitative data.

Factor Analysis of Mixed Data (FAMD) is a method dedicated to exploring data with both continuous and categorical variables. It can be seen roughly as a mix between PCA and Multiple Correspondence Analysis (MCA). More precisely, the continuous variables are scaled to unit variance and the categorical variables are transformed into a set of dummy variables. They are then scaled using the specific scaling of MCA. This ensures to balance the influence of both continuous and categorical variables in the analysis. It means that both variables are on an equal foot to determine the dimensions of variability. This method allows one to study the similarities between individuals taking into account mixed variables and to study the relationships between all the variables. It also provides graphical outputs such as the representation of the individuals, the correlation circle for the continuous variables and representations of the categories of the categorical variables. It also provides specific graphs to visualize the associations between both type of variables, RDocumentation:

<https://www.rdocumentation.org/packages/FactoMineR/versions/2.0/topics/FAMD>

Now the issue we have is that the libraries that are available in Python are not as well established as those provided by R, the statistical programming language. You can find out how to install R here:

<https://cran.r-project.org/bin/windows/base/>

and this video shows you how to do FAMD using the R FactoMineR library:

<http://www.sthda.com/english/articles/22-principal-component-methods-videos/72-famd-in-r-using-factominer-quick-scripts-and-videos/#course-video>

There is an implementation of a FAMD using both Python “light-FAMD” and “prince”:

<https://pypi.org/project/light-famd/#factor-analysis-of-mixed-data-famd>

You can access the Google Colab for this step by following this link here:

<https://drive.google.com/open?id=1DlT9WF3ZbDY3AO3bmursO18jLk2WceS->

# Quiz

**Question 1 –** From the list below which is the best metric to measure multicollinearity?

* Pearson correlation coefficient.
* Coefficient of determination.
* Shapiro Wilks.

**Question 2 –** Consider the following statement:

If you have a dataset with multi-collinearity then it is OK to use the estimates of the features to help determine the influence of each variable. Is this statement true or false?

* True
* False

**Question 3** – Consider the following statement:

Independent component analysis assumes that the original signal sources come from non-Gaussian signals. Is this statement true or false?

* True
* False

Topic 3: Feature Engineering II

# Topic 3: Feature Engineering II

# Random Forest

## 3.2 Random forest feature selection

In this step, we are going to outline how the random forest machine learning approach can help in determining the importance of a feature.

Random forests are one of the most popular machine learning algorithms. They are so successful because, in general, they provide:

1. good predictive performance,
2. low overfitting,
3. easy interpretability.

**This interpretability is given by the fact that it is straightforward to derive the importance of each variable on the decision tree**. In other words, it is **easy to compute how much each variable is contributing to the decision**.

In the Scikit-learn random forest library, the relative rank (i.e. depth) of a feature used as a decision node in a tree can be used to assess the relative importance of that feature with respect to the predictability of the target variable. Features used at the top of the tree contribute to the final prediction decision of a larger fraction of the input samples. The expected fraction of the samples they contribute to can thus be used as an estimate of the relative importance of the features. For further reading on this please follow this link:

<https://scikit-learn.org/stable/modules/ensemble.html#forest>

Random forests are also incredibly robust and generally are very easy to implement. In every project I give, students always use random forests for predictions. Lately, I have also found that the students use them for dimensionality reduction. However, there is an issue with them that Sebastian Raschka explains really nicely below:

“*The random forest technique comes with an important gotcha that is worth mentioning. For instance, if two or more features are highly correlated, one feature may be ranked very highly while the information of the other feature(s) may not be fully captured. On the other hand, we don’t need to be concerned about this problem if we are merely interested in the predictive performance of a model rather than the interpretation of feature importances.”* - Python Machine Learning by Sebastian Raschka.

**This** “gotcha” is really important to understand and in a way **negates the use of random forests for feature importance and thus the use of them for feature reduction and engineering**.

Have a look at the code below and play about with the standardisation methods from Scikit learn. The description of how the random forest classifier works can be found here.

Please follow this link to go to the Google Colab for this step:

<https://colab.research.google.com/drive/1exeOqpBaLZ1SjbiedkcENx8uiRiwcS5r#scrollTo=UsIyHV_L6_7r>

# Neural Networks

## 3.3 Neural Networks

In this step, we will explore Neural Networks.

In previous courses, **we learned how to use principal component analysis (PCA), Lasso and Ridge regression to reduce the dimensionality of datasets**. One of the underlying assumptions with these methods is that the compression or dimensionality reduction is **linear in nature**. PCA effectively creates new latent variables that are new representations of the original data but of a significantly lower dimension. These techniques can be very useful when dealing with variables that are highly correlated. They can also be useful when you just want to simply reduce the volume of your input variables in order to improve the processing speed of your algorithms.

So by now, you may have come across the concept of Neural Networks. If you haven’t then had a look at this site:

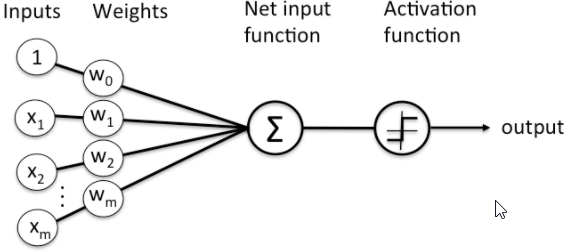
<https://pathmind.com/wiki/neural-network#define>

Neural Networks are algorithms that are loosely modelled on the human brain. The concept can be best described in Figure 1 below.



You will definitely come across neural networks in other courses. They can be used to build prediction models. They can also be used as part of autoencoder models to reduce datasets and remove noise from our datasets.

The principle behind them is that the input or our data is passed into the system through the input nodes. An input function is applied and this information is then transferred to the hidden nodes, where an activation function transfers that information to the output function, see Figure 2.



The objective is to minimize the difference between the outputs and the results of the activation function, by optimising the input and hidden node weights. Normally, optimization is done using algorithms such as gradient descent. Neural Networks will introduce a level of non-linearity and complexity into our data models, thus allowing for greater predictive power. The architecture of these systems can vary with the number of nodes in the hidden layer and the extension to deep learning architectures can be achieved through the addition of multiple hidden layers. In reality, this process will be equivalent to a linear regression model if we remove the hidden nodes and apply a linear activation function.

Now we are going to implement a simple neural network regressor on the Boston housing dataset. It is important that you understand this simple example as it will help you build an autoencoder in the next step.

# Autoencoders

## 3.4 Introduction to autoencoders

In the previous step, we described how methods such as:

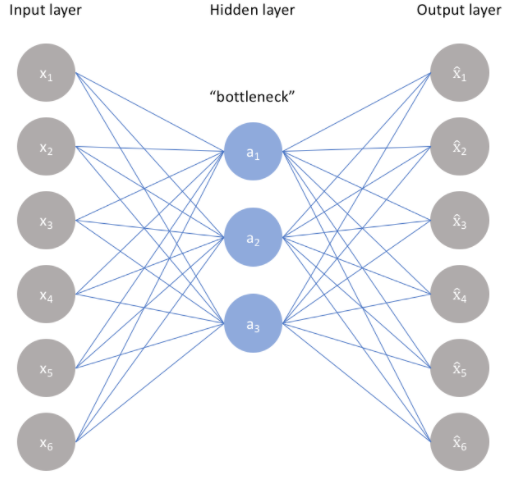
* neural networks, linear regression  can be used to make **predictions**
* principal components  can be used to **reduce dimensionality** and help **remove noise**.

**Autoencoders are a form of unsupervised technique which use neural networks:**

* To help **reduce dimensionality** or **remove noise** by compressing our input data through a hidden layer back to an original representation of the input layer.
* We leverage neural networks for the task of **representation learning**.

See: <https://towardsdatascience.com/deep-inside-autoencoders-7e41f319999f>

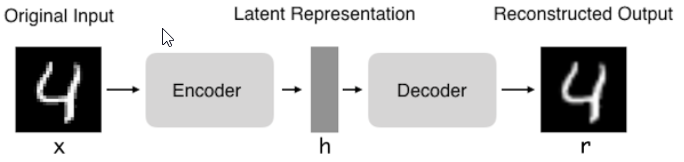
Figure 1 shows how the network decompresses the data before recreating it in the output layer.



See: <https://www.jeremyjordan.me/autoencoders/>

We examine this process a little more by looking at Figure 2 where we see there is an “Encoder” and “Decoder” used to create “h” the latent representation. The latent representation can then be used as a decompressed or dimensionally reduced dataset.

**Latent representation learning (LRL)**, or latent variable modeling (LVM), is a machine learning technique that attempts to infer latent variables from empirical measurements. Latent variables are variables that cannot be measured directly and therefore have to be inferred from the empirical measurements.



**Autoencoders are only useful when there are relationships between the input variables**, as mathematically it is difficult to compress the features without substantial loss of information if there was no correlation between the variables. However, if there is a relationship then the structure can be learned and we can effectively compress our data into a hidden layer. The outcome variable is a transformed version of the hidden nodes. Thus the output variables can be considered to be denoised.

Now we have outlined that autoencoders can be used for dimensionality reduction as **we will be creating a lower number of latent variables than the number of original input variables**. This is similar to principal component analysis (PCA). So what are the differences?

Differences between autoencoders and PCA:

* PCA is a lineartransformation of the data and assumes the new latent variables are a linear combination of the original variables.
* PCA features are not linearly correlated but autoencoders might have correlations.
* PCA is less computationally intensive than autoencoders.
* A single-layered autoencoder with a linear activation function is very similar to PCA.
* Autoencoders may require regularization as they are prone to overfitting.

Please go to this link to access the Google Colab file for this step:

<https://drive.google.com/open?id=1kcgQt7ucinl5DigW-_Iob755XIHuh73P>

## 3.5 Sparse autoencoders

In the previous step, the input layer was constrained by the size of the hidden layer (128). In such a situation, what typically happens is that the hidden layer is learning an approximation of principal component analysis(PCA).

Another way to constrain the representations to be compact is to add a sparsity constraint on the activity of the hidden representations, so fewer units would “fire” at a given time. In Keras:

<https://keras.io/>

this can be done by adding an activity\_regularizer to our Dense layer. The code below outlines how this works. In particular, look at the line in the code that specifies the regularization:

* “activity\_regularizer=regularizers.l1(10e-5)”

Remember L1 regularization constraints the cost function with an absolute value of the magnitude of the weights:



L1 regularization shrinks the less important feature’s coefficient to zero thus, removing some feature altogether.

In the following example we use the “binary\_crossentropy” loss or cost function:

<https://towardsdatascience.com/understanding-binary-cross-entropy-log-loss-a-visual-explanation-a3ac6025181a>

This term effectively tries to maximise the Log-loss function and is very similar to the entropy calculations we did when attempting to discretize a continuous variable. In this approach we are trying to match pixels rather than match the intensity of them which the “mean\_squared\_error” does.

Now when you print out the hidden feature set you will notice that a number of them are zero. This means we have introduced sparsity into the autoencoder.

Have a go at changing the loss function and changing the optimizers in the following Google Colab:

<https://colab.research.google.com/drive/1_7bPYNBIhyuyb-Ef6ljv7jJtlbWpof3d#scrollTo=GC42QRJn0oZ6>

## 3.6 Convolutional autoencoders

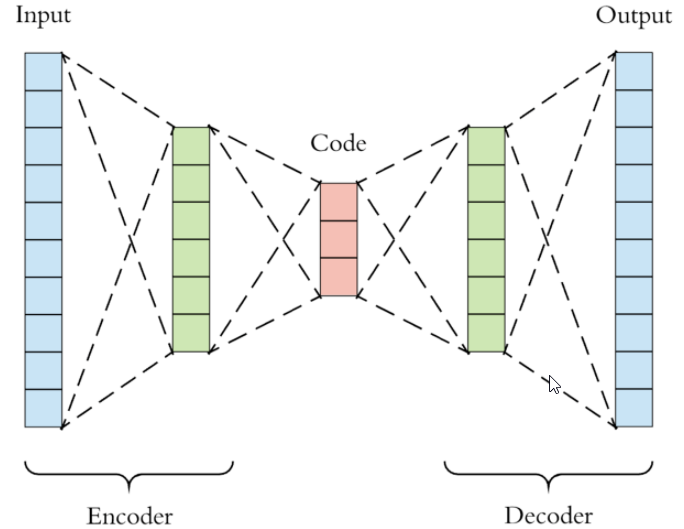
In the next course, Processing Unstructured Data, we will talk about image pre-processing.

Specifically, we will look at how we can use terms known as convolutions, max pooling and upsampling to help us pre-process an image. Some of you may be familiar with these terms so this step will be relatively easy to extend from the regular autoencoder. If you are not familiar with them then go to this link:

<https://pathmind.com/wiki/convolutional-network>

In Step 3.4 we outlined how an autoencoder works. This process is very similar for convolutional autoencoders. Figure 1 shows the structure of a Convolutional Neural Network (CNN) autoencoder. Convolutional Autoencoders can be used to reduce the levels of noise found in images and can be a very useful pre-processing step.

In Step 3.4 we outlined how an autoencoder works. This process is very similar for convolutional autoencoders. Figure 1 shows the structure of a Convolutional Neural Network (CNN) autoencoder. **Convolutional Autoencoders can be used to reduce the levels of noise found in images and can be a very useful pre-processing step**.



As you can see from Figure 1, it is very similar to regular autoencoder, with the exception that it has multiple dimensions. I won’t be going into to much detail here as you will learn all about CNN’s in Machine learning with Prof Tomas Ward. However, I want to run through an example of how a convolutional autoencoder works using a simple example.

Now we will use the Modified National Institute of Standards and Technology (MNIST) dataset from the Keras dataset library, as we did in previous examples, but this time we will add some noise to each image. Make sure you set TensorFlow to version 2.x and open a folder that you want to work in.

You can access the Google Colab for this step here:

<https://colab.research.google.com/drive/159C7nV_Vmdr-OffKF3hx0Ltfyyp8kl8F#scrollTo=cQa9w37LmkG7>

## 3.7 Variational autoencoding

Variational Autoencoders belongs to the general family of autoencoders but are less similar to the approaches we came across in earlier steps of this topic.

Before we move on, it might be worthwhile looking at the following video below:

[VIDEO] <https://youtu.be/9zKuYvjFFS8>

It provides a nice recap of autoencoders and gives a good description of variational autoencoders. Alternatively, if you are really struggling with the whole concept of autoencoders have a look at this article from Towards Data Science. It provides a concise and effective overview of the whole area:

<https://towardsdatascience.com/the-variational-autoencoder-as-a-two-player-game-part-i-4c3737f0987b>

**Recap “normal” autoencoders:**

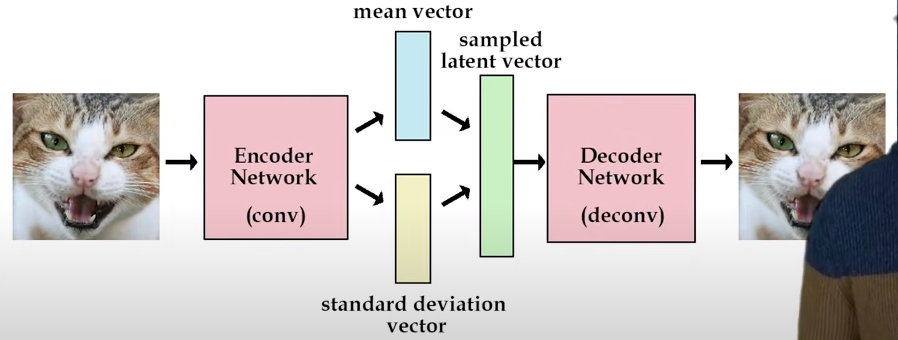
* Using NN or CNN the encoder compresses the dimensions to create a bottleneck.
* From the bottleneck the decoder then reconstructs the input, as before using fully connected convolutional layers.
* At the end we create a reconstruction loss function by comparing the reconstructed input to the original input, e.g. comparing pixel to pixel differences of an image.
* Then we can train our network to e.g. compress images, audio…
* This is useful for sharing images on phones: you take a photo with huge resolution, it’s compressed when you share it, your friend gets the image in low resolution in WhatsApp then opens it up to high resolution, i.e. it has been compressed and reconstructed.
* Can also be used to remove noise or impaint in images (e.g. removing watermarks, logos, photo bombers, etc.)
* A 2D latent space means your compressing to 2 variables, 3D to 3 variables, etc. It’s important to get the size right.

**Variational autoencoders:**

* ‘Normal’ autoencoders  Input mapped to fixed ***vector***
* Variational autoencoders Input mapped to ***distribution*** as close to normal distribution as possible
* Variational bottleck Z is split into two separate vectors:

1. Mean of the distribution
2. Standard deviation of the distribution

* Whenever you need a vector to feed through your decoder network, you just take a sample from the distribution.



In the first autoencoder example in Step 3.4, we mapped the original data back to itself through a fixed function. We did this to try and reduce the dimensionality of our dataset and to reduce noise, as shown in the previous step. The mapping can be shown as follows:

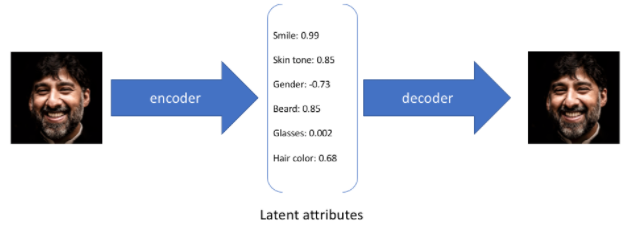


where ***f(X)*** is a reduced feature set of the original ***X***.

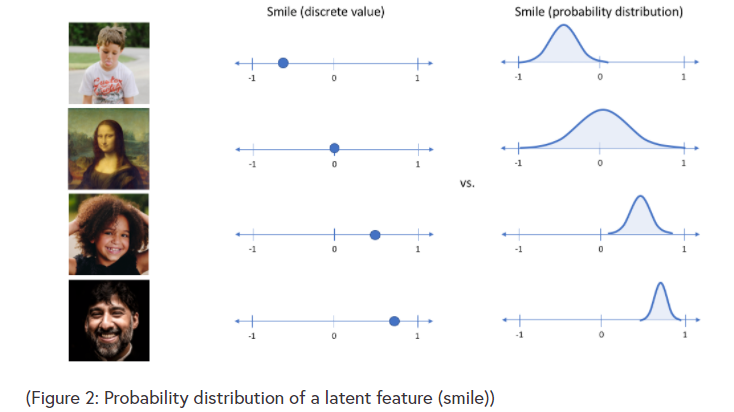
Notice how the latent variables have fixed values (these attributes are fictional) and how “regimented” this process is. If we put a new image into the autoencoder we would get fixed values from the ***f(X)*** function. In fact, it will only really work with the dataset that we have and will not really create realistic features if new data is introduced. In a way what we are looking for is smooth latent state representations of the input data. The following two figures try to highlight the differences between regular autoencoders and variational autoencoders. They are taken from a blog by Jeremy Jordon and hopefully will give you an intuitive interpretation of the difference between them:

<https://www.jeremyjordan.me/variational-autoencoders/>

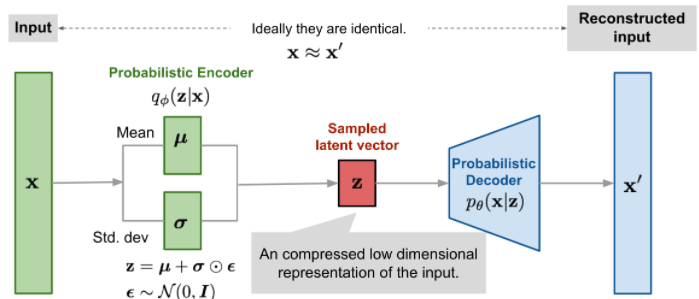
Now if you look at Figure 1 below, you will see that from the image inputted you will get a latent feature with a fixed value.



Now intuitively if you or I were looking at this picture we might score the smile with a value but with some variation. So we might say this is a smile score of . In Figure 2 we can see there are varying sureties of the degree of smile occurring.



So now we can create an encoder and decoder that have as the bottleneck not a fixed function but a probability distribution. Figure 3 shows how the introduction of a probabilistic encoder gives a nice example of how an image goes through a typical variational encoder.



Now, the benefits of this type of encoder are that it will not just create latent vectors but it will also allow you to fill in gaps or missing sections of data. It has a lot of similarities with Generative Adversarial Networks (GANs) although the structures can be said to be reversed.

* A Variational Autoencoder Architecture (**VAE**) reduces its hidden features
* **GANs** expand on them.

According to Towards Data Science, *“a variational autoencoder can be defined as being an autoencoder whose training is regularised to avoid overfitting and ensure that the latent space has good properties that enable generative process*”.

So we now have an algorithm that can potentially generate content and could be considered as a missing value imputation technique.

The following code comes from Github:

<https://github.com/fchollet/keras/blob/master/examples/variational_autoencoder.py>

and was written by the Keras team. It was based on the original work from Kingma, Diederik P., and Max Welling, “Auto-Encoding Variational Bayes.”. I have adapted it to work on Google Colab. You will notice I have commented the pieces out where arguments are expected from the command line.

**Note:** An **isotropic gaussian** is one where the covariance matrix is represented by the simplified matrix:



Where  a diagonal matrix and each diagonal element is equal to  implying that all dimensions are independent.

Please go to the following Google Colab for this step now:

<https://colab.research.google.com/drive/1LdkiXPDAGyfnf90XIPl-9CcUbSFgjzoq#scrollTo=uYPqkg7R_8Wf>

# Quiz

**Question 1 -** In neural networks, a neuron is a function that acts on its input. The result of this input is then transferred to another neuron or output layer.

* True
* False

*The functions are mainly weighted averages of the inputs.*

**Question 2 -** Backpropagation calculates the derivative of the neurons in a neural network.

* True
* False

*Backpropagation calculates the gradient/derivative of the loss/error function.*

**Question 3 –** Autoencoders are the equivalent of independent component analysis (ICA) for nonlinear data.

* True
* False

**Question 4 –** Autoencoders can**not** reduce noise from our input data.

* True
* False

*Autoencoders* ***can*** *be used to reduce noise.*

**Question 5 –** When using sparse autoencoder we must use a lasso regularization constraint.

* True
* False

*It is not the only regularization constraint. You can also use ridge or a derived function to do this.*

**Question 6 –** Sparse autoencoders are used typically used to find independent features in our input data.

* True
* False

*The features they learn are not necessarily independent, this is one of the differences with PCA.*

**Question 7 -** An autoencoder has at least 2 layers.

* True
* False

**Question 8 –** Variational autoencoders can be used for image generation.

* True
* False

**Question 9 –** A variational autoencoder has a bottleneck that produces fixed values.

* True
* False

*VAE’s use a function that is based on a probability distribution. This is then sampled to give a representative sample.*

**Question 10 –** Random Forests are an excellent way to reduce the dimensional size of a dataset.

* True
* False

Topic 4: Cluster analysis

# Topic 4: Cluster analysis

# Clustering, distance and similarity measures

## 4.2 What is clustering?

Clustering is the process of finding groups of objects such that the objects in a group will be similar (or related) to one another and different from (or unrelated to) the objects in other groups.

Diagram

Description automatically generated

Clustering falls into a category of machine learning known as **unsupervised learning**. The idea behind it is to apply an algorithm (for which there are many) which will **use a similarity or distance function**, for example, to separate the data you have into groups. Unlike Principal Component Analysis (PCA) which combines columns/features, **cluster analysis generally groups by row**. So you are really trying to combine groups of people or objects which would have a multitude of variables/features.

PCA  combines columns/features

Cluster analysis  generally groups by row

So, for example, you may have done a survey about the reading habits of people. You could, for example, have asked people if they worked in various sectors, had a certain level of education, where they lived or what their parents read. Now cluster analysis will attempt to group all the people based on their answers to the four questions.

Clustering is really useful when attempting to determine intrinsic grouping among unlabeled data. For example, we may want to subdivide the market into segmented components as follows:

**Market Segmentation:**

* **Goal:** subdivide a market into distinct subsets of customers where any subset may conceivably be selected as a market target to be reached with a distinct marketing mix.

**Approach:**

* Collect different attributes of customers based on their geographical and lifestyle-related information.
* Find clusters of similar customers.
* Measure the clustering quality by observing buying patterns of customers in the same cluster vs. those from different clusters.

Or alternatively, we may want to cluster documents:

**Document Clustering**

* **Goal:** To find groups of documents that are similar to each other based on the important terms appearing in them, Figure 2.
* **Approach:** To identify frequently occurring terms in each document. Form a similarity measure based on the frequencies of different terms. Use it to cluster.
* **Gain:** Information Retrieval can utilize the clusters to relate a new document or search term to clustered documents.

Clustering Points: 3204 Articles of Los Angeles Times Similarity Measure: How many words are common in these documents after some word filtering.

Table

Description automatically generated

As I said previously there are many cluster algorithms, and they can be categorized as follows:

**1. Density-Based Methods**

These methods consider the clusters as the dense region having some similarity and different from the lower dense region of the space. These methods have good accuracy and the ability to merge two clusters. Example Density-Based Spatial Clustering of Applications with Noise (DBSCAN) and Ordering Points to Identify Clustering Structure (OPTICS).

**2. Hierarchical Based Methods**

The clusters formed in this method form a tree-type structure based on the hierarchy. New clusters are formed using the previously formed one. It is divided into two categories:

* Agglomerative (bottom-up approach)
* Divisive (top-down approach).

Examples of this include; Clustering Using Representatives (CURE) and Balanced Iterative Reducing Clustering and using Hierarchies (BIRCH).

**3. Partitioning Methods**

These methods partition the objects into k clusters and each partition forms one cluster. This method is used to optimize an objective criterion similarity function such as when the distance is a major parameter example **K-means**, Clustering Large Applications based upon randomized Search (CLARANS) etc.

**4. Grid-based Methods**

In this method, the data space is formulated into a finite number of cells that form a grid-like structure. All the clustering operation done on these grids are fast and independent of the number of data objects example Statistical Information Grid (STING), wave cluster and CLustering In Quest (CLIQUE).

Now if you would like some detailed notes on cluster analysis please take a look at these slides:

<https://drive.google.com/file/d/1hKeS0rdeLEyGKmwK-cNzUtP4_kME9Fn6/view>

I usually give these out as supplementary material in my lectures. They are very detailed and will take a bit of reading, but they are a good guide to determining which algorithm might suit your needs.

In the next number of steps, I will outline a number of metrics that are used to measure similarity. I will then go through a number of clustering algorithms which will include K-mean, K-medoids and DBSCAN. Finally, we will show you how to assess your clustering techniques with a comparison of the within-cluster variation compared to the between cluster variation.

Read the notes in the slides and ask yourself how would you know if you have “good clusters”. As usual, post any thoughts that you may have in the comments section below.

## 4.3 Similarity metrics

The Euclidean distance is discussed in this step.

In the previous step, we talked about clustering and where it can be used. At the end of the step, I asked the question of how do we know if we have a good cluster? Well, typically a good clustering technique has the following:

* Will produce high-quality clusters with
* **High intra-class similarity.**
* **Low inter-class similarity.**

The quality of a clustering result depends on both the similarity measure used by the method and its implementation and can be measured by its ability to discover some or all of the hidden patterns.

Measuring the quality of the clustering technique is based on how we calculate the distance/similarity. Dissimilarity/Similarity metric is usually expressed in terms of a distance function, typically a metric: ***d(i,j)***.

In addition to the ***d(i,j)*** we need a separate “quality” function that measures the “goodness” of a cluster. This could be the Mean Square Error (MSE) for example.

The definitions of distance functions are usually very different for interval-scaled, boolean, categorical, ordinal ratio, and vector variables. Additionally, the weights of the importance of particular variables will be application-specific. It can be very hard to define “similar enough” or “good enough” as the answer is typically highly subjective.

So generally we want the following requirements from a clustering algorithm:

* Scalability.
* Ability to deal with different types of attributes.
* Ability to handle dynamic data.
* Discovery of clusters with arbitrary shape.
* Minimal requirements for domain knowledge to determine input parameters.
* Able to deal with noise and outliers.
* Insensitive to order of input records.
* High dimensionality.
* Incorporation of user-specified constraints.
* **Interpretability and usability** 🡺 arguable the most important!

### Similarity

This is a considerable list and I generally give the most important to the last request for interpretability and usability. Now if you remember in Pre-processing Data and Feature Impact Calculation we discussed entropy and used the information criteria to create a discrete variable that maintained as much information from the original continuous variable. Now finding the right clustering approach is similar. We are looking for clusters that can give us new information and not just cluster for clustering sake. So the first step in this process is to decide how we calculate similarity or dissimilarity.

**Similarity can be described as follows:**

* Numerical measure of how alike two data objects are.
* Is higher when objects are more alike.
* Often falls in the range [0,1].

**Dissimilarity can be described as follows:**

* Numerical measure of how different two data objects are.
* Lower when objects are more alike.
* Minimum dissimilarity is often 0.
* Upper limit varies.

The first measure dissimilarity we will look at is probably the most well known and is known as the Euclidean distance.

### Euclidean Distance

The Euclidean distance between two points is the length of the path connecting them. Pythagorean theorem gives this distance between two points. Euclidean distance is regularly the default distance for most practitioners.

In Cartesian coordinates:

if **p = (p1, p2,…, pn)** and **q = (q1, q2,…, qn)** are two points in Euclidean n-space,

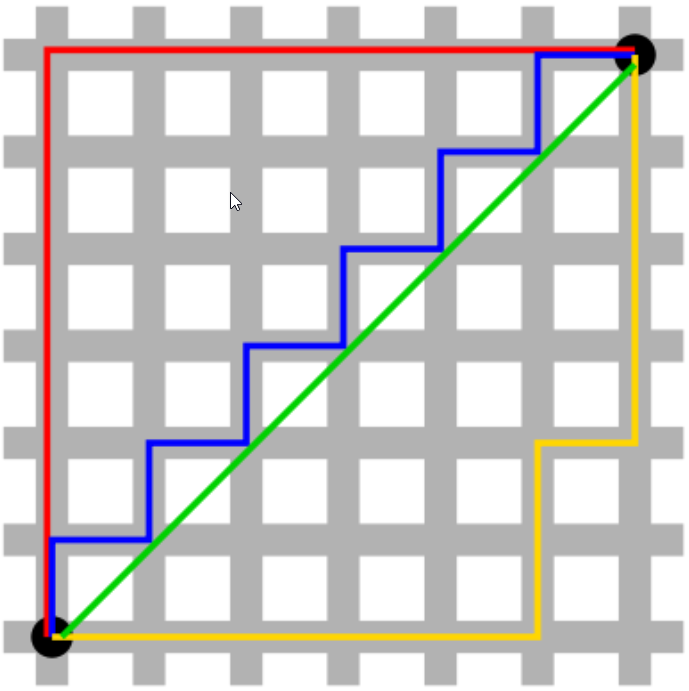
then the distance (d) from p to q, or from q to p is given by the Pythagorean formula:



## 4.4 Manhattan distance

The Manhattan distance gets its name from the New York City taxi drivers problem. The goal is to calculate the shortest distance between 2 points.

With the euclidean distance, the shortest distance will be that calculated by Pythagoras or the hypotenuse of a triangle (the green line in Figure 1). But obviously, if you are a taxi driver you cannot drive through buildings so you have to do an up and across motion.



So we now realise we cannot drive through buildings, which rules out the Euclidean distance as an appropriate formula:



If we want to reflect the taxi drivers path then we should use the sides of each triangle. This is effectively the Manhattan distance:



This Manhattan distance metric is also known as Manhattan length, rectilinear distance, L1 distance or L1 norm, city block distance, Minkowski’s L1 distance, taxi-cab metric, or city block distance. In some cases, it can be preferable to use as a distance measure as it can put less weight on outlying points. It has also been found to be beneficial when you have a very high dimensional dataset.

The code below is a very simple example of how it works using the Scikit learn manhattan\_distances library:

<https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.manhattan_distances.html>

Play about with it and compare it to the Euclidean distance. Post your thoughts on the comments section below.

You can access the Google Colab for this step here:

<https://colab.research.google.com/drive/1l9-VTjJIACpMN16rsEe00eEmblfpIsEO#scrollTo=S_VrPV_GYOVY>

## 4.5 Cosine similarity

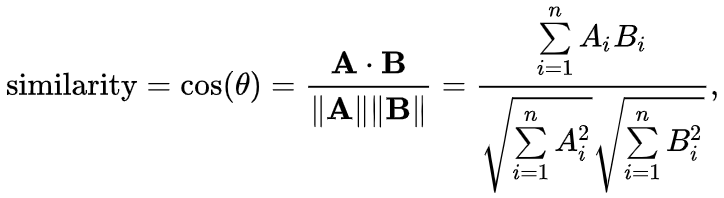
Cosine similarity has been used to assess **how similar documents are to each other**.

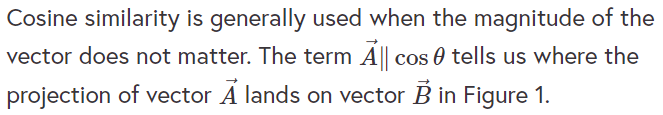
Where Euclidean distance measures the magnitude of the separation between 2 vectors, **cosine similarity gives a measure of the angle between two multidimensional vectors**. It is very similar to correlation where the cosine similarity between centred versions of x and y, again bounded between -1 and 1. Cosine similarity is a metric used to measure how similar the documents are irrespective of their size.

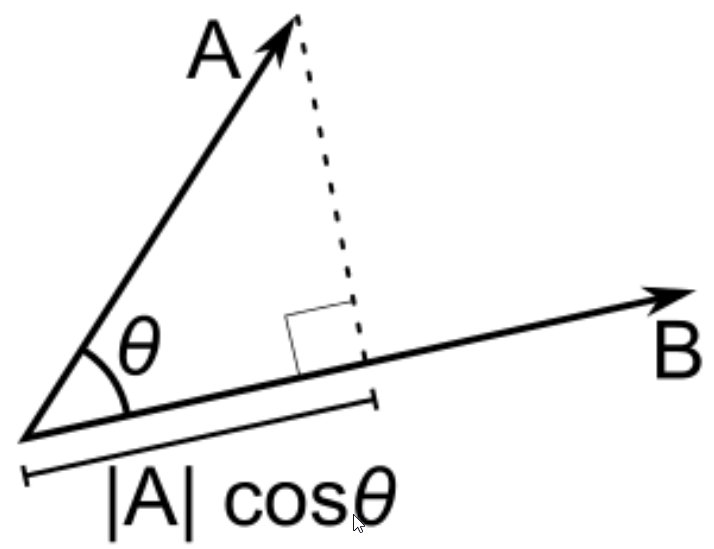
The maths behind this measure are derived from the Euclidean dot product:



this can be converted to similarity by doing some simple algebra:







The following code from Github implements Scikit learns cosine\_similarity. You should notice how two vectors going in the same direction but with differing magnitudes have the same cosine similarity with and :



but when we get the euclidean distance there is a vast difference.

Again, play with these measures. Would you expect z and z2 to be correlated with each other? Place your thoughts in the comments section below.

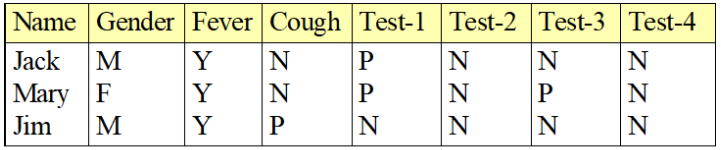
Follow this link to go to the Google Colab for this step:

<https://colab.research.google.com/drive/1VlXzF2fkbRokfFihQrwE7o-otGMi4A4d#scrollTo=yylA0jwbHLoL>

## 4.6 Jacardian Index

So far in all our examples of distance/similarity measure, we have dealt with variables/features that are continuous in nature.

The following is an example where we compare the results of 4 tests between 3 patients. The test results are Positive/Negative and all other information is categorical.



In this situation, it would not make sense to use the Euclidean distance or the cosine similarity measure and so we introduce the Jaccardian index. It is **also known as the Intersection over Union** or **Jaccard similarity coefficient**.

The idea is to:

Compare the rows in question in the dataset by counting the number of times both rows (or subjects in our case above) have a positive occurrence at the same time.

### Similarity of asymmetric binary attributes

Given two objects, A and B, each with n binary attributes, the Jaccard coefficient is a useful measure of the overlap that A and B share with their attributes. Each attribute of A and B can either be 0 or 1. The total number of each combination of attributes for both A and B are specified as follows:

* ***M11*** represents the total number of attributes where A and B both have a value of 1.
* ***M01*** represents the total number of attributes where the attribute of A is 0 and the attribute of B is 1.
* ***M10*** represents the total number of attributes where the attribute of A is 1 and the attribute of B is 0.
* ***M00*** represents the total number of attributes where A and B both have a value of 0.

Each attribute must fall into one of these four categories, meaning that



The Jaccard **similarity** **coefficient**, ***J***, is given as (note we don’t include ***Moo*** as we don’t get score for predicting negatives).



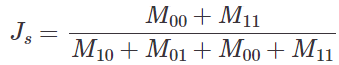
The Jaccard **distance**, ***dJ***, is given as



So from the example above, we would create a contingency table such as that shown in Figure 2 below:

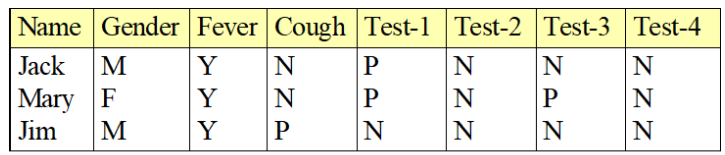


So to calculate the Jaccard index we would use the following formula for a symetric variable:



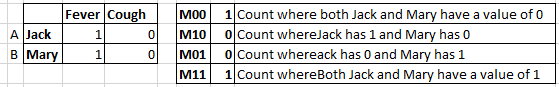
The next question you might ask is; what is the asymmetrical variable? **An asymmetrical variable is a categorical variable where the likely outcome is uneven between the possible outcomes**. So if you were testing patients for a rare disease then we would not expect many people to have it, but when they do we want our metrics to score this as a high occurrence. So in our example above the only **symmetrical variable is gender** (we are assuming an equal probability of the occurrence of male/female). Now the Jaccardian distance for both cases is effective ***1 - J*** for both the symmetrical and asymmetrical cases.

So let’s try and calculate the Jaccardian distance for the example in Figure 1 between Jack and Mary. We will assume the following for this example:

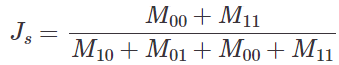


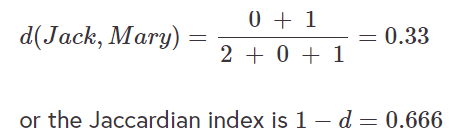
* Gender is a symmetric attribute.
* The remaining attributes are asymmetric binary.
* Let Y and P be set to 1 and the value N be set to 0.

Now the Jaccardian distance between Jack and Mary is as follows:



***This looks wrong!!!***





***Think it should be…***

Jaccard distance = 1 – Jaccard Similarity

Jaccard similarity = 0.33 (as above)

Jaccard distance = 1 – 0.33 = 0.666

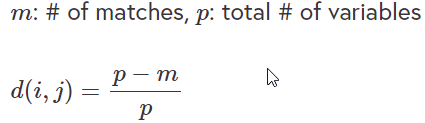
Note that because Gender is symmetrical it was left out of the calculations. If we include Gender it will bias the results, to the point that the distance would go from 0.33 to 0.5. To answer this question one really needs to look at the problem being addressed. In some cases, the Gender variable may be asymmetrical. For example, in a breast cancer study, it is rare for a male to show up, but it is possible.

See this page for more examples:

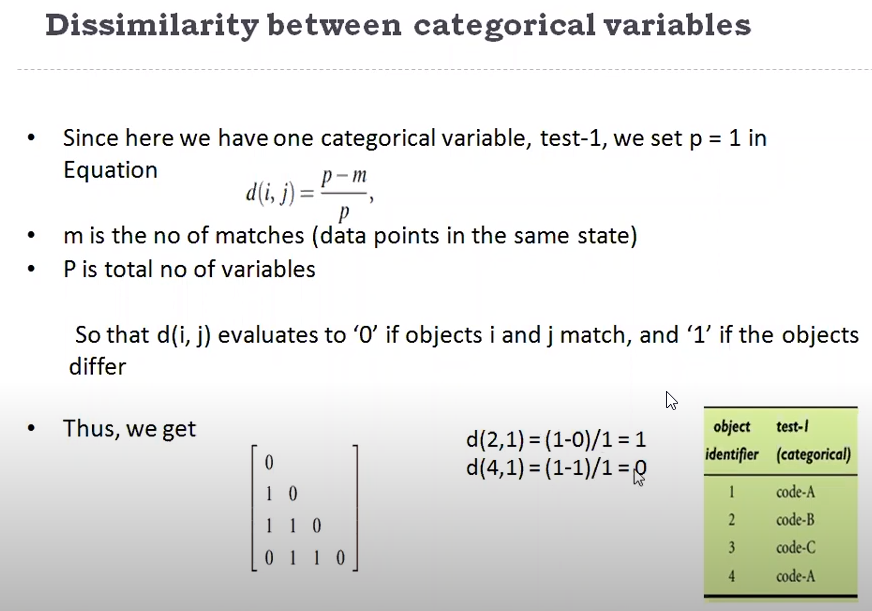
<https://www.statology.org/jaccard-similarity/#:~:text=This%20measure%20gives%20us%20an,0.8%20%3D%200.2%20or%2020%25>.

### Multinomial data

A generalization of the binary variable in that it can take more than 2 states, e.g., red, yellow, blue, green is as follows:



[VIDEO:] <https://www.youtube.com/watch?v=s-1RHbOfT5Y>



### Ordinal Data

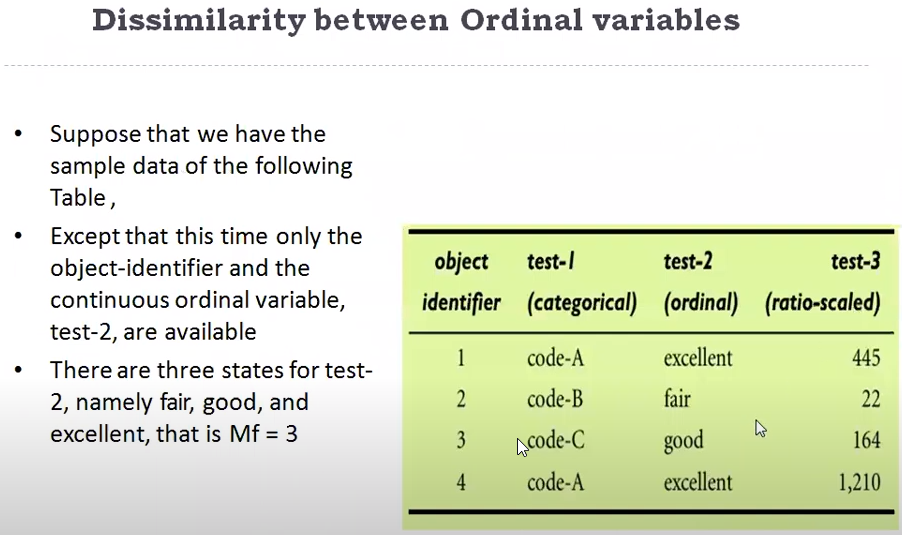
If you have an ordinal variable do the following:



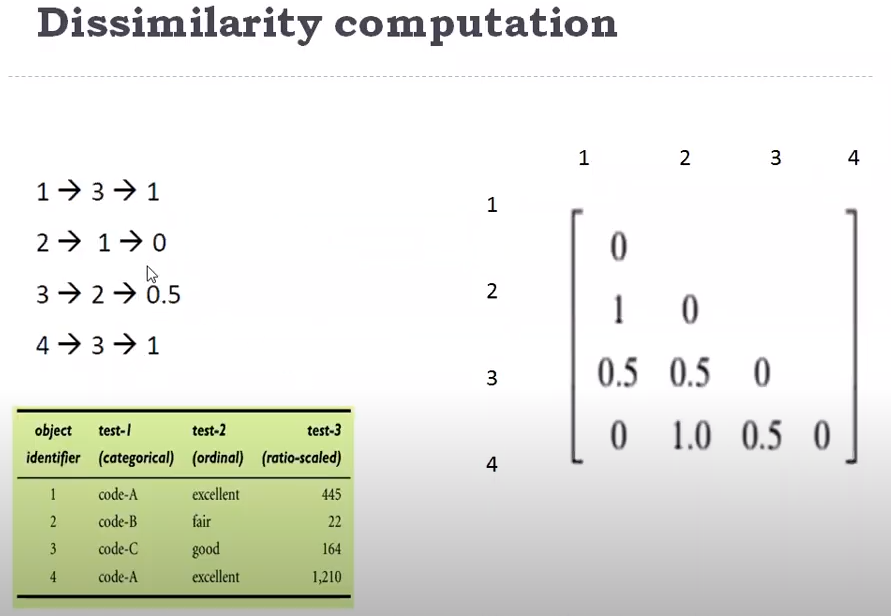
map the range of each variable onto [0, 1] by replacing i-th object in the f-th variable by



Compute the dissimilarity using methods for interval-scaled variables.

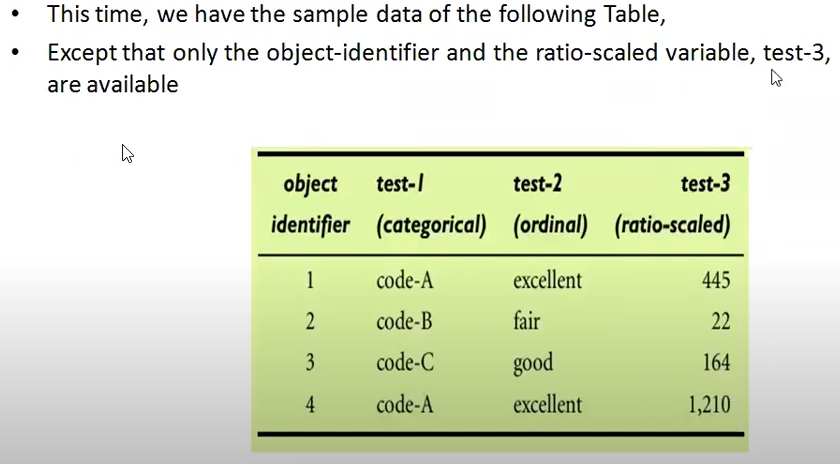




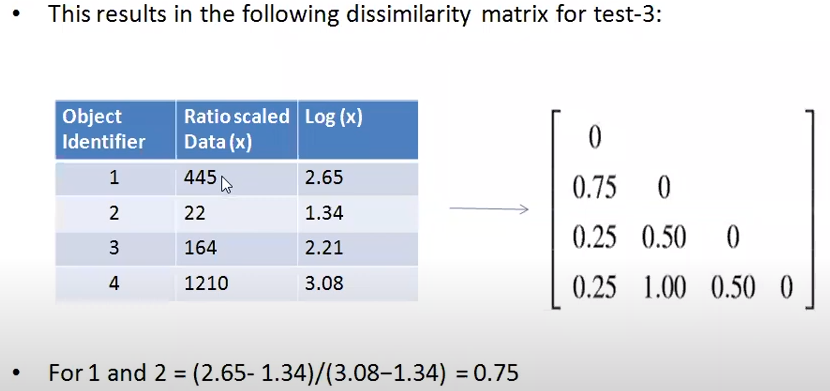


### Ratio-Scaled Data

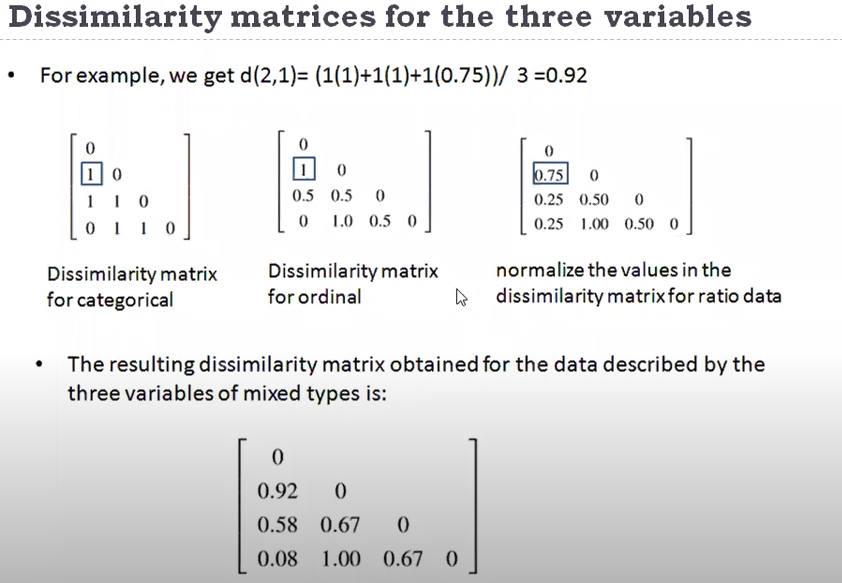
If you have a ratio-scaled variable (e.g. weight) do not treat it as an interval scaled variable as the scale can be distorted. The solution is to apply a logarithmic transformation.





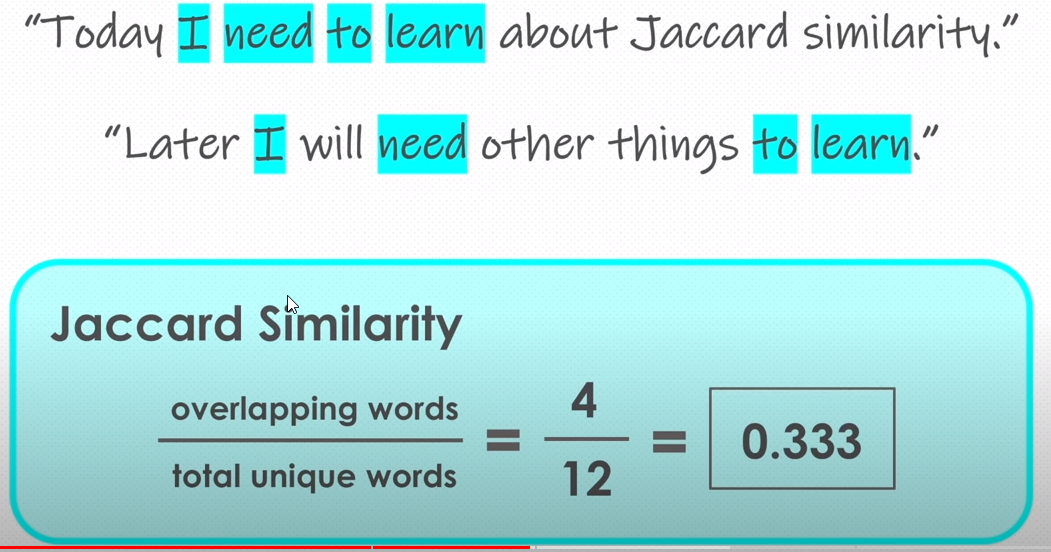


Then you can compare all three dissimilarity matrices for the different variables:

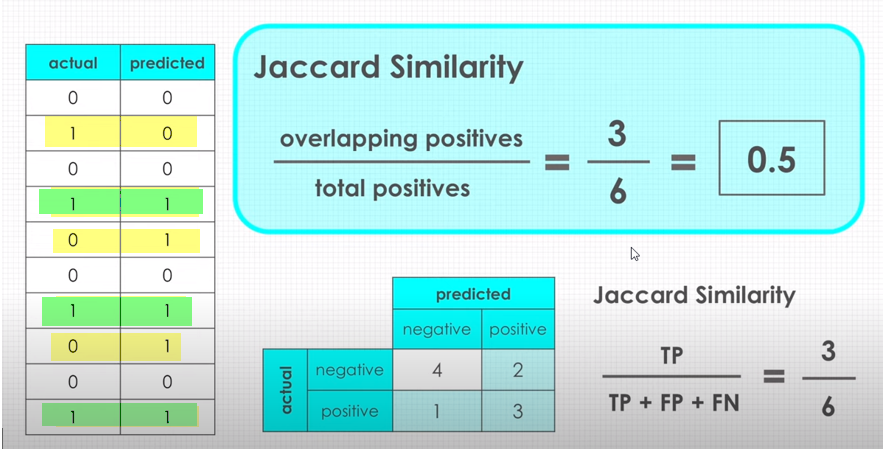


### Uses of Jaccard

1. Similarity of documents (words) example below:



1. Scoring machine learning prediction models, example below:



### Example

The following code shows you how to calculate the Jaccardian distance for the example I gave in Figure 1. It relies on the Python Scipy library and assumes asymmetry. Have a go at trying to calculate the overall distance matrix for the whole dataset. As usual, leave your thoughts on the comments board:

<https://colab.research.google.com/drive/1IoUPaOd6Nhz2yUp0m0Vug9psU1kYkgpe#scrollTo=xEnRwpr8aMJb>

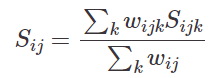
## 4.7 Gower's similarity coefficient

In the previous examples of distance/similarity metric we used either continuous or categorical data as inputs to our distance matrix, but not both.

### What is gower distance?

Like normal Euclidean distance or cosine distance, Gower distance is a distance measure. However, it can be used to calculate distance between two entity whose attribute has a mixed of categorical and numerical values. Why this is important? Many common clustering algorithms, e.g. K-means clustering - only works when all variables are numeric.

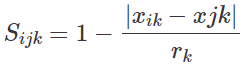
In real-world databases, we encounter both types of data and will regularly find that we have to accommodate both in our cluster analysis. Gower’s similarity coefficient is applied for mixed data types, namely, databases with continuous, ordinal or categorical variables at the same time. Gower’s general similarity coefficient ***Sij*** compares two cases i and j and is defined as follows:



where ***Sijk*** denotes the contribution provided by the k-th variable and ***wijk*** is usually 1 or 0 depending if the comparison is valid for the ***k-th*** variable.

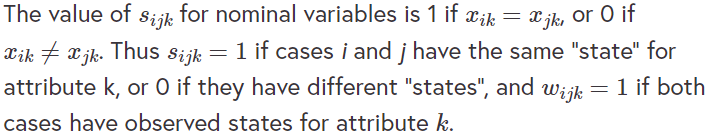
### Ordinal and continuous variables

Gower similarity defines the value of ***Sijk*** for ordinal and continuous variables as follows:



where ***rk*** is the range of values for the ***k-th*** variable.

### Nominal Variables



The following code snippet imports a gower object and lets you calculate the distances using the Google Colab file for this step. You can also specify the weights to determine what variables should be included in your distance measures. Again play with the parameters and see how you get on:

<https://colab.research.google.com/drive/1D6BQDrl6rYQSOovQUY-Ney1wEQckpXuR#scrollTo=9ybanzBmA2dZ>

## 4.8 Clustering approaches

In the previous steps, we examined a number of metrics that could be used to measure distance or similarity.

The next step of the “Clustering” journey is to outline an approach/algorithm that could be used to identify relevant clusters. The following is a list of potential cluster algorithm categories and some of the algorithms that belong to them.

**Partitioning Approach:**

In these approaches, we construct various partitions and then evaluate them by criterion such as minimizing the sum of square errors.

* Typical methods: k-means, k-medoids, CLARANS

**Hierarchical approach:**

We create a hierarchical decomposition of the set of data (or objects) using some criterion.

* Typical methods: Diana, Agnes, BIRCH, ROCK, CAMELEON

**Density-based approach:**

These approaches are based on connectivity and density functions.

* Typical methods: DBSACN, OPTICS, DenClue

**Grid-based approach:**

Grid-based approaches are based on a multiple-level granularity structure.

* Typical methods: STING, WaveCluster, CLIQUE

**Model-based:**

A model-based approach used a hypothesized model for each of the clusters and tries to find the best fit of that model to each other.

* Typical methods: EM, SOM, COBWEB

**Frequent pattern-based:**

Frequent pattern-based are based on the analysis of frequent patterns.

* Typical methods: pCluster

**User-guided or constraint-based:**

Clustering by considering user-specified or application-specific constraints.

* Typical methods: Clustering with Obstructed Distance (COD obstacles), constrained clustering

The slides I mentioned in Step 4.2 give a lot more detail to each of the clustering approaches outlined above. Have a look at the references at the bottom of them as they could be really useful to you if you want to get into more detail. The book that these slides are based on is called Data Mining Concepts and Techniques by Morgan et al. (2016). This is an excellent reference guide to the area.

We are now going to implement a number of popular methods and will discuss the pros and cons of each method.

## 4.9 K-means (Lloyd's algorithm)

K-means is one of the most popular cluster techniques applied in data mining.

K-means attempts to divide the ***n*** rows of data into ***k*** clusters, and is implemented in four steps:

1. Partition objects into k non-empty subsets.
2. Compute seed points as the centroids of the clusters of the current partition (the centroid is the centre, i.e., mean point, of the cluster).
3. Assign each object to the cluster with the nearest seed point.
4. Go back to Step 2, stop when there are no more new assignments.

**K-means can be relatively efficient** (O(tkn), where t is the number of iterations and k, t « n), but disadvantages are:

* susceptible to outliers or noisy data
* struggles with local optimums when you have large datasets
* only applicable to continuous data

The global optimum may be found using techniques such as deterministic annealing and genetic algorithms.

It also requires us to specify ***k*** which is not always possible. Finally, the shape of the clusters is very important for -means. In reality, it will not do a good job finding clusters of non-convex shapes, and generally, it requires that the clusters be of similar size and density.

This code snippet is modelled on an example from Towards data science. However, I have changed the dataset to the Scikit learns Iris dataset.

<https://drive.google.com/open?id=1pCp_c5m5i76-Iiik0t80IXQU9BVraUsi>

<https://towardsdatascience.com/machine-learning-algorithms-part-9-k-means-example-in-python-f2ad05ed5203>

## 4.10 K-Medoids clustering

In the previous step, **we outlined how the k-means algorithm** is the most commonly used clustering algorithm. We also outlined how it **has a number of drawbacks related to outliers and the size of the dataset** for example.

K-medoids is a clustering algorithm that uses partitioning to create clusters. **Unlike the k-means algorithm which attempts to minimize the total squared error, the k-medoids choose actual datapoints as centres**. K-medoids is also a partitioning technique of clustering that clusters the data set of n objects into k clusters with k known a priori. It’s more robust to outliers than the k-means algorithm and uses the medoid as opposed to the average of a cluster to determine centrality.

The most common realisation of k-medoid clustering is the Partitioning Around Medoids (PAM) algorithm and is as follows:

1. **Initialize:** Randomly select k of the n data points as the medoids.
2. **Assignment step:** Associate each data point to the closest medoid.
3. **Update step:** For each medoid m and each data point o associated to m swap m and o and compute the total cost of the configuration (that is, the average dissimilarity of o to all the data points associated to m). Select the medoid o with the lowest cost of the configuration.
4. Repeat alternating steps 2 and 3 until there is no change in the assignments.

**PAM works well for small datasets and deals with outliers well** but has many of the other problems that K-means has. There are other alternatives such as CLARA (Clustering Large Applications) algorithm which select random observations from the dataset and performs partitioning around medoids (PAM) algorithm on them.

The following Google Colab file implements the k-medoids algorithm. As usual, we encourage you to play with the code and try it out on a dataset of your choosing.

<https://drive.google.com/open?id=1GCGejokDZoYz5bW87LjMpfNkaYV-lAeo>

## 4.11 DBSCAN

**DBSCAN** differs from the k-means and k-medoids algorithm in so far as it is a **density-based algorithm** as opposed to a partition approach.

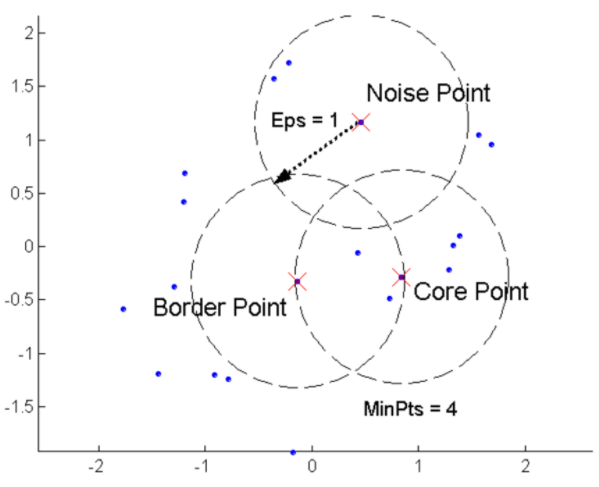
The DBSCAN algorithm views clusters as areas of high density separated by areas of low density and tries to avoid including “outliers” in clusters. A really nice explanation of the algorithm can be found on the Scikit learn site:

<https://scikit-learn.org/stable/modules/clustering.html#dbscan>

The following definitions are useful and will help you understand the algorithm.

* Density is defined as the number of points within a specified radius.
* Eps specifies how close points should be to each other to be considered a part of a cluster. It means that if the distance between two points is lower or equal to this value (eps), these points are considered neighbours.
* MinPoints are the minimum number of points to form a dense region. For example, if we set the minPoints parameter as 5, then we need at least 5 points to form a dense region.
* A point is a core point if it has more than a specified number of points (MinPts) within Eps. These are points that are at the interior of a cluster.
* A border point has fewer than MinPts within Eps, but in the neighbourhood of a core point.
* A noise point is any point that is not a core point or a border point.

The following figure illustrates how DBSCAN labels data points on each iteration of the algorithm.



* Label all points as core, border, or noise points.
* Eliminate noise points.
* Put an edge between all core points that are within of each other.
* Make each group of connected core points into a separate cluster.
* Assign each border point to one of the clusters of its associated core points.

Similarly to all other types of clustering algorithms, DBSCAN possesses its own advantages and disadvantages.

Advantages:

* Can discover arbitrarily shaped clusters
* Can find a cluster completely surrounded by different clusters
* Robust toward outlier detection
* Insensitive to the ordering of the points in the database

Disadvantages:

* DBSCAN Does not work well when dealing with clusters of varying densities.
* While DBSCAN is great at separating high-density clusters from low-density clusters, DBSCAN struggles with clusters of similar density.
* Struggles with high dimensionality data.

There are other density-based approaches such as OPTICS, Ankerst et al (SIGMOD’99) and DENCLUE, Hinneburg & Keim (KDD’98).

We will now implement an example which comes from Scikit learn in the following Google Colab file available here.

<https://drisve.google.com/open?id=1mUw7VOfZ0pl-26mZmrLch7uoadDu3tbw>

**Quiz –** Question 1

Cluster analysis uses supervised learning to find groups of objects within a dataset.

* True
* False

**Question 2 –** Density-based approaches are better than partition-based approaches.

* True
* False

**Question 3** – The cluster structure will depend on the algorithm you use.

* True
* False

**Question 4 -** K-means is not good at clustering no-convex shapes.

* True
* False

**Question 5 –** A good cluster method will produce a high-quality cluster with a high intra-class similarity and a low inter-class similarity.

* True
* False

**Question 6 –** The Manhatten distance is not a member of the family of Minkowski distance.

* True
* False

**Question 7 -** A symmetric binary variable is one where one category is more important than the other.

* True
* False

**Question 8 –** The Gower coefficient allows us to combine variables of differing types.

* True
* False

**Question 9 –** DBSCAN, Optics and DenCLue are from the Hierarchical family of clustering approaches.

* True
* False