← Go Back to Making Sense of Unstructured Data

Course Content

Basics of Linear Algebra

Matrix and Vectors

To understand machine learning in-depth, let us first understand a few topics related to linear algebra. You do not necessarily need to understand linear algebra in-depth, to get started with machine learning. However, it will be useful in understanding the mathematics behind machine learning algorithms.

In linear algebra, data is mostly represented using matrices and vectors.

Matrix

Matrices (plural of matrix) are used throughout machine learning algorithms, specifically for input variables, i.e., the variables we try to understand in machine learning. A matrix of size $m \times n$ is a two-dimensional array that has **m** rows and **n** columns.

Vector

A vector is a one-dimensional array that has only one row - called a row vector, or just one column, also called a column vector.

For example, in the below figure, X is a matrix with 3 rows and 2 columns, while Y is a simple column vector.

$$X = \begin{pmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \end{pmatrix} \qquad Y = \begin{pmatrix} 1 \\ 3 \\ 5 \end{pmatrix}$$

Now, let's look at some operations that can be applied to vectors and matrices.

Addition and Subtraction

For the addition and subtraction of two vectors or two matrices, both should be of the same size. The addition and subtraction operations are performed in an "element-wise" manner between the two objects - that is, the corresponding elements from the two vectors or matrices are added to each other or subtracted from each other.

For example, the below figure shows the addition and subtraction of two column vectors, each having 3 elements.

$$X = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \quad A = \begin{bmatrix} 1 \\ 4 \\ 7 \end{bmatrix}$$

$$X+A = \begin{bmatrix} 3 \\ 5 \\ 10 \end{bmatrix} \quad X-A = \begin{bmatrix} 1 \\ -3 \\ -4 \end{bmatrix}$$

This kind of "element-wise" addition and subtraction can similarly be performed for two or more matrices of the same size as well.

Multiplying a matrix with a vector

To multiply a matrix with a column (or row) vector, **the matrix must have the same number of columns (or rows) as the number of elements in the column (or row) vector.**

Let us look at an example of multiplying a matrix with a vector, where **X** is an $n \times 1$ column vector and **A** is an $m \times n$ matrix.

$$X = \begin{bmatrix} 2 \\ 1 \\ 3 \end{bmatrix} \quad A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$

As the number of columns in **A** is equal to the number of elements in **X**, it is possible to find **AX** (**A** multiplied by **X**).

To do multiplication, we multiply each element of row 1 of the matrix with the corresponding element of the column vector and add them, i.e., (1*2 + 2*1 + 3*3) = 13. Similarly, we can apply the same operation to rows 2 and 3 of matrix **A** with the same three elements of **X**, and get the below result:

$$AX = \begin{bmatrix} 13 \\ 31 \\ 49 \end{bmatrix}$$

Multiplying a matrix with another matrix

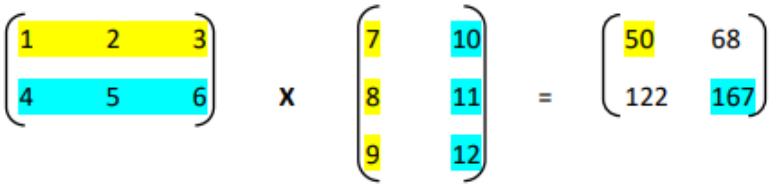
One way to multiply a matrix with another matrix is what's known as the **Dot Product.**

To be able to multiply two matrices and get a dot product, **the number of columns in the first matrix should equal the number of rows in the second matrix.**

The resultant matrix would have the same number of rows as the first matrix and the same number of columns as the second matrix.

For example, the below figure shows the dot product of two matrices.

1. The first element, i.e., the element at the first row and the first column of the resultant matrix, would be*: (1*7) + (2*8) + (3*9) = 50*
2. The second element, i.e., the element at the first row and the second column of the resultant matrix, would be: *(1*10) + (2*11) + (3*12) = 68*
3. The third element, i.e., the element at the second row and the first column of the resultant matrix, would be: *(4*7) + (5*8) + (6*9) = 122*
4. The last element, i.e., the element at the second row and the second column of the resultant matrix, would be: *(4*10) + (5*11) + (6*12) = 167*

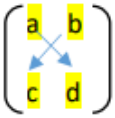


Determinant of a matrix

The determinant of a matrix is a **special number** that can only be calculated from a square matrix, i.e., a matrix with the same number of rows and columns.

For example: A (**2 x 2)** or (**3 x 3)** matrix would be considered a square matrix.

For a (2 × 2) square matrix:



The determinant is calculated using the below formula:

det(A)= a*d - b*c
"The determinant of A equals a times d minus b times c"

Note: In Python, vectors and matrices are represented as Numpy arrays. Numpy is one of the most popular linear algebra packages in Python and can be used to implement several basic mathematical operations on matrices, such as calculating dot products and determinants.

‹ Previous

Next ›

← Go Back to Making Sense of Unstructured Data

Course Content

Covariance

Variance: Variance helps us understand how far our random variable is spread out from the mean, for example, the income of the people may have a high variance as some people may have high income levels.

The formula for variance for a sample is given by:

$$\sigma_x^2 = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

where n is the number of samples (e.g. the number of people) and $\bar{x}$ is the mean of the random variable x (mean of the income).

Covariance: It measures how much two random variables vary together. e.g. The income of a person and the expenses of that person in a population. More precisely, covariance refers to the measure of how two random variables in a data set will change together. A positive covariance means that the two variables at hand are positively related, and they move in the same direction. A negative covariance means that the variables are inversely related, or that they move in opposite directions.

The formula for covariance is given by:

$$\sigma(x,y) = \frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})$$

where n is the number of samples (e.g. the number of people) and $\bar{x}$ is the mean of the random variable x (represented as a vector).

The variance $\sigma^2 x$ of a random variable x can be also expressed as the covariance with itself by σ(x,x).

Covariance Matrix: Following from the previous equations, the covariance matrix for the two dimensions is given by:

$$C = \begin{pmatrix} \sigma(x,x) & \sigma(x,y) \\ \sigma(y,x) & \sigma(y,y) \end{pmatrix}$$

In this matrix, the variances appear along the diagonal and covariances appear in the off-diagonal elements.

Note: You can use the function **numpy.cov** to get the covariance matrix in Python.

← Go Back to Making Sense of Unstructured Data

Course Content

Dimensionality Reduction (PCA & tSNE)

In real-world situations, we often deal with **high-dimensional** data, with lots of columns or "features" that represent the information collected about each observation. High-dimensional data is disadvantageous for a couple of reasons:

- It is generally difficult to analyze or visualize high-dimensional data and identify hidden patterns.
- Not all the features or dimensions of the data are equally important.

Therefore, we need to reduce the dimensionality of the dataset in such a way that by losing only a minimal amount of information, we can visualize the data and identify patterns more easily with a smaller number of features.

Two important techniques that we can use for dimensionality reduction are:

- PCA
- t-SNE

PCA

The main idea of principal component analysis (PCA) is to reduce the dimensionality of a dataset consisting of many variables correlated with each other, either heavily or lightly, while retaining the variation present in the dataset.

This is done by transforming the variables to a new coordinate space of variables, which are known as principal components (or simply, the PCs), and are orthogonal to each other.

The selection of principal components is such that the retention of variation present in the original variables is the maximum for the first principal component and decreases as we move down in order. The principal components are the eigenvectors of the covariance matrix, and hence they are orthogonal.

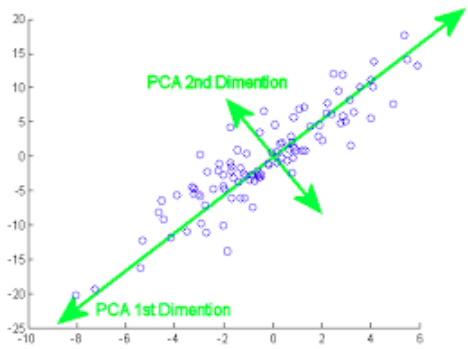


Image Source (Links to an external site.)Links to an external site.

t-SNE

The t-SNE algorithm calculates a similarity measure between pairs of instances in the high dimensional space and in the low dimensional space. This is done by setting the probabilities from the low-dimensional space similar to those of the high-dimensional space. We measure the difference between the probability distributions of the two-dimensional spaces using Kullback-Leibler divergence and try to optimize it.

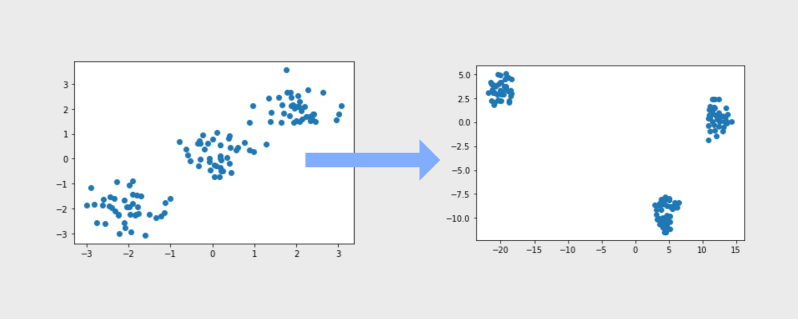


Image Source (Links to an external site.)Links to an external site.

Previous

Next

← Go Back to Making Sense of Unstructured Data

Course Content

Distance and Scaling Measures

Unsupervised learning algorithms use different distance measures or similarity/dissimilarity measures between each pair of observations to group data into different clusters. Points which are close to each other on that distance metric are likely to be grouped into the same cluster, while points far apart on that distance metric are likely to be members of different clusters. All this is done by computing a distance matrix that has distances between every pair of observations. There are different ways of calculating these distances, some of which are mentioned below:

Euclidean distance: Euclidean distance is calculated as the square root of the sum of the squared differences between two vectors.

Say there are two points P (x_1, y_1) and Q (x_2, y_2). The Euclidean distance between these two points would be calculated as:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Manhattan distance: Also called city block distance, it calculates the distance between two points by drawing an orthogonal, zig-zag grid between them.

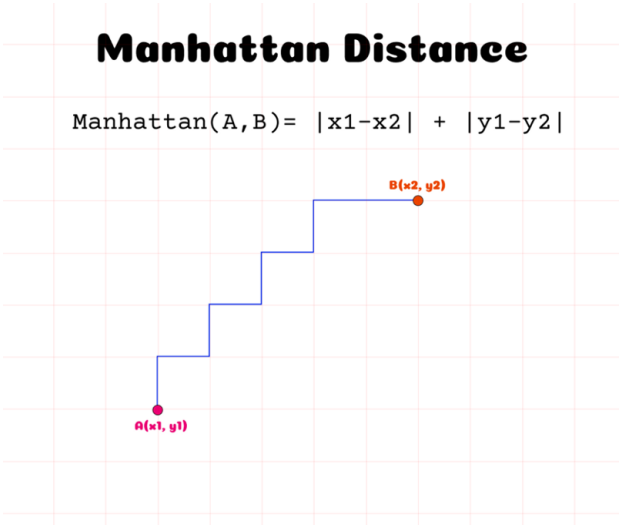


Image Source

Scaling:

Input variables in a dataset may have different units e.g. kilometers, hours, kilograms, etc. i.e. different scales. This difference in scales increases the difficulty in modeling, in turn resulting in an unstable model. Unless you

This difference in scales increases the difficulty in modeling, in turn resulting in an unstable model. Unless you scale the data, the importance of 1 km would be the same as 1 kg, which would be the same as 1 hr, the same as 1 cm, etc.

In other words, while 1000 gms and 1 kg mean the same thing, a quantity of 1000 ml from another feature is at a different scale as a number relative to 1000 g, than it is with respect to 1 kg. Unless both the weight (i.e. gms) and the volume (i.e. ml) are on the same scale, machine learning algorithms might give one of them more weightage than the other simply due to the number in that quantity. That in turn, diminishes the effect of the variable that has the smaller number (lower scale).

Thus, scaling the variables is an important step in machine learning models. By scaling the variables, we can ensure that the machine learning algorithm gives every variable a similar weightage in terms of its likelihood of contributing to the decision making of the algorithm, and that no single variable with a high numerical quantity as its value unnecessarily influences the algorithm's predictive power.

Normalization: One of the ways of scaling the data is so that all the values lie between 0 and 1. This is called Normalization. A value can be normalized as follows:

$$y = (x - min) / (max - min)$$

where,

- **y**: normalized version of the variable
- **x**: variable of interest
- **min**: minimum value of the variable x in this dataset
- **max**: maximum value of the variable x in this dataset

Standardization is another way of scaling the data, where the mean of the observations becomes 0 and the standard deviation is 1. A value can be standardized as follows:

$$y = (x - mean) / std\_dev$$

where,

- **y**: standardized version of the variable
- **x**: variable of interest
- **mean**: average (arithmetic mean) of the variable x in the dataset
- **std_dev**: standard deviation of the variable x in the dataset

‹ Previous

Next ›

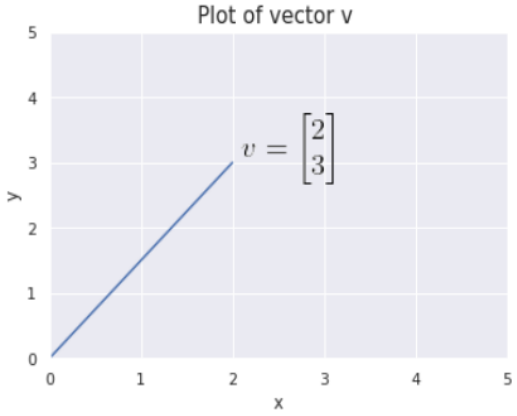
← Go Back to Making Sense of Unstructured Data

Course Content

Eigenvectors and Eigenvalues

Consider that we have a vector **V** and a matrix **A**.

If we multiply matrix **A** with vector **V**, we obtain a new, transformed vector.

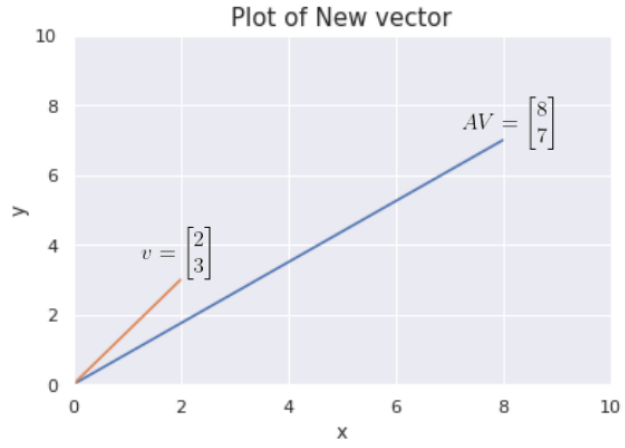


$$A = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix}$$

$$AV = \begin{bmatrix} 1 & 2 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

$$= \begin{bmatrix} 8 \\ 7 \end{bmatrix}$$

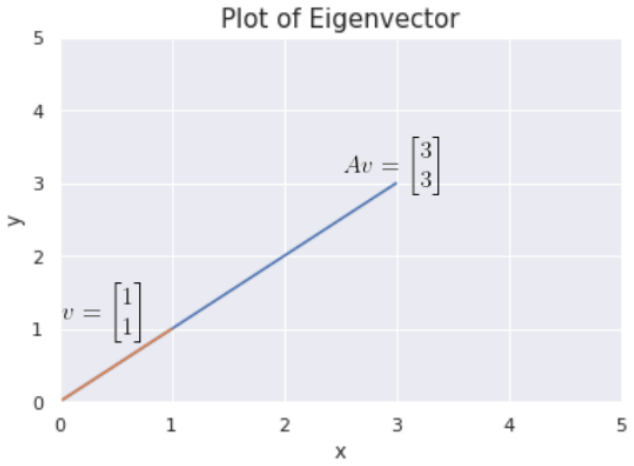
The below figure shows that multiplying by the matrix **A** has scaled the vector **V** to a new vector, with a different magnitude and a slightly different direction.



As we know, vectors have both a magnitude and a direction. The new vector **AV** seems to have a different direction as well as magnitude in comparison to the old vector **V**.

In linear algebra, the operation above is known as a **linear transformation.** It is not only restricted to scaling - linear transformations can be used for flipping, rotating, shearing and other mathematical operations.

Let's now consider a different vector **V** and multiply it with the same matrix **A**:



$$A = \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix} \quad v = \begin{bmatrix}1\\1\end{bmatrix}$$

$$Av = \begin{bmatrix}1 & 2\\2 & 1\end{bmatrix}\begin{bmatrix}1\\1\end{bmatrix}$$

$$Av = \begin{bmatrix}3\\3\end{bmatrix} = 3\begin{bmatrix}1\\1\end{bmatrix}$$

In this example, we notice that something different has happened.

We see that in this instance, while the magnitude of the new vector is certainly different, **its direction has not changed.**

These special vectors are known in linear algebra as **Eigenvectors**.

As illustrated in the example above, given the corresponding matrix A, eigenvectors are directionally-invariant when multiplied with that matrix i.e. they don't change their direction on multiplication with matrix A - they merely get scaled in terms of magnitude. The value with which the eigenvector gets scaled, is known as the **Eigenvalue,** denoted by the symbol lambda.

In our example, the eigenvector is $\begin{pmatrix}1\\1\end{pmatrix}$ and the eigenvalue is 3.

So to summarize, the eigenvector of a matrix is a vector whose direction does not change when a linear transformation (matrix multiplication) is performed on it.

Mathematically, the equation is represented as:

$$Av = \lambda v$$

- **A** - Transformation matrix
- **v** - Eigenvector
- λ - Eigenvalue

Taking λv to the left side:

$$Av - \lambda v = 0$$

Lambda is a scalar, so taking **v** out in common:

$$(A - \lambda I)v = 0$$

An eigenvector is a non-zero vector. So, **v** cannot be zero.

Hence, to satisfy the right-hand side condition, $(A - \lambda I)$ needs to be zero.

If we were to multiply by the inverse of the matrix on both sides:

$$(A - \lambda I)^{-1}(A - \lambda I)v = (A - \lambda I)^{-1} \times 0$$

Since the product of a matrix and its inverse is I (the identity matrix), equating the left and right sides, we get:

$$v = 0$$

This is contradictory, as we have mentioned above that the Eigenvector cannot be zero.

So that means we cannot actually use the inverse of $(A - \lambda I)$, as it is not an invertible matrix.

In linear algebra, **if a matrix is not invertible then the determinant of the matrix is equal to zero.**

That means, we only need to solve the equation $det(A - \lambda I) = 0$, to get the eigenvalues and through them, the eigenvectors.

Due to Numpy in Python, we do not need to perform these operations by hand. Numpy has functions to find the eigenvalues and eigenvectors of a matrix for us.

Why are Eigenvectors important?

The directional invariance of Eigenvectors turns out to be an incredibly important mathematical property, and is utilized by many applications - Principal Component Analysis (PCA), for example, is a highly popular data projection technique that can be used to reduce the dimensionality of a dataset and visualize it in lower dimensions.

To understand the concept of Eigenvectors and Eigenvalues in some more detail, check out this video from 3 Blue 1 Brown:

Eigenvectors and Eigenvalues

This video talks about Eigenvectors and Eigenvalues with the help of visual representations and an example.

It includes the calculations that are needed to get these values, and how to interpret them.

‹ Previous

Next ›

← Go Back to Making Sense of Unstructured Data

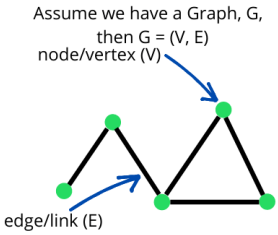
Course Content

Graph Theory

Graph Theory is the study of graphical structures that model relations between two variables or objects. Structurally, graphs are merely a collection of nodes inter-connected by edges in various ways.

Graphs are used by various algorithms in machine learning to perform tasks like clustering, classification, and regression.

A graph is usually represented using nodes (or vertices) and edges (or links).



Mathematically, graphs are sometimes represented as:

G = (V,E)

where **V** is the number of vertices and **E** is the number of edges in the graph.

Graphs can either be directed or undirected.

In an undirected graph, the path between 2 nodes is merely a connection between them, and has no inherent source/target, while in a directed graph, every edge is a clear path from one node to another.

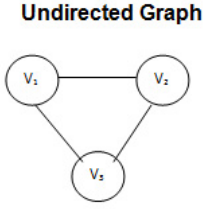


Figure 1: An Undirected Graph

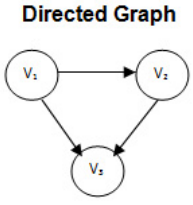


Figure 2: A Directed Graph

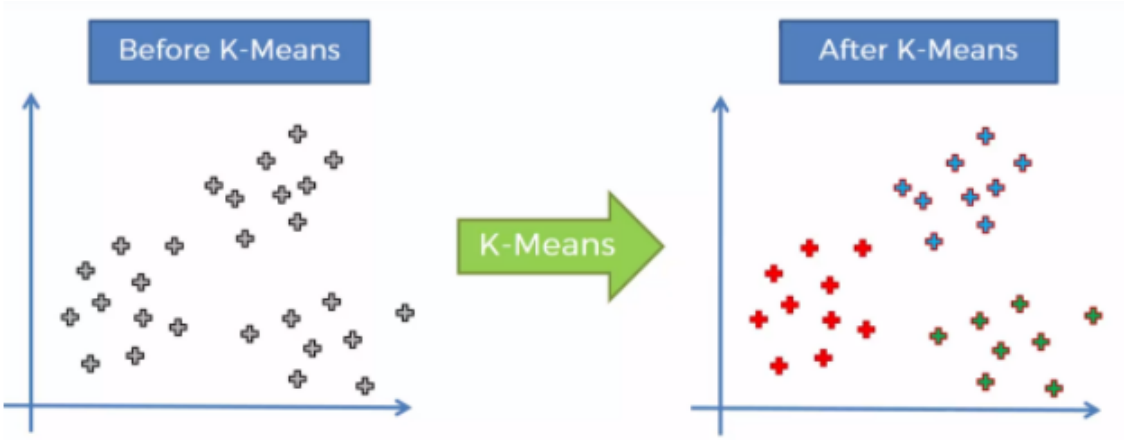
← Go Back to Making Sense of Unstructured Data

Course Content

K-means Clustering

K-means Clustering is an unsupervised learning algorithm. Like other clustering algorithms, it tries to aggregate similar objects into groups called clusters. In K-means Clustering, **K** refers to the number of clusters required. The concept of a centroid, which is the geometric center of a cluster, is used to determine the clusters that K-means finds in the dataset.

Let's understand this using an example. Suppose you go to a vegetable shop to buy some vegetables. There, you'll see different kinds of vegetables. One thing you may notice is that the vegetables will be arranged in a group of their type. The carrots and radishes will probably be kept together in one place, onion and garlic will probably be arranged in another place, potatoes will be kept together, and so on. This arrangement resembles a group or a cluster, where each vegetable is kept within its kind of group, forming the clusters.



The image on the left is *before clustering*, where all the categories or groups appear to be mixed up (same color), while the image on the right is *after clustering* where groups of similar data points seem to be clustered together and depicted with different colors.

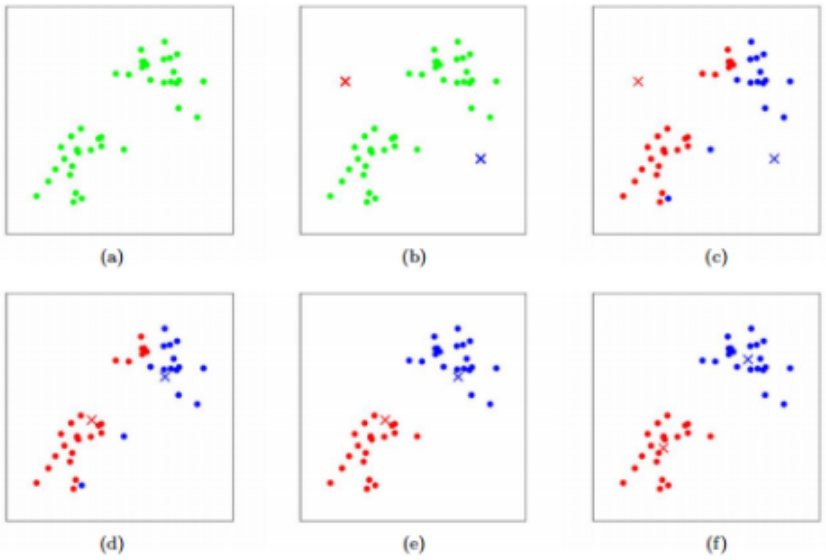
To human eyes, it can be difficult to analyze or understand a mixed-up group like the one represented by the image on the left. So we apply clustering techniques like K-means Clustering to convert this into a more visually distinct dataset with separate groups or categories of data.

Now that we have understood the basic rationale behind clustering, let's look into the working of K-Means Clustering specifically.

Working of K-Means Clustering

The steps involved in K-means Clustering are:

1. Choose the number of clusters K
2. Initialize the centroids
3. Assign each data point to the closest centroid.
4. Update the centroid by taking the mean of the cluster.
    - Repeat steps 3 and 4 until convergence i.e No discernible change in centroids is observed.



For step 3, to assign each data point to the nearest centroid, we use the distance between the centroid and the data point. This distance can be found using Euclidean distance.

The Euclidean distance d between two points (x1,y1) and (x2,y2), is defined as,

$$d = \sqrt{|x2 - x1|^2 + |y2 - y1|^2}$$

Therefore, K-means clustering uses the Euclidean distance to allocate each of the data points to its nearest cluster, so that the sum of squares within each cluster is minimum.

For step 4 we update the centroid using the mean value of all the points in the cluster, hence the name *K-means clustering*.

Example:

Let's understand this with an example:

N={2,3,4,5,10,11,13,15} - Performing K-means on these points, all points on the x-axis .

Step 1: Choose K, K=2

Step 2: Initialize the centroids randomly

- c1=3

- c2=12

Note : As y=0 on the x-axis , d=|x2-x1|

Step 3.1: For each point calculate distance and assign it to the closest centroid

Data Point	Distance from C1=3	Distance from C2=12	Cluster Assigned
2	1	10	c1
3	0	9	c1
4	1	8	c1
5	2	7	c1
10	7	2	c2
11	8	1	c2
13	10	1	c2
15	12	3	c2

Cluster c1 contains {2,3,4,5}

Cluster c2 contains {10,11,13,15}

Step 3.2: Update the centroids

- c1=(2+3+4+5) / 4 = 3.5
- c2=(10+11+13+15) / 4=12.25

Step 3 should repeat until there is no change in the centroids.

Things to consider :

- It is always better to standardize/normalize the data points with any distance-based algorithm like K-means clustering, because different variables may have different scales, and that may affect the sum of squared error. Hence, it is a good practice to bring all the data points under one scale.
- We initialize centroids randomly, so different initializations may lead to different clusters. There is a

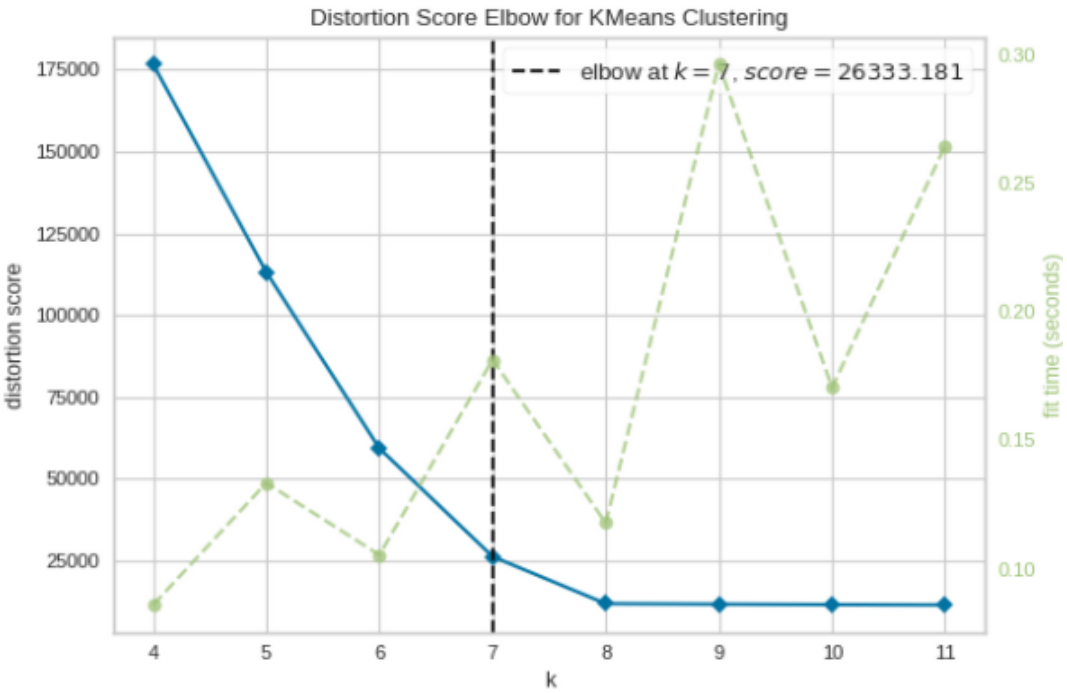
possibility that the algorithm falls into a local optimum rather than the global optimum. So it's better to iterate with different initializations and select the one with the least sum of squared distances within clusters.

Evaluation Methods:

For K-means clustering we need a fixed value of K, and that should be known before performing the method. It is not a learned parameter, and we will have to figure it out. To find out the optimal K number, one method we can use is called the ELBOW method.

ELBOW Method:

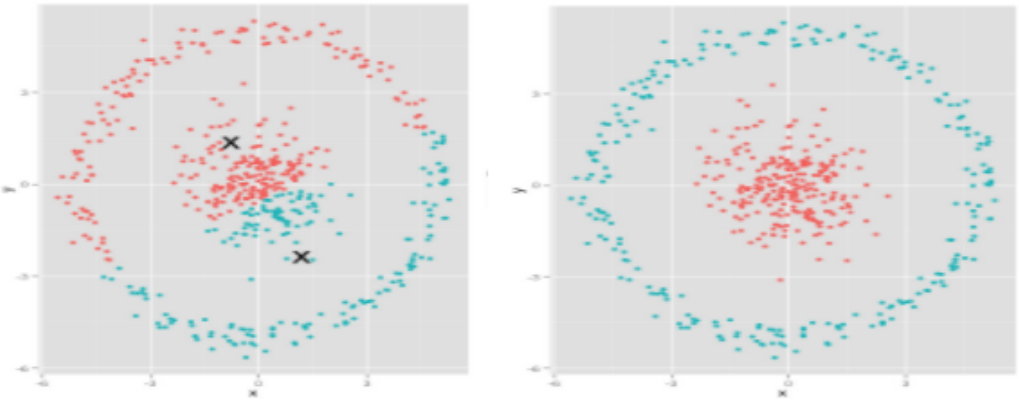
In this method, we take different values of K i.e from 1 to the range required. For each value of K, we calculate the sum of squares within clusters, usually called WCSS (Within Cluster Sum of Squares). So we calculate the sum of the squared distance between the centroids and the data points for each cluster and do a summation of all the clusters. Then, we plot WCSS vs K - the plot shape looks similar to that of an elbow. As we see in the graph, as K increases, the WCSS will decrease. The optimal point is at the elbow - after that as K increases, the fall in WCSS is not that significant. In the given example, K=7 appears to be the optimal value of K.



Source

Assumptions:

1. K-means Clustering is limited to linear cluster boundaries: The fundamental model assumptions of K-means Clustering (points will be closer to their cluster center than to others), means that the algorithm will often be ineffective if the clusters have complicated geometries



2. All clusters are of the same size.
3. Clusters have the same extent in every direction. This assumption would not be true in a dataset where different measurements are in different units.
4. Clusters have similar numbers of points assigned to them.

Image Source

Applications:

1. Document Clustering - Group similar documents together
2. Customer Segmentation - Divide customers into similar groups
3. Image Segmentation - Grouping similar pixels together

Previous

Next

← Go Back to Making Sense of Unstructured Data

Course Content

Kullback-Leibler (KL) Divergence

Sometimes, in machine learning, it is desirable to analyze the actual and observed probability distribution to quantify the difference in the distributions of a random variable. We calculate KL divergence to measure that difference.

First, let us understand what divergence is.

Divergence is a measure of how one distribution differs from another. It is not symmetrical in nature i.e. score of the divergence for distributions p and q would give a different score from q and p.

Here we will talk about one specific type of divergence, the Kullback-Leibler (KL) divergence.

KL divergence: It quantifies how much one probability distribution differs from another probability distribution.

The KL divergence between the two distributions p and q can be calculated using the formula:

$$D_{KL}(p||q) = \sum_{i=1}^{N} p(x_i) \cdot log\frac{p(x_i)}{q(x_i)}$$

Two important points to note about KL divergence:

1. The lower the KL divergence value, the better the two distributions match. If two distributions perfectly match, then it is zero.
2. It is not symmetrical i.e.

$$D_{KL}(p\,||\,q)! = D_{KL}(q\,||\,p)$$

The intuition behind the KL divergence score: When the probability for an event from p is large, but the probability for the same event in q is small, there is a large divergence. When the probability from p is small and the probability from q is large, there is also a large divergence, but not as large as in the former case.

To know more about KL divergence, please refer to this video here.

This video talks about the intuition behind KL divergence by explaining it with a numerical example. It also provides a real-life application of this concept.

Note: In Python, we use the rel_entr function to calculate the KL divergence.

← Go Back to Making Sense of Unstructured Data

Course Content

Unsupervised Learning

Data science and machine learning algorithmic techniques can be divided into three categories:

- **Supervised Learning:** The algorithm has prior data (labeled data) to learn from.
- **Unsupervised Learning:** The algorithm has no prior labeled data to learn from i.e., it learns from unlabeled data by finding patterns among examples and grouping them accordingly.
- **Reinforcement Learning:** The algorithm learns patterns or behaviors in a recursive manner by taking actions to maximize a certain reward, and uses feedback from the environment to learn the best action to take under the right circumstances in order to maximize that reward.

In the coming lecture, our focus will be on unsupervised learning.

Unsupervised Learning is a category of techniques that train computers to use a set of unlabeled / unseen data and learn by themselves. The algorithm is provided with a large volume of data and expected to identify hidden patterns. Since there's no prior information / label available for each data instance, there is no defined / correct outcome. The machines only need to determine if there are any patterns in the given data.

Unsupervised Learning is utilised in the following use cases:

1. Clustering
2. Association
3. Dimensionality Reduction

Clustering:

In clustering, the key idea is to divide the data into groups such that *each group shares similar properties and each group is as dissimilar as possible to the other groups.*

Let us now try to understand clustering using a simple example:

Let us now try to understand clustering using a simple example:

An organization wants to make a marketing strategy for its new product. They want to segment the customers into different groups in order to target the right audience. They have made segments:

1. based on customer demographics like occupation, age and gender

2. based on income and spending habits

3. based on geographic locations of the customers

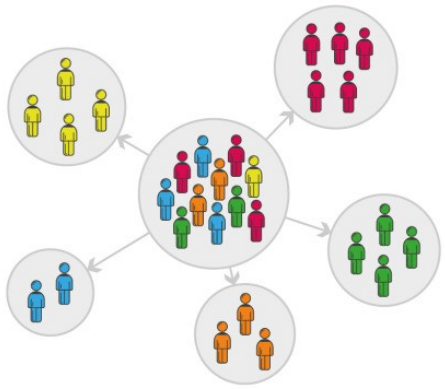


Image source

This is **unsupervised learning.** There was no labeled information about the people, just various features describing their characteristics such as age, gender, income, etc. The algorithm simply found groups of people that may be similar to each other based on these features and organized them into these groups. Due to the absence of labels, we don't know what exactly these groups represent, just that the algorithm found the people in each group similar to the others in that group for some reason, and dissimilar to people from other groups for another reason.

Association:

Association learning is a rule-based machine learning and data mining technique that finds important relations between variables or features in a data set. It checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable.

Dimensionality Reduction:

Dimensionality Reduction is the transformation of data from a high-dimensional space into a low-dimensional space so that the low-dimensional representation retains some meaningful properties of the original data, ideally close to its intrinsic dimension.

What is Supervised and Unsupervised Learning?

Supervised Learning:

Suppose, you're doing a task under someone's supervision. So, someone is present there to judge whether you're getting the right answer or not. Similarly, in supervised learning, we have labels for all the data points and the model learns from that data.

Example:

Suppose, we have an email dataset and all the emails are labeled with either "Spam" or "Not Spam". So, each of our data points has one of the two labels "Spam" or "Not Spam".

Now when we get a new email, the job of the Machine Learning algorithm is to predict whether the email is Spam or Not Spam.

This is a Supervised Learning Problem.

Unsupervised Learning:

In unsupervised learning, we try to discover hidden patterns in data but we don't have any labels. The dataset here is a collection of examples without a specific desired outcome or correct answer.

Example:

In our email inbox, we have thousands of emails. And, say we want to read at any moment all the emails which are on the same topic. And we don't have any labels on the emails like Spam, Not Spam, etc.

So, we want to run a machine learning algorithm that groups together similar kinds of emails.

Now after running the algorithm, we find some natural groups of emails in the inbox, for example, emails about homework & classes, emails about hiking & adventures, etc.

So, this is an Unsupervised Learning problem where we don't have any labels on the dataset (i.e emails). But after running the algorithm we find some natural grouping of emails.

Some other examples of Unsupervised Learning are:

1. Google groups together all the articles that are about a similar story. (Google News)
2. Facebook trending stories are also just groupings of lots of individual articles and posts.

Challenges of Unsupervised Learning:

- Sometimes it's really expensive to get labels on the dataset.
- It's easier to understand how well a supervised learning algorithm is performing. We have to just check for example what percentage of spam and not spam emails are getting a label. But in the case of Unsupervised Learning, it's not that straightforward to check whether the emails are getting a good grouping by topic.

Unsupervised and Supervised Learning Problems

Let's look at some Data analysis problems and state whether they are supervised or unsupervised problems:

Problem 1:

Suppose we have measured the concentration in the air of a certain type of pollution. We have measured the amount of pollution at different locations, under different weather conditions at different times of the day and for different days of the week.

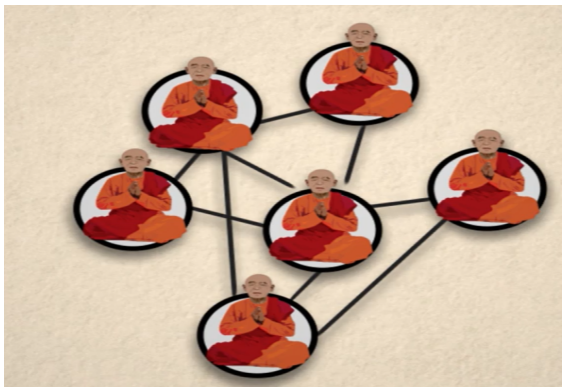
Now we would like to predict the air pollution concentration at a new location where weather and date are given.

This is an example of **Supervised Learning**. Here the data points are the collection of information about the surroundings like Location, Weather, Date & time. And the labels are air pollution concentration which is a numerical value.

Problem 2:

When Frank Simpson was a Ph.D. student in the 60's he spent some time in a Monastery. He followed the novices' training there and recorded their interactions with each other.

Now we want to discover what the friendships or social groups among the novice monks are.



This is an **Unsupervised Learning problem**. As here, we don't have any labels that tell us who belongs to which group and are there any social groups that exist.
Here we have to find the pattern in the data in an unsupervised way.

We have some similar kinds of problems like this one. For example: Detecting communities or groups in large social networks like Facebook or Twitter.

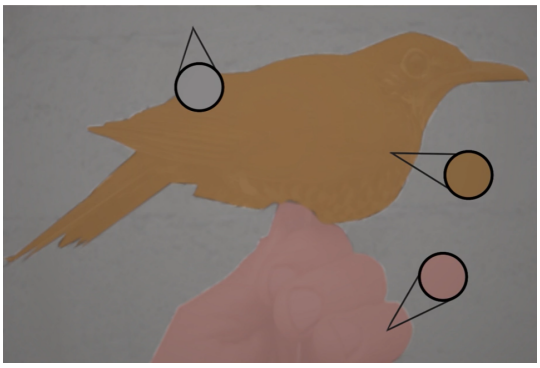
Problem 3:
NASA uses rocket boosters to launch spacecraft into orbit. It's expensive to build a whole new rocket booster for every slight tweak and it's difficult to recreate real atmospheric conditions in a wind tunnel for testing these out. So, NASA uses computer simulations, but it takes a long time to run the computer simulation for different rocket boosters specifications.

So, NASA would like to measure the simulated lift, for different rocket booster specifications and predict the simulated lift under other given specifications.

This problem is a **Supervised Learning Problem**. Here the data points are the rocket booster's specifications and the labels are the simulated lift.

Problem 4:

This problem is a little different than the previous problems.



Suppose, we have small cameras that we have installed to monitor bird's nests in New England. But, we don't have a lot of Bandwidth to send the video. So, we have to compress the recorded video before sending it.

We can think of this as an Unsupervised Learning Problem. If we consider the pixels at each timestamp as a data point, then we can group together the pixels which are similar in color and similar in time to achieve the compression.

What is Clustering?

We know that Unsupervised Learning means finding hidden patterns in the data when there are no labels present in the dataset. There are a lot of different problems that we can solve using Unsupervised Learning.

But the most popular version of Unsupervised Learning is **Clustering**.

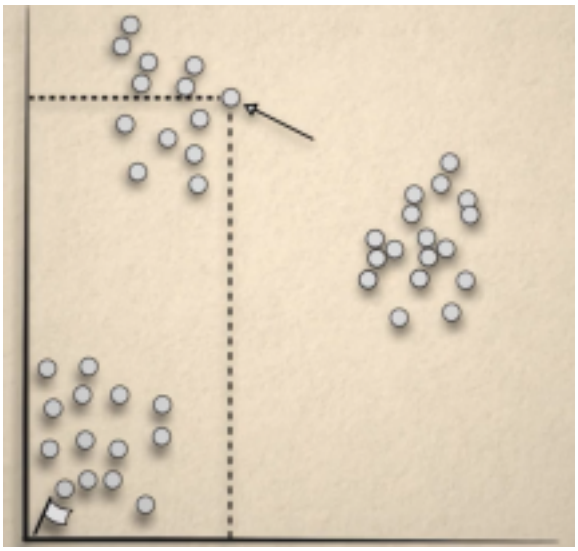
What is Clustering?

In clustering, our goal is to group the data according to similarity.

Let's take an example to understand the keywords **Group**, **Data,** and **Similarity** in more detail.

Data:

Imagine you are leading an archaeological dig. And every time someone on your team unearths an artifact, you have them record the position of the artifact.



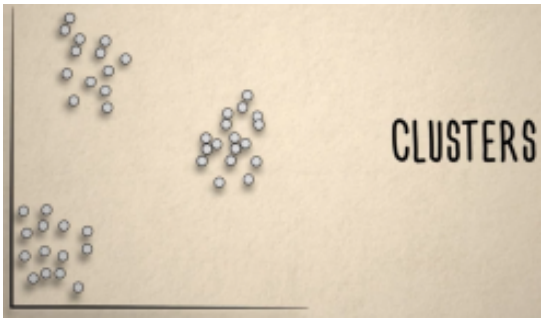
In particular, you can put down some marker near the archaeological site and you can record how far north and how far east each artifact was found relative to your marker. The location for one artifact will be a data point, and when you list all of the locations of the different artifacts, then you get the dataset.

Suppose you consulted with a few historians before your archaeological dig and you come to know that three families lived in the area that you are exploring.
Now, from the first figure, it is easily identifiable where the three families lived and which artifact belongs to which family. To understand the process even better, we need to talk about "**Similarity**".

There is a notion of similarity here. In this archaeology example, we say that two artifacts are similar or close if they are close in terms of physical distance. When we think about where the three families lived, we are grouping the data according to this distance.

Here, we are assigning each data point to only one family group. So, data points that are physically close, tend to belong to the same group.
When we assign each data point to only one group like this, then we call the groups as **Clusters.**

So, Clustering is the Unsupervised Learning. It is the problem of assigning each data point to exactly one group or cluster. Also, we want the clusters to be meaningful, that is we want the clusters to give us some interesting insights into our dataset.

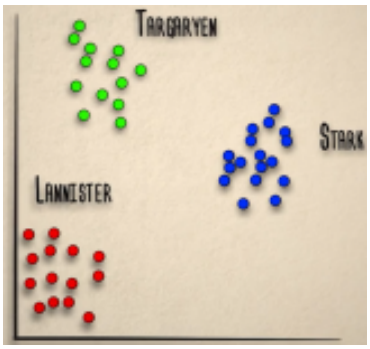


In the example that we have discussed, we imagined the clusters correspond to 3 different families who used to live in the examined area.

The data points in any cluster are for example pottery shards or other items that we Believe that a single-family had discarded when they lived at that spot. Identifying the items that we think belong to one family will let us perform further scientific or archaeological or historical analysis of the artifacts.

Classification vs Clustering:

We already know about Supervised Learning Problems. There is a Supervised Learning problem that is somewhat similar to clustering. The problem is called **Classification.**
In classification, like in Supervised Learning in general, we are given labels.



For instance, consider the archaeology example, if we run classification on this data, we might not only know that there are 3 families, but we might know their names as well. For example, the names may be Lannister, Stark, and Targaryen. Here the labels are Categorical. So, the labels have no order.
For the archaeology example, if we had a classification problem there, then we would have been given a bunch of data points with labels. For instance, we might imagine each family actually made a lot of pots with a sigil or some other identifying mark.

Then our goal would be to find labels for any remaining unlabeled artifacts. So, our goal will be to predict labels for new data points given the labels of all data points.

But in **Clustering,** we don't know the families or their names. We might know that there are 3 families, but we don't know anything at all beyond that number. And none of the artifacts come with labels. So, we have to discover how to group the artifacts from their locations alone.

In many cases, labels may be hard to come by for various reasons. For the archaeology example, we might not be lucky enough for families from thousands of years ago to have carefully labeled their pottery for our benefit.

But on the positive side, Clustering lets us find hidden groupings in data even when we can't run classification. But Clustering can be a more difficult undertaking than Classification, since we don't have the information contained in the labels.

So, the Clustering problem may get complex when we deal with higher dimensional data.

Here for the archaeology example, we have seen that our data was numerical. But our data need not always be numerical. It might take the form of words or pictures or genomes or something else entirely.

And it is not so easy to immediately see the hidden patterns in these types of data. But it turns out that we can still get a lot of information about the data from Clustering.

Why and When to use Clustering?

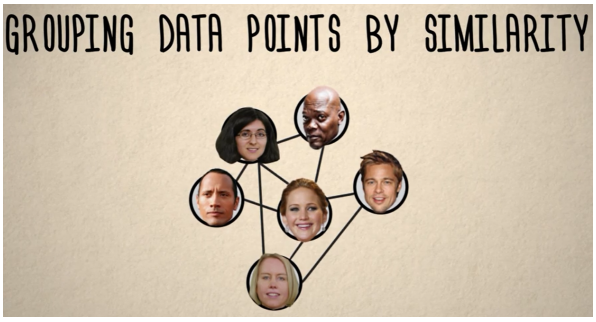
We already know that clustering is a particular form of unsupervised learning, and generally, in unsupervised learning, we try to find any hidden pattern in a set of data points without labels.

In clustering, we want to find a latent grouping such that each data point belongs to exactly one group called a cluster.

We have already seen an example of clustering in action, which is how we might use clustering to find the arrangement of artifacts at an archaeological dig.

Now we will look into some more examples of clustering and understand why and when we should use clustering in practice.

Sometimes we have a collection of data, but we are not quite sure what to do with it yet. We might want to explore the data without a particular end goal in mind. Perhaps the data will suggest some interesting approaches for further analysis.



In this case, we say that we are performing Exploratory Data Analysis.

For instance, suppose we have some data points of users on meetup.com. Roughly each data point corresponds to a person. In order to apply clustering, we have to define a **notion of similarity** too.

Suppose, the similarity is the number of common interests between two people. We can't actually feed a user or person into our computer. So, we need to be more precise about what constitutes a data point in this case.

For the data points, perhaps what we have actually collected for each person is a list or vector of whether that person is interested in each meetup.com group on the meetup website.

With this dataset, it is easy to calculate the similarity between two users. **It's the total number of groups those two people are both interested in.**

After we apply clustering, we might hope to find groups of users with broadly similar interests. Maybe after running clustering, it may turn out that, for example, some of the users are interested in musical performance, some are interested in outdoor sports. Also, some users are interested in hackathons and so on.

So, we have found some patterns or groupings from our dataset.

Now the question is,

What are some other times when we might use clustering?

So basically, we can use clustering for problems that kind of look like classification, but we don't have labels in advance for the dataset.

***Example**:*

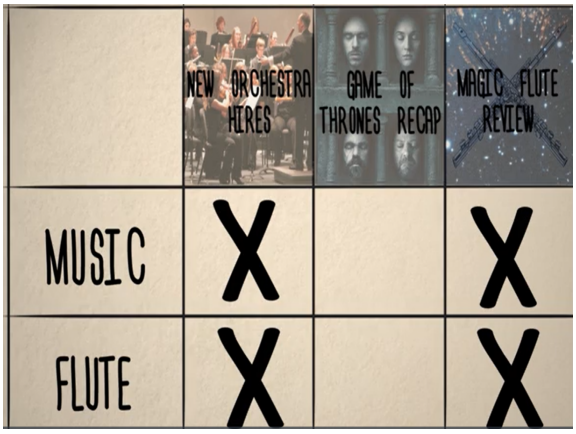
Suppose we have all the documents on Wikipedia or all the past articles from WIRED magazine. Now we would like to find out topics or themes that are represented in these documents.

In general, we don't know the possible topics in advance. So, we might guess at some topics. But we can't be sure in advance that we have got all the topics that would really appear in the corpus.

So, to discover the topics we can apply clusters. We can say each data point corresponds to a word and **we can measure the similarity of two words by counting how many documents they both appear in. And we can group together similar words to form the cluster.**

Here the data points that correspond to any particular word could be a list of vectors of whether this word occurs in each document, by labeling '**Yes**' or '**No**'.

Then we can actually calculate the overall similarity of the two words.

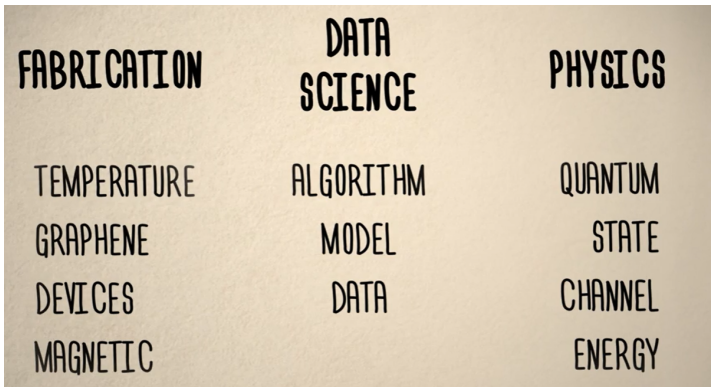


Let's see another example:

Some students at MIT looked at research abstracts from Professors of the Electrical and Computer Science department. And they found some patterns of some words which tend to co-occur.

Words like temperature, graphene, devices, and magnetic go together. Words like algorithm, model, and data go together. And words like quantum, state, channel, and energy go together. It turns out that there are some groupings among these words. The first set of words are related to fabrication, the second set of words are related to Data Science and similarly, the last set of words are related to physics.

The interesting thing here is, the students didn't use any knowledge in advance about what MIT professors are working on.



The different topics in the data came out in an automated way.

So, this is one of the examples where we don't know the labels of our clusters in advance.

But sometimes we don't have labels on the data because it's expensive to label data. It might take a long time to manually label each data point. And the labels might be changing too quickly, so it's tough to keep up with it.

Take the example of Google News. We might want to put news documents about similar topics in a cluster. But the topics in the news are changing every day. For instance, just hours after a new Game of Thrones episode comes out, there are many great articles available about that episode.

We would like to detect all those articles and group them together. And also we would like to do this even when a surprise wild event happens, which we didn't anticipate in advance.

In this case, our data points might be a list of the topics which occur in a given document, and similarity measures between two documents might be accounted for by looking at how many topics are shared by the two documents or how similar the topic proportions are between the two documents?

Alternatively, it might also be expensive to label data simply because there is way too much data available.

So in summary, here in this article we have seen a lot of different reasons to use clustering. We have also seen a lot of different types of data where we might apply clustering.

K-Means Preliminaries

K-Means Clustering is not only the most popular algorithm for clustering, but quite possibly the most popular algorithm within Unsupervised Learning.

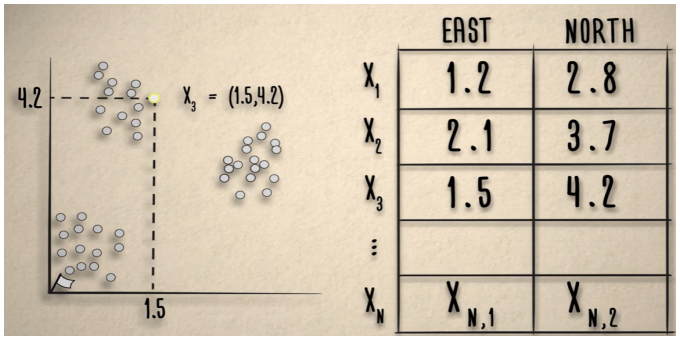
The reasons why K-Means is popular are:

- It's fast. The steps of the algorithm are simple and quick to run.
- The algorithm can easily take advantage of parallel computing by running calculations simultaneously across multiple cores or processors. So, we get even further speed up the running time.
- It's fast in programmer time. The algorithm is straightforward to understand and to code. Also, it doesn't require large numbers of parameter tweaks like some other algorithms.

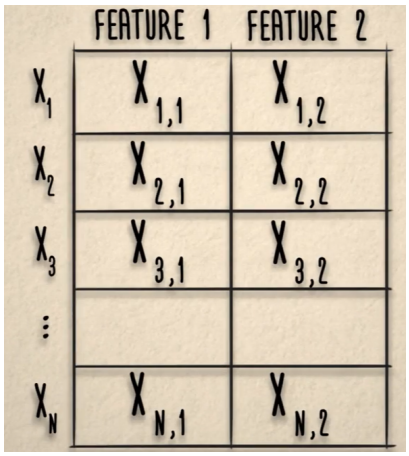
The Set-Up and Assumption of the K-Means Clustering Problem:

Assumption:

- The K-Means algorithm assumes that we know the K-value in advance.
- It also assumes we can express any data point as a list(vector) of continuous values.



Example: From the previous archaeology example, we can write the location of an artifact as a list or vector with two elements, which are the distance in east(in meters) from a marker near the site and the distance in north from the marker.

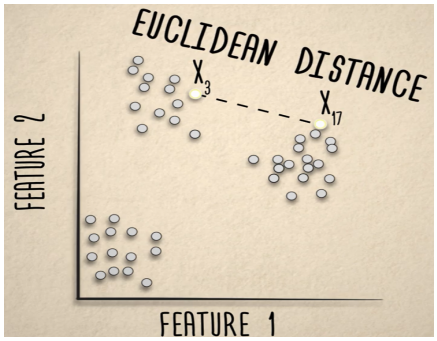


For example, in the figure the datapoint X3 is 1.5 meters east and 4.2 meters north from the marker.
We have one vector for each data point. So, the dataset is the list of all these vectors.

Here, we have N data points. In general, the information that we collect for any data point does not have to be a distance relative to some marker in an archaeological dig.
Usually we just call each component of the vector a feature.
In general, we might have any number of features.

Now we know that clustering is all about grouping the data according to similarity. We have defined what a data point is for the K-Means Clustering problem. Now we will see more elaborately what **Similarity** means in K-Means clustering.

It's equivalent to defining dissimilarity for two data points instead of similarity.
Here we say, the dissimilarity between two data points in the archaeology example, is the physical distance that is the length of the line connecting them. This length is sometimes called the **Euclidean Distance**.



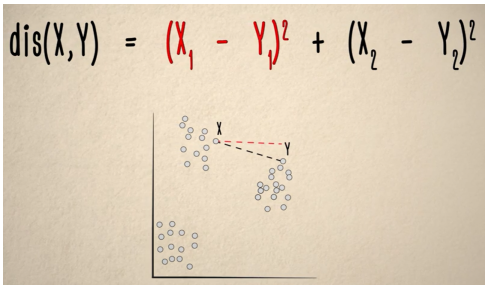
- The Euclidean Distance between two points (X1, Y1) and (X2, Y2) is calculated as:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}.$$

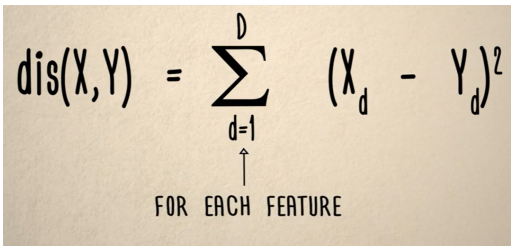
One pair of data points that is farther in Euclidean Distance than another pair, will also be farther away in the Squared Euclidean Distance.

In this case, it turns out that we get the same result from K-Means Clustering if we define dissimilarity as **Squared Euclidean Distance.**

- The Squared Euclidean Distance is just the square of Euclidean Distance.So, for two data points X(X1, X2) and Y(Y1, Y2) it is given by:



- A more convenient way to write the formula is using summation:



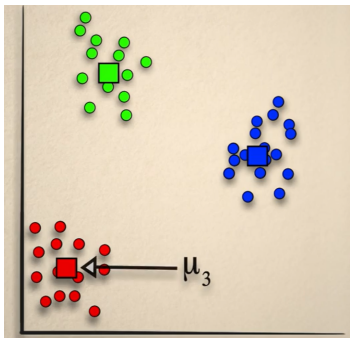
Now, we have our data, we have also defined "**Dissimilarity**". Now we need to group the data.

The question now is: **What is the output we expect from our algorithm?**

One thing to keep in mind here is that, the “K” in the K-Means refers to the number of clusters.

Like, in the archaeology example, K is equal to 3.

- Now for each of the clusters, we plan to get out a description of the cluster that is what the clusters looks like, which data points belong to it, etc. To get a clear picture, here we get a **Cluster Center** for each cluster.
- Suppose, the cluster center for the below cluster(in red) is μ3, and now we want to know which data points are assigned to each cluster.
- Let S_3, be the set of data points assigned to the cluster with cluster center μ_3.



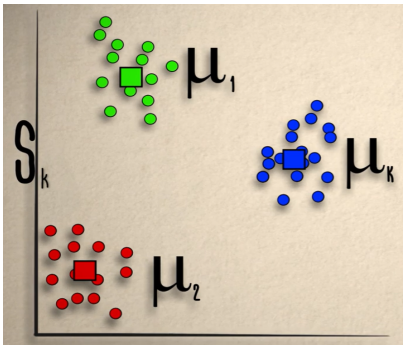
A Few Points to Note :

- In clustering each data point has to be assigned to exactly one cluster.
- The K-Means Clustering problem is more specific than clustering in general. We assume that the data points can be expressed as continuous numbers, which is not really true if we have a “Yes or No” vector. Even if we encode “Yes” as 1 and “No” as 0 we can’t get any value other than those two in our vector.
- We have also assumed that there are exactly K clusters, that is the value of K is known in advance, but this is not always true in applications.

The K-Means Algorithm

We have seen the K-means problem as a particular subset of general clustering problems. We have assumed that each data point is a vector of continuous values and Square Euclidean Distance is used as the dissimilarity measure between two data points.

In the last example we have seen, the output of the K-means Clustering problem should be a set of cluster centers which are μ_1, μ_2, ... μ_k and a set of assignment of data points to clusters. For example, S_k is the set of data points assigned to the cluster μ_k.



The K-means clustering problem tells us exactly how we should choose the cluster centers and how to assign data points to clusters.

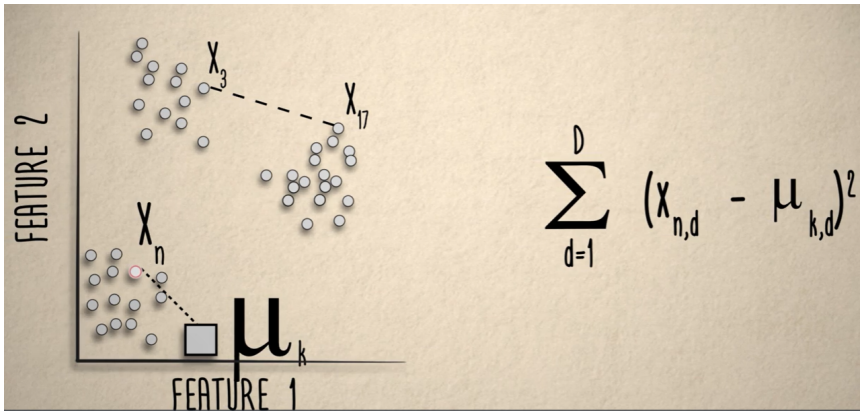
As we want to group the data according to similarity, the idea here is to **minimize dissimilarity** within each cluster.

We know how to calculate the dissimilarity between two data points.

Now we need to calculate the Global Dissimilarity.

Global Dissimilarity :

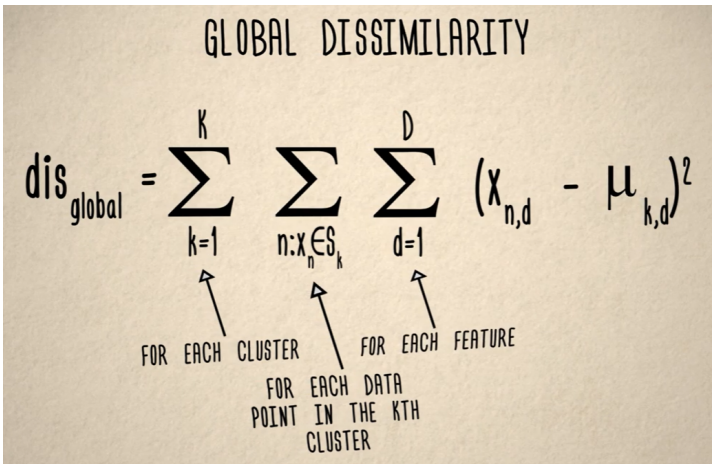
The dissimilarity between any data point and a cluster center is the distance from the



data point to the cluster center. We have used Squared Euclidean Distance here.

So, the dissimilarity within a cluster is the sum over the distances between data points in that cluster and the cluster center.

Finally the global dissimilarity is the sum over the cluster dissimilarity.
So, we calculate the Global Dissimilarity by iterating over each cluster, over each data point within the cluster and over each feature of the data point.

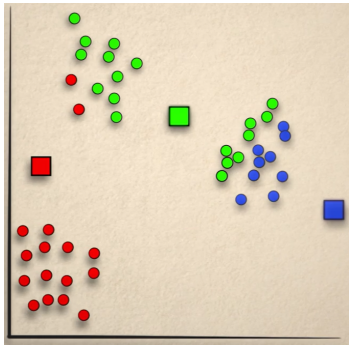


So, the main target of the K-Means Clustering Algorithm is to minimize the global dissimilarity which is also called the **K-Means Objective Function.**
We minimize the objective function by choosing a set of K-clusters and by properly assigning data points to clusters.

We are still assuming that we know the value of K in advance but we generally don't.
But the K-Means clustering algorithm or LLOYD's algorithm seems to perform very well in such a situation, thus solving this problem in a way.

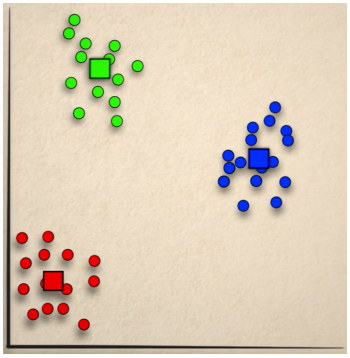
The K-Means Clustering Algorithm:

We don't know the cluster centers in advance. So, we start by initializing the cluster centers to some values. Then we alternate between the below 2 steps:



1. We assign each data point on the cluster with the closest cluster center.
2. We update the cluster centers to be the mean of all the data points in the cluster.

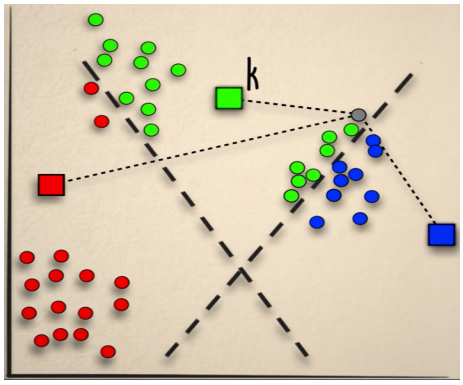
We iterate through these two steps until we can't make any more changes.
And finally, the cluster center's position looks something like this.



One important point here is:
How do we initialize the cluster centers at first?

- One option here is to draw the cluster centers uniformly at random from the existing data points.
- Some more sophisticated initialization methods are available, which is followed in K-means ++.

But, no matter how we choose to initialize the cluster centers, once we have some values for the centers, we can start iterating the main steps of the algorithm.



So, first we assign each data point to the cluster with the closest center. For doing this for each data point, at first we compute the distance to all K cluster centers. The cluster center for which the distance between the data point and cluster center is minimum is the cluster center to which that data point gets assigned.

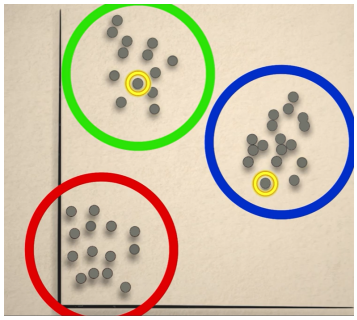
The distance calculator can be done completely separate for every data point. So, it's extremely easy to divide up the data points across cores or processors and perform these calculations separately to speed up the running time. This is called **Embarrassingly Parallel Computation.**
So, after the assignment of data points to clusters, we recalculate the cluster centers for every cluster by taking the mean of all the data points in that particular cluster.
That is, for each feature, we sum up the values of the data points in the feature and divide it by the total number of data points in the cluster.
One significant thing here is that, luckily the K-means algorithm always stops in finite time.
In summary in this article, we get familiar with what the K-Means clustering problem is and how to develop the algorithm to solve the problem.

How to evaluate clustering?

We already know how to develop a K-Means algorithm. Now the important thing is to evaluate the output of a clustering, that is we want to know how good the output of the algorithm is.

Let us see, how to evaluate a clustering of a data,
Let's first talk back about the Supervised Learning problem of classification.
There we were given a set of data with labels and we wanted to predict the labels for some new data. So, for classification, the evaluation is straightforward if we have enough data.
There, we typically set aside some data where we know the labels. The data points that we feed to the algorithm together with their labels is called **Training Data**. And the data points that we set aside for evaluation purposes are called **Test Data**.
We use our classification algorithms to predict the labels on the test data and compare them to the true labels. Then we have an absolute universal scale to check what percentage of the test, did the algorithm predicted correctly.
We can easily compare other algorithms. A better algorithm gets more labels right. But we can also see whether an algorithm on its own is good or not. Did the algorithm get a significant proportion of the labels correct or not?

Now sometimes in Clustering, one may have access to a ground truth clustering of the data. In the archaeology example, perhaps in some cases, the artifacts clusters were known. So, we can compare to the known clusters or perhaps someone was paid to cluster some images manually by hand.



It's still not quite as easy to evaluate clustering as evaluating a classification.
Since, any permutations of the labels leads to an equally good clustering.
The only thing that defines the clustering is whether two data points belong to the same clusters or to different clusters in both our algorithm and ground truth.

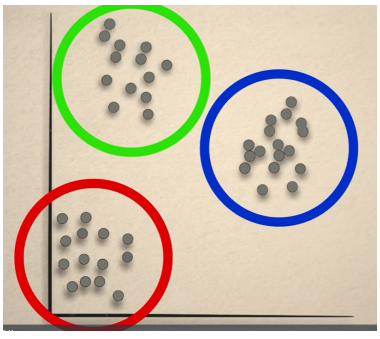
For example,
In figure, the two marked data points (in yellow) belong to different clusters.
There are some measurements like **Rand Index** and **Adjusted Rand Index** that help us measure whether the clustering found by our algorithm really captures the information in the ground truth clustering.

They calculate it by counting the pairs of data points that belong to the same or different clusters according to the algorithm and the same or different clusters according to the ground truth. Then the value is normalized.

Measures like the Rand Index are called **External Evaluation**, because they require **outside information** about a **ground truth clustering.**
The problem of evaluation is still even trickier in Clustering, because there is rarely any ground truth information that we can use for testing.

Then, how to evaluate whether a clustering is good or whether one cluster is better than another?
The answer is, we can check K-Means Objective function value across different clusters. It lets us compare two clustering. But it does not tell us whether a particular clustering is good or not on its own. It also might not capture exactly the patterns which we are looking for in the data.

So, the evaluation of Clustering is Hard! But, researchers have developed a number of useful heuristics.
The first and most useful observation is that, we are often trying to find some particular meaning, latent pattern in our data via clustering.
We might be able to check the pattern by visualizing the result of the clustering.

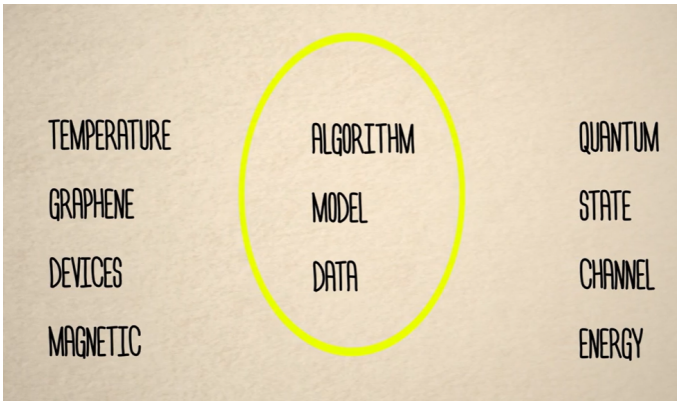


For instance:

- For the archaeology example, it was obvious what the cluster should be.
- Similarly for the image segmentation example, we would like our clustering to distinguish the three

different types of objects in the image. And by visualizing the clustering of points we can easily check this.

- Consider the topic modeling example, where the words were clustered together. We can list out the clusters of words and ask ourselves do these words go together or do the words have any meaningful association ?



Another option we have for evaluation is to use special tools for visualization. Like the GGOBI tool.

We might have a dataset where D (= Number of dimensions) is much higher than 2.
For example we might have collected data on a large number of different health metrics like age, daily calories consumed, daily water consumed, weekly alcohol consumed, weekly miles driven, weekly exercise in minutes and so on.

But this data is not an image or a text data set. So, it might be hard to plot meaningful pictures of the data.
GGOBI tools help us find meaningful visualizations of high dimensional data for evaluating the clustering.

While these methods are often more qualitative than quantitative on their own. They can be made more quantitative by incorporating some form of crowdsourcing. For example, Amazon Mechanical Turk workers could be asked to weigh in on the quality of the clustering. But even with the help of Amazon Mechanical Turk or other platforms human evaluation can be expensive in terms of both time and money.

So, we may consider some other automated forms of evaluation.
By contrast to External Evaluation, there are also a number of measures for internal evaluation that depend only on the data at hand.

The basic idea of this measure is to typically make sure that data within a cluster are relatively close to each other and data in different clusters are relatively far from each other. An example of this is the Silhouette Coefficient.

Another example is to split the data into two data sets and then applying clustering to each and then computing the clustering found across the two sub data sets.

So, in this article, we have discussed some of the ways to evaluate the output of the K-means Algorithm and clustering algorithm in general.

Beyond K-Means: What Really makes a cluster

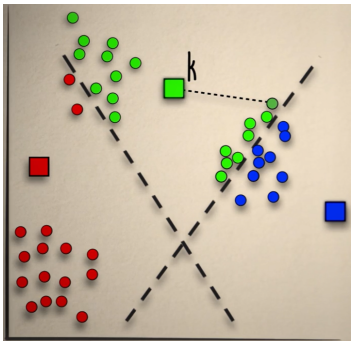
We already know about the clustering problem and the particular algorithm for coming up with the solution to it, which is K-Means algorithm or LLOYD's algorithm.
Now what happens if the output of the K-Means algorithm is not what we expected or wanted?
So, in this article we will discuss how to troubleshoot K-Means and ways to make K-Means clustering better.

- The first thing to do is to make sure we are getting the best possible output from the K-Means algorithm.
- We know the K-Means algorithm eventually stops and there are only finitely many possible clusterings. And the K-Means algorithm moves to a new clustering only if the new clustering decreases the K-Means objective function.
- But just because the algorithm will stop eventually, it does not mean the algorithm will stop quickly.

In practice, the K-Means algorithm tends to be quite fast. But we may find that, in a particular dataset the algorithm is taking a while to run. So, what to do in such a situation?

Various speed ups methods have been proposed in such situations.

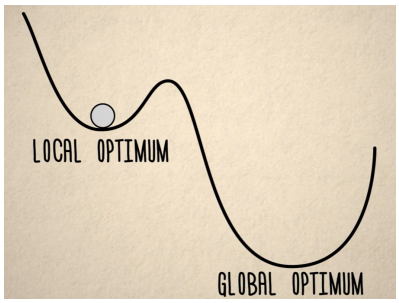
- In the K-Means algorithm we need to calculate the distance from every data point to every cluster center. But, we don't actually need to check every possible cluster center to decide which is the closest cluster for a given data point. Here we can easily observe that the data point is closest to the cluster center in green (Distance is shown in dotted line).



This observation relies on the fact known as the **Triangle Inequality**, that lets us ignore cluster centers that are relatively far away from a given data point in our calculation.

Another related issue is that when the K-Means algorithm stops, it does not necessarily stop at a minimum value of the K-Means objective, which is Global Dissimilarity. When the K-Means algorithm stops at a value that is not the minimum value of the K-Means objective function, then we call it Local Optimum.

But it would be better if we could reach the true minimum value, that is the **Global Optimum**. But that is a very difficult problem which we can't solve in general.



One option is to try to get closer to the global optimum. We can try to do this, by running the K-Means algorithm multiple times for multiple random initialization.

Then we finally take the configuration of cluster centers and cluster assignments with the lowest objective function value, that is the least global dissimilarity.

This process does require a more total computation time, but these multiple runs can be performed completely in parallel, which makes the process less complex.

Another option to get better output from our algorithm is to use **smarter initialization**, like in **K-Means ++**. K-means++ is designed to improve the centroid initialization for k-means. The basic idea is that the initial centroid should be far away from each other. The algorithm starts by randomly choosing a centroid c_0 from all data points. For centroid c_i, the probability of a data point x to be chosen as a centroid is proportional to the squares of the distance of x to its nearest centroid. In this way, k-means++ always tries to select centroids that are far away from the existing centroids, which leads to significant improvement over k-means.This also has the potential to reduce total running time.

Now suppose somehow we had access to the global optimum, that could know the actual set of cluster centers and assignment of data points to clusters that minimize the K-Means Objective function but we could still be dissatisfied with our result. This is a topic which we will see in later articles.The idea here is, in this case the issue is not our K-Means algorithm but the K-Means problem itself.

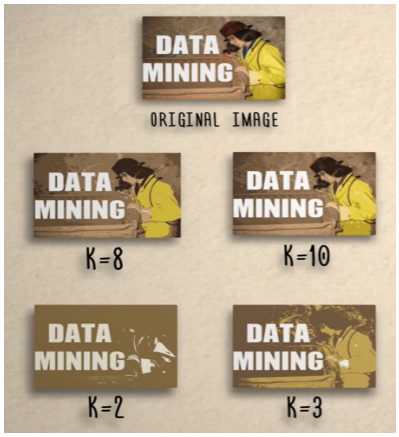
Examples of K-Means Clustering Problems

As we know, Clustering is basically grouping data according to similarity. In this article, we will see some different notions of grouping.
One of the most immediate and obvious issues with K-Means clustering in a variety of applications is that it requires the user to specify the number of clusters K.

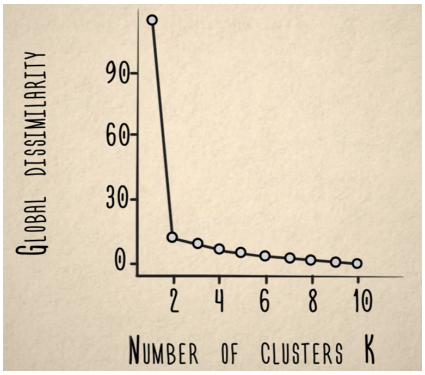
Sometimes this is not a major problem. For example, K-Means can be used for image segmentation. In this application each data point might be Red, Green and Blue, which are the intensity values for each pixel.

The K-Means algorithm finds K colors that can be used to approximate the true and typically much wider range of colors in the image.

Here, K might be determined by how much we want to compress the image.
We can see in the image that, increasing the value of K increases the quality of the compressed image in this case.



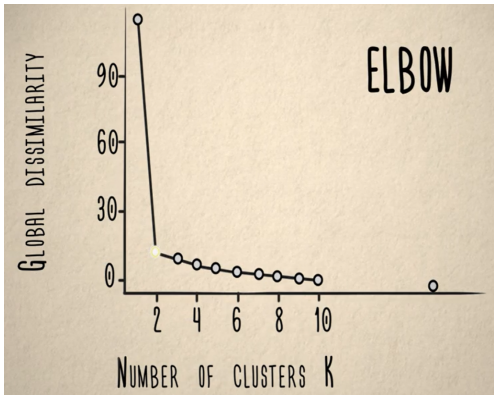
In many other cases though, K is unknown in advance. Suppose we get some new data and we need to determine both K as well as the latent clusters. One option is to use popular heuristics.



One heuristics arises from plotting the global dissimilarity across a wide range of values of K.
We can't choose K immediately from such a figure because the K-Means Objective function will always decrease as we increase K.

If we just choose K as the value that minimizes the K-Means objective function, we would have to choose K equal to the number of data points i.e N. But in that case it would become useless clustering.

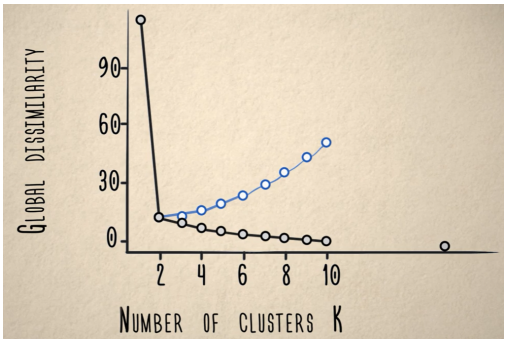
An alternative heuristics is to look for an ELBOW in a plot of Global Dissimilarity vs the number of clusters. Elbow point is a point in the figure where the gain from the new clusters suddenly slows down.



We can set K to be the number of clusters at this Elbow. In the figure which is K=2.

Other more theoretically motivated options for choosing the value of K exist as well.

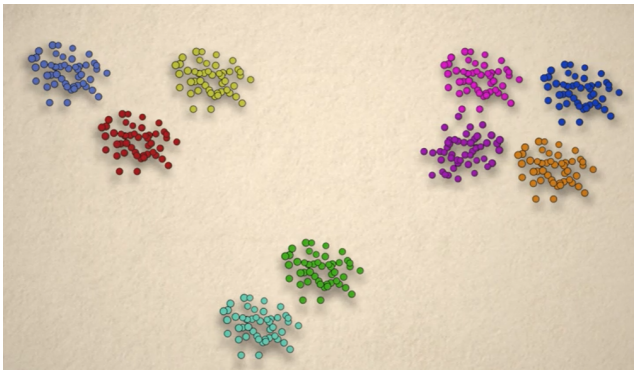
These methods include Gap Statistic as well as methods that change the objective of K-Means. The idea of this method is to explicitly account for the fact that we pay a price for more complex models.



In particular, we can start with the original K-Means Objective function and add a penalty term that grows with the number of parameters.

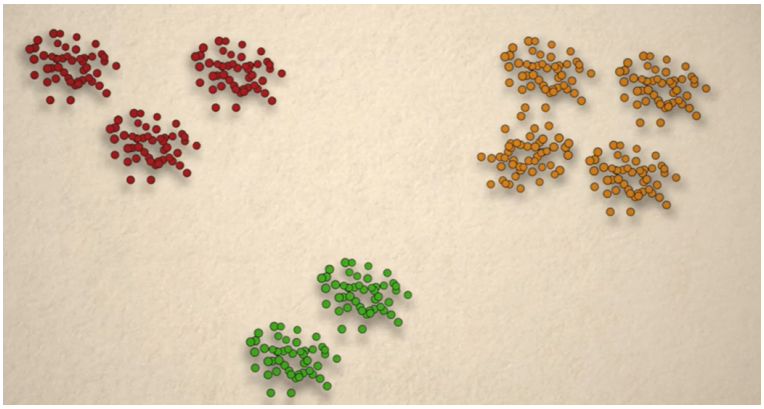
Some appropriate penalties are given by AIC, BIC or other so-called information criteria.

Once these penalties are added, we can plot the new objective function as a function of K, and now there will typically be a non-trivial minimum. We can choose K to be the number of clusters of this minimum.
There are also some more broader issues in choosing the value of K. Sometimes there is not a clear right value for K.



Suppose, we have a bunch of data on various organisms that we have studied in some environment. We might find that if we favor smaller clusters (see in fig), then we cluster the organisms into species and it's a great clustering.As we have picked up an interesting and meaningful latent grouping of the organisms.

But if we favor larger clusters, then we will end up clustering the organisms at the genus, family or order level which are also all useful and meaningful classifications used in biology.



So, clustering at any of these levels would give us an insight into the problem, but each level would result in a different number of clusters.

In this case, it might be useful to try a hierarchical clustering approach, such as **Agglomerative Clustering.**
These approaches explicitly model that some clusters might be composed of further sub-clusters.
So, all the clustering that we have talked about so far puts each data point into only one cluster. This is called **Hard Clustering.**

We also sometimes have clusters that are not perfectly separated. For instance, some data points might be on the border between two or more clusters. And we might not be sure about which cluster these data points should belong to.

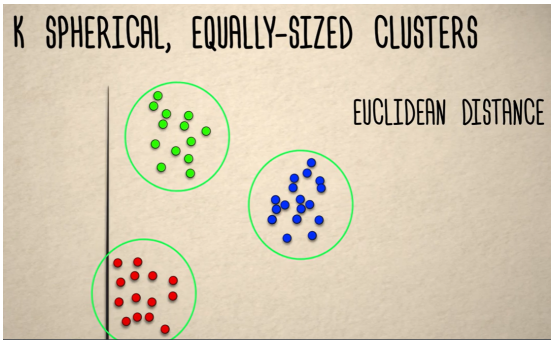
An alternative to Hard Clustering that allows us to solve this pattern is **Soft clustering**. In soft clustering, we might allow each data point to have a different degree of membership in each cluster. For instance, a data point might have a probability distribution over its belonging in different clusters. For many data points, this probability distribution might express that we are very sure that the data point belongs in one particular cluster. But in other cases near the border between clusters, the probability distribution for a data point might favor one cluster, but also make it clear that the data point could reasonably belong in another cluster.

In the soft clustering case, we might use alternatives like Fuzzy Clustering and Gaussian Mixture Models instead of K-Means clustering.

Beyond K-Means: Other Notions of Distance

K-Means clustering is a powerful framework for clustering. But we have seen some instances where K-Means clustering is not always the best clustering framework for a particular problem.

As we know clustering is about grouping data by similarity and in the last article we have briefly looked at how we might be interested in different notions of groupings or clustering beyond K-Means.



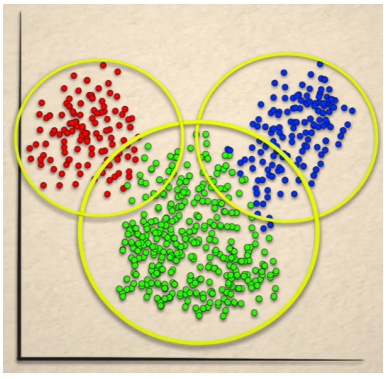
Here,we will discuss how some time we define **Similarity** differently than the definition used by K-Means clustering.

K-Means clustering assumes that we have K spherical equally sized clusters. This follows from the use of the Euclidean Distance as the dissimilarity measure for K-Means.

Using the Squared Euclidean Distance in our objective function is equivalent to using Euclidean Distance and does not change the fact.

So, what is the problem with the K-Means definition of similarity?

First consider the case where our clusters are not equally sized. For instance consider an example where we have small circles filled with data points adjacent to a very large circle filled with data points. We might naturally think there are 3 clusters corresponding to the 3 circles.



In the K-Means objective, the clustering where all the data points in a large circle are assigned to the same cluster has relatively higher or worse objective value, and the clustering where the data points in the large circle are closest to the small circles and are assigned to the outer clusters that have a relatively lower or better objective value.

This is because the data points at the edges of the large circle are relatively far from the cluster center of the large circle, but they are relatively close to the cluster centers in the small circles.

Now the question is, How can we really separate out the three circles as clusters?

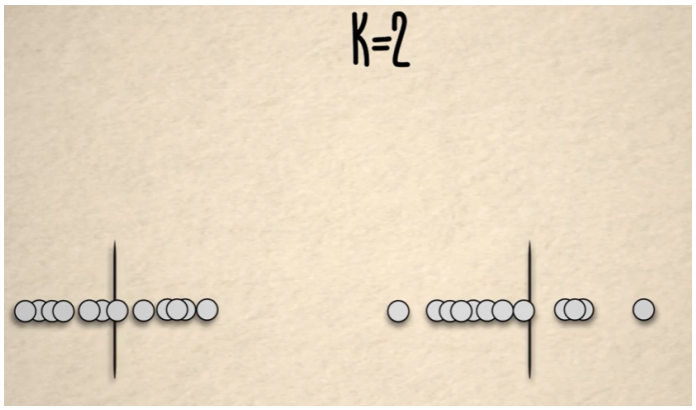
One option is to use a model that specifically encodes clusters of different sizes.
For instance, previously we mentioned Gaussian Mixture Models. When the mixture components are allowed to have different covariances, then these models can capture the kind of clusters that we are trying to find here.

One useful algorithm for this type of model is called the **Expectation Maximization or EM** algorithm.

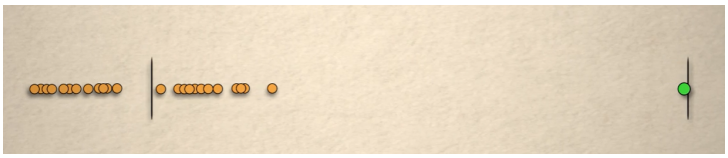
Another issue here is, we might encounter outliers in data. Outliers are data points that are relatively far away from most of the data points in the dataset..

As K-Means Clustering uses Euclidean Distance, that's why K-Means Clustering is typically very sensitive to outliers.

Consider an example of one dimensional dataset:
If we run the K-Means algorithm with K=2, we find that there are two clusters as we expect.



But if we add a single point very far away from the data, then we get one cluster with all the main dataset and the second cluster contains only the outlier point. This cluster does not really capture the two clusters that really exist in the data.



So, what can we do in such a situation?
One option is to use an alternative measure of distance, that is less sensitive to outliers.

Previously to calculate the Squared Euclidean Distance, we added up the squared difference between every pair of features shared by the two data points. This is also called the L2 distance.

$$dis(x_3, x_{17}) = \sum_{d=1}^{D} (x_{3,d} - x_{17,d})^2$$

L^2 DISTANCE

An alternative distance is the **L1 distance**, where we say the distance is the sum of the **absolute values** of the differences between every pair of features shared by two data points.

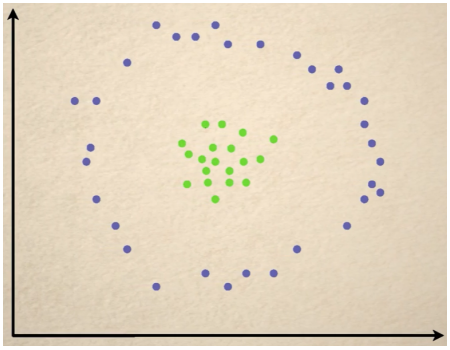
$$dis(x_3, x_{17}) = \sum_{d=1}^{D} |x_{3,d} - x_{17,d}|$$

L^1 DISTANCE

Pretty much everything we already did can be adapted to this case. In particular, we can also derive an iterative algorithm like K-Means, for this problem. The algorithm is called **K-Medoids**, since the optimal cluster centers are now medoids instead of means. The medoid in one dimension is the median.

Finally we might be interested in clusters of different shapes, rather than only spherical clusters.

Consider the example dataset, here we can see two identifiable groups or clusters of points. But one group is situated inside the other. If we apply K-Means with



two centers, then we would just split the data into two sides of some essentially straight lines.

We need something different to capture the two clusters that we can see here.

- For instance, we might define a radial notion of similarity.
- We can also consider splitting our data into polar coordinates or using kernel methods.
- Another alternative is to use Agglomerative Clustering.

Another potential issue with applying K-Means clustering is that it requires continuous numerical features. If we are dealing with text or binary YES/NO features that are not continuous numerical, then we might consider alternative notions of similarity.

Another option here could be to transform our data into a form, which is more amenable to K-Means Clustering.

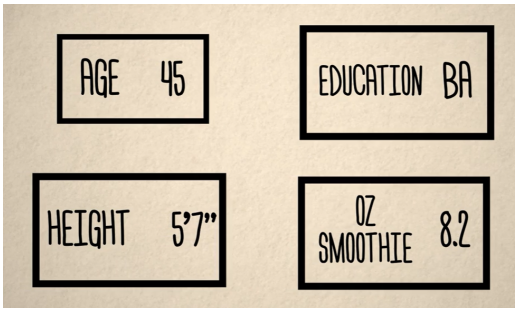
Beyond K-Means: Data and Preprocessing

We have discussed different ways of troubleshooting the K-Means algorithm.

Now we will take a closer look at our **Data**.
In the archaeology example that we have discussed before, there we imagined our example dataset as the locations of the archaeology artifacts.In particular, for each data point we recorded a distance in east from a marker and the distance in north from the same marker.
But there are a lot of other types of data out there that we might collect. So, how could we know when the data is ready to run K-Means?
Before running K-Means on our data there are a bunch of questions that we should ask ourselves which are:

1. **Is our data featurized?**

Example: Suppose we have made a fitness app and we have collected a lot of data about our users. Like their age, height, education and post frequency on the app forum.
But K-Means needs a vector of data. So, the first thing that we have to do is to turn all of this information into an orderly vector. Each entry in the vector will have a particular meaning. Each of these entries is called a feature.



We can imagine this as a big table or matrix that collects these features across all of the users of the app. Once we organize our data like this, then it becomes featurized.

FEATURIZED	AGE	HEIGHT	EDUCATION	OZ SMOOTHIE
PERSON 1	45	5'7"	BA	8.2
PERSON 2				
...				
PERSON N				

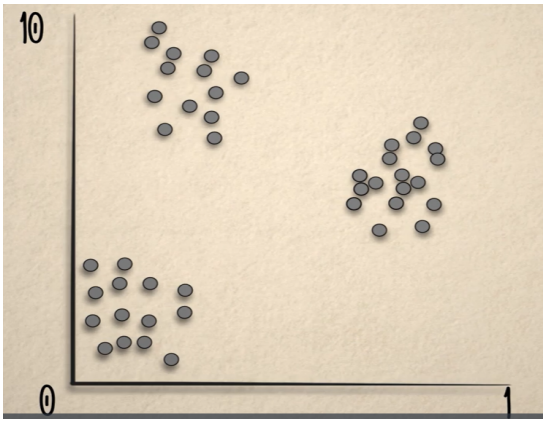
2. We check in the matrix is each of our features a continuous number?

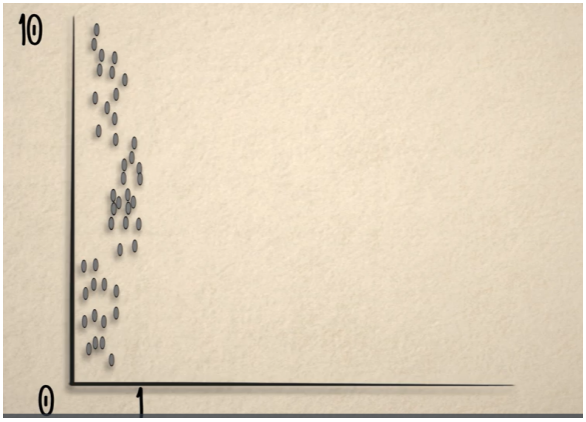
Now K-Means also needs its features to be valued as continuous numbers. So, the next thing we check in the matrix is each of our features a continuous number?
In our example, age is in years, so it should not be changed. We can convert heights into inches. We can express education as years of education(for example, we write 16 in the education column for Bachelors Degree) and smoothie quantity type also need not to be changed.

	AGE	HEIGHT	EDUCATION	OZ SMOOTHIE
PERSON 1	45	67	16	8.2
PERSON 2				
...				
PERSON N				

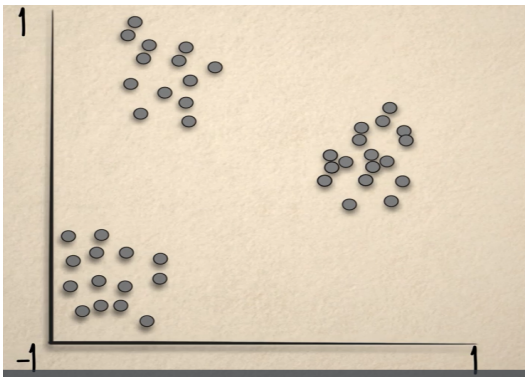
3. Are the numbers in the feature matrix commensurate?

As K-Means assumes that we have spherical clusters. So, the clusters have basically the same width in all directions. Consider the figure in right for a given dataset. Here the Y-axis ranges from 0 to 10 and X axis ranges from 0 to 1. So, if we plot the axes on the same scale, then the data look like the below figure, to the K-Means algorithm.



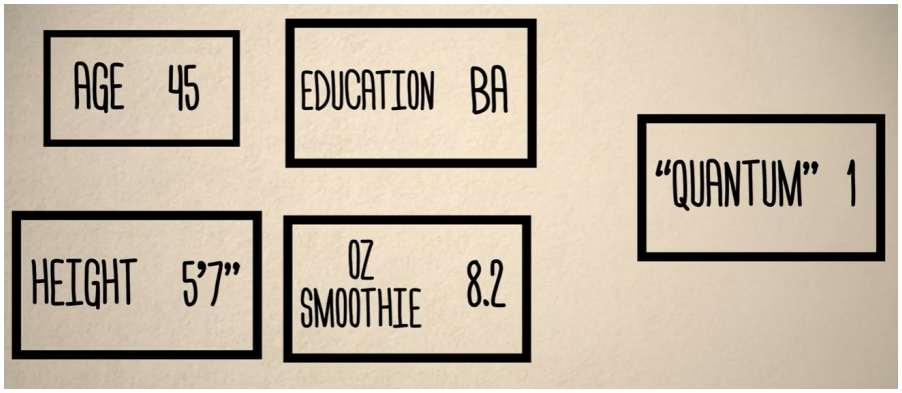


So we want to put all the data on the same scale and for this typically a standardization or a normalization is used. For instance all data on each feature can be separately normalized to lie between -1 and 1.
A common alternative is to standardize the data in each feature to have a standard deviation of 1.

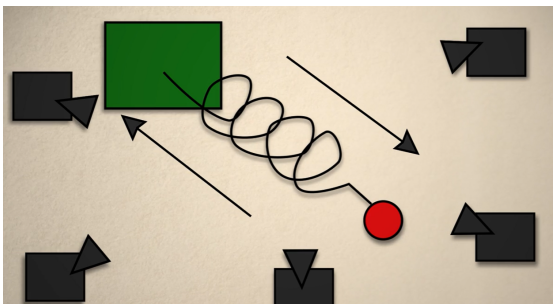


4. Are there too many features?

For instance, if we have a fitness app, then we might want to find different groups of individuals who have similar health and fitness needs. But suppose we have added a bunch of useless features, like for every word in the English dictionary, we add a feature for how many times this particular person uses that word in the forums.



A lot of times we have columns in our data that are not entirely useful to the purpose at hand. When we know which features are not useful because of some domain knowledge, then we want to get rid of them beforehand. But sometimes we don't know which are the useful and useless features. In that case, there are some algorithms that let us reduce the dimension of our feature space.
So we basically reduce the number of features to get the most important features out.



The most widely used algorithm for this is, **Principal Component Analysis(PCA).**

Example:

Suppose, we have a spring and a ball is moving back and forth on that spring. And we record this motion with a bunch of different cameras (Suppose we have 5 cameras). But the motion of the spring is just one dimensional. So, here we have 4 extra features. PCA will help us pick out the feature that really matters, even though it doesn't necessarily correspond to just one camera.

So, it's always a good idea to run PCA on our data before running K-Means. So, PCA is a preprocessing step for K-Means.

5. Can we use domain knowledge?

So, we need to check if there are any domain specific reasons to change the features. If we already know, it's best to get rid of any feature that we know in advance which won't be helpful for our final problem.

Any domain specific feature change should be done before running PCA or K-Means or any other algorithm. It's also best to change any features that might not satisfy the assumptions of K-Means. For instance, we discussed previously that there might be transformations of our features to something that better fits the assumption of K-Means.

It's always good to automate everything using Machine Learning and Statistics. But if there is any domain specific knowledge that we can use specific to our problem area then sometimes it can make a big difference in result.

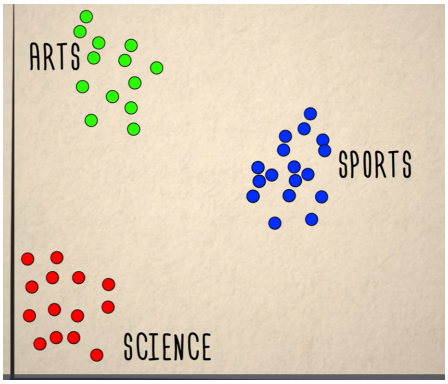
So, in summary, in this article we have discussed preprocessing the dataset, so we can run the K-Means algorithm with the best possible result.

Assumption of K-Means Clustering - Part 1

We already examined some of the assumptions in K-Means clustering. Also we discussed what to do if we want to cluster the data but our application or our data don't quite fit those assumptions.

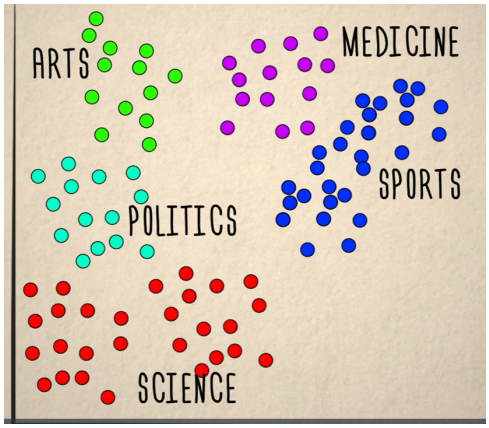
We know K-Means Clustering is just one particular type of fixed K clustering, that is the value of K is fixed . Here we also assume the number of clusters is finite.
Even if we don't know the number of clusters in advance, we often still assume that the number of clusters in our data is some fixed and finite number.

In this article, we will discuss why that assumption might not always be right.
Often bigdata doesn't just mean that we have a single very large data set. We might have **Streaming Data** (where new data is being added to our dataset all the time) .



For instance, English language wikipedia alone averages around 800 new articles per day right now.
If we build a new fitness app, maybe it will not only get new users all the time, but users will interact with each other everyday.

So, what changes when we have Streaming Data?
In the case of wikipedia, no matter how many articles we have read in the past, there are always new topics to read about.
We can think of this as the "Wikipedia Phenomenon".



Suppose we have 100 articles from wikipedia. We could cluster those articles and find some topics to group articles together.
But if we have thousands of wikipedia articles, then we expect to find new topics(shown in figure) that we haven't seen in the first 100 articles.

As we read more and more wikipedia articles, getting to millions and billions of articles, we expect to keep finding new topics.
In this case it might be okay to run clustering with some fixed number of clusters for a hundred articles or any fixed number of articles.
But the number of clusters grows with the size of the data set. We can also say that we want the complexity of our model to be able to grow with the size of the data.

This happens in a lot of other types of data as well. For example,

1. Humans have been discovering species for a very long time and yet at any time if we do a search on google news, we will surely find that there are articles about all kinds of new species that were discovered in the past week. So, we are still discovering new species all the time.

Rates of species description [edit]

According to the RetroSOS report,[13] the following numbers of species have been described each year since 2000.

Year	Total number of species descriptions	New insect species described
2000	17,045	8,241
2001	17,003	7,775
2002	16,990	8,723
2003	17,357	8,844
2004	17,381	9,127
2005	16,424	8,485
2006	17,659	8,994
2007	18,689	9,651
2008	18,225	8,794
2009	19,232	9,738

2. Take the example of our fitness app that we have been referring to for some time. We can ask people about their exercise routines. And no matter how many people we ask, we might expect that we will keep discovering new and unusual exercises that we haven't seen among the previous users.

3. In genetics, we might expect to find some new and unknown ancestral populations as we study more individual's DNA. Or suppose we look at newborns in a hospital and as we study more and more about newborns, we might expect to find more new and unique health issues among those newborns, and we want to be prepared for the new health issues, so that we can treat the newborns better.

4. Consider a social network, as more and more people join the social network, we expect to see new friend groups and interests represented in the network.

In all these cases, we don't want a fixed number of cluster K, for clustering. Here we want K to be able to grow as the size of the data grows.

One solution to this problem is provided by **Non-Parametric Bayesian Methods**.
Here, the 'Non-Parametric' part means many parameters or infinitely many parameters.

These methods let the number of instantiated parameters grow with the size of the data. But these methods are more complex than K-Means Clustering. But some very recent work on **MAD-BAYES** shows how to turn Non-Parametric Bayesian methods into K-Means like problems and algorithms for clustering in particular and Unsupervised learning in general.

An additional read on Non-Parametric Bayesian Methods can be found at Here.

Assumption of K-Means Clustering- Part 2

In the last article, we talked about the case where we can't assume there is a fixed number of clusters in our dataset and also, the complexity of our model to grow with the size of the data.

In this article, we will discuss the cases where clustering doesn't quite represent the hidden patterns that we want to find in our data.

Let's take an example:

Suppose we have a website where people post pictures of their pets.
Now we want to cluster these images. We would like our cluster algorithm to put all of the images of cats in one cluster, all dogs images in another cluster and all the images of lizards in another cluster.Like the figure in below

Figure: 2

One way we could represent this clustering is in a table or a matrix where each row of the table corresponds to one of our images. We will put a 1 in the column that corresponds to the cluster for an image. In the figure, picture 1,2 and 7

CLUSTERS

	CAT	DOG	LIZARD
	1	0	0
	1	0	0
	0	1	0
	0	0	1
	0	1	0
	0	0	0
	1	0	0

belong to the same cluster. Picture 3 and 5 belong to the same cluster and picture 4 is in its own cluster.

But one issue here is,

What if someone posts a picture of both a cat and dog in a single picture.

Also, if someone posts a picture, that doesn't have any animals?

The problem with clustering is that each data point has to belong to only one cluster. There may be uncertainty about which group that is, for some data points.

But ultimately we believe the ground truth, that is there is exactly one true group that the data points belong to.

Now again consider our website and people's pet pictures. There we might want our data points(the pictures) to belong to multiple groups simultaneously or no groups or just one group.
Any of the above options will be allowed.

FEATURES

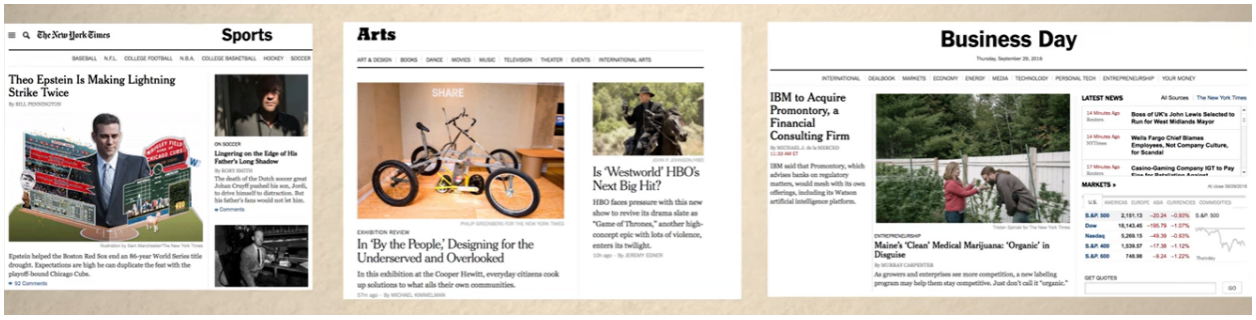
	CAT	DOG	LIZARD
	1	0	0
	1	1	0
	0	0	0

Here, we call the groups **features** instead of clusters and the underlying structure is a **Feature Allocation** instead of a clustering. This **"Feature"** is different from what we have discussed in previous articles.

A similar idea is to say that, the data points exhibit **Mixed Membership**. They can belong to multiple groups at the same time unlike clustering. Sometime clustering is called the **Mixture Model** since each data point comes from one of a bunch of different groups. The bunch of different groups forms the mixture.

Then the case where each data point can belong to multiple groups at the same time is called the **Admixture Model**. Here all the three terms **"Feature Allocation", "Mixed Membership", "Admixture"** capture the idea that **data points** can belong to **multiple groups simultaneously**.

We see this type of example in lots of different data. Suppose we are analyzing a corpus of documents and we want to find the topics or themes that occur, say we have looked at all of the documents and past issues of the New York Times. So, some natural topics that we might find include sports, arts and economics.



Then suppose we read a review of the movie “Moneyball” based on the book by Michael Lewis. Here, the topic can be classified as arts because it’s a movie review. The review is also about sports because the movie is about baseball players. And the review is also about economics, because the movie is about using analytics and statistics to trade players and choose the best players.

So, if we think of a document as a data point, then here we want our data points to be able to belong to multiple groups.

Similarly if we study genomics, we often see that an individual’s DNA may be composed of parts from a number of different ancestral groups.

If we study social networks, we may see that an individual’s interactions on a social network represent various different personal identities such as work identity, family identity, and a social identity separate from the first two identities.
There are numbers of different models and algorithms that let us go beyond clustering and capture this kind of mixed membership structure in data.
The most popular among this kind of algorithm is **Latent Dirichlet Allocation(LDA)** .

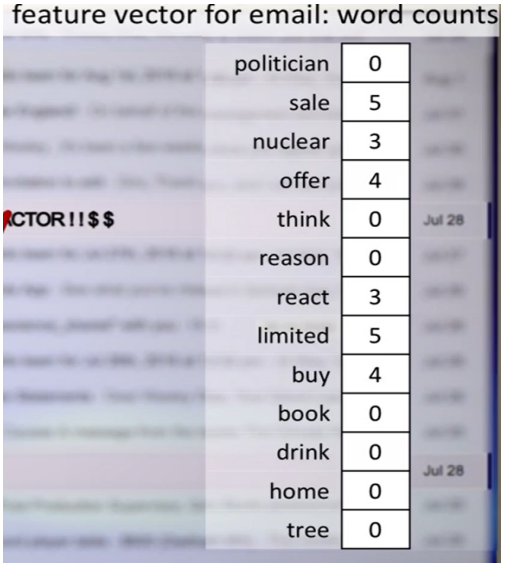
LDA was originally designed for text data and it became popular due to the rise of massive amounts of text data available online. But it could be applied much more widely to other types of categorical data including genetics data.

The other K-Means algorithm for the Feature Allocation problem has been derived using MAD-BAYES. For example, DP-Means and BP-Means algorithms have been used to study tumor hydrogenation. In many tumors multiple different types of cancers are present and it’s important to identify all of the different types of cancer, to design effective treatments.

K-Means Clustering- Recap

We have already discussed many interesting and useful methods to find grouping structure in the data.
For example: The K-Means Algorithm.

Let's recap some of the things that we have discussed so far:
For clustering and many other tasks, our data comes in the form of data points. Each data point is a feature vector. We can think of a feature vector as a string of numbers and each number describes a feature.



For example:
In the email (Spam or Not Spam) example that we have discussed at the beginning, each email is a data point. It is described by the words it contains. So, we have a huge vector and each entry in this vector corresponds to a word. The number for each word means how often this word occurs in the email. With such representation we can use K-Means.

But we don't get such vectors that easily all the time. Sometimes the feature vector can contain too much useless information or noise. We may not be able to construct such vectors at all. In such cases, we may be able to construct new features from whatever we had.
We will discuss how to do it later.

Now look at some examples:

Data can contain a lot of measurements i.e each data point is a huge feature vector with many entries.

1. Suppose a person is a data point and the description is the variation in the person's genome. This can be huge.
2. Think of a collection of images that are face portraits. Each image is a data point and is described by a few hundred thousand pixels. But, some pixels are more meaningful than others.

For example, the upper right corner of face images, that part is most likely going to be not so relevant. It's a constant value or varies randomly across images since it's background. So there is not a whole lot of information.

The important information is how the images deviate in a major way from some average image. For example, there will be variation in hair, glasses, beard, etc.

These major access variations are what captures that data in a meaningful way.
And maybe there are actually much fewer axes and pixels. As some groups of pixels tend to vary together, because they belong to some subregion like eyebrows and chin. These few trends can help us compress the data.

3. Similarly user ratings can be much more easily understood with patterns. My rating for a movie can be understood more easily if you know which genre of movies I like, for example: comedy or adventure movies.

Finding and recognizing such patterns can help us to reduce the complexity in the data and also to reduce noise and bring out important trends and also to compress it.
We will discuss some important methods for finding such patterns later on.

Sometimes we do not even have a feature vector. Let's take an example from article 1.2, which is about the social network of monks that Dr. Simpson recorded back in the 60's.
The monks are our data points and we want to find groups of friends among them.

Can we use K-Means to find groups here?
The answer is, it's going to be hard. As we don't have any feature vectors. All we know here is who talks to whom.

So, we have data points as the monks and we have a relationship of who talks to whom.
This will give us a graph.
We can draw all the monks on paper and draw lines between those who talk. The structure of points and lines is a graph. The lines are called edges.

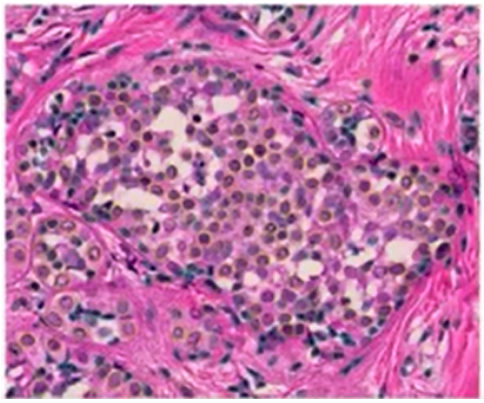
This graph tells us a whole lot about the patterns in our data. Groups of friends will have lots of edges between them and between such groups there are not many edges.

We will discuss how a little bit of math will magically find us the groups in the graph.
For all the problems of high level which are mentioned here, the same approach will work. We are going to create new features that represent our data points. These new features will reveal a lot about the hidden structures in the data. The features are constructed by looking at the entire data.

Introduction to Principal Component Analysis

In this lecture, we will start talking about the process after you have collected large-scale datasets - what can you do with them, and how do you even start thinking about data analysis?
Here, we will be presenting only one particular application, but you will also get an idea about how this relates to different kinds of industries as well, so that you can relate it to the applications that you care about in your work.

Let's take this example from the healthcare domain.
Assume we have a very large scale and a lot of microscopy images. The image here shows a microscopy image where you have many different cells, and we may want to identify in this image which cells might be abnormal cells.



So the question here is,
How can we visualize this dataset to find clusters or abnormal cells?
Here we have,
Input : x_1 ,..., $x_n \in R^D$
Output: y_1 ,..., $y_n \in R^d$, where d << D

So these are the microscopy images of human tissue slices that we are dealing with.
And the first thing we can do is crop out these cells (n cells) and summarize each cell by 100 different texture features (i.e, D = 100).
So every cell is represented as a high dimensional vector based on how many pixels you have and how many color channels you have.

Say every cell has like 10,000 pixels, then that means every cell here is represented as a point in 10,000-dimensional space. Now we humans are very bad at looking at very high dimensional spaces, we can look at maybe two or three-dimensional spaces. So if we need to get a first impression of what the data set actually includes, then we would like to represent these many many cells (where say we have 100,000 cells and every cell lives in a 10,000-dimensional space) into a **lower dimension**.

For example, we would like to represent it in two or three dimensions and by doing so we may see some clusters. There maybe we could find some groups of cells that are actually clustered together and maybe some of these clusters correspond to abnormal cells, maybe we can get some cancerous cells or maybe some of them might be abnormal in terms of going more towards the cancer state, etc.

Similarly, if you're thinking of other industries, for example, say you have a whole lot of data on your customers that you even have time-series data on who bought what and in terms of their buying patterns. So those could be long time series where you have like 10,000 different variables for every one of the customers. For example let's say you have 100,000 customers so that you have a data set of 100,000 samples and every sample (every customer) lives in 10,000-dimensional space, and we cannot look at 10,000-dimensional spaces. We would like to be able to represent this data in two or three dimensions so that we can actually look at it and see whether we can identify different groups of customers so that we can then maybe look and analyze a bit more carefully about what are these different clusters? What is it that makes these different people clustered together? What is it in their buying behavior that makes them clustered together? And maybe then we can target our treatment, for example, advertising, different kinds of processes that we want to use with different customers, etc. But we're really caring about actually identifying these different clusters. So first step, we just want to be able to visualize this data in two or three dimensions.

So now we will discuss two methods about how we can do this.

Principal Component Analysis (PCA): Projection that spreads data as much as possible.
Stochastic neighbor embedding(t-SNE): Non-linear embedding that tries to keep close-by points close.

Now, PCA is a very simple method, it's very, very fast, and that's the advantage. So it can be applied very quickly, it should always be one of the first things that you should do with your data.

And t-SNE is much slower but you will see the advantages of this method in order to be able to identify clusters.

t-SNE is one of the newer methods to have come out and is used a lot for finding clusters, and PCA is one of the standard methods that has a long history but is very, very fast.

PCA is a linear method, so it's very fast. And t-SNE is a non-linear function. But you will see how much better t-SNE is in general, in finding clusters or identifying them.

The previous version was only called SNE, and the newer version has a t in front and it's called t-SNE. We will also see the advantages and disadvantages of these two methods.

Let's talk about PCA first, because it's a very, very standard method and it's really something that everyone should do as a first thing when they're getting a dataset to just represent and see whether there are outliers in the space and to just find some general patterns in your dataset.

Now, what is the goal of PCA or Principal Component Analysis?
Goal: Dimensionality reduction to a few dimensions.
Intuition: Find the low-dimensional projection with the largest spread.

For example, say you have a dataset of your customers and every customer is a vector in 10,000-dimensional space and you would like to represent each one of your customers in two or three dimensions so that you can look at it.
So there are a couple of different ways of thinking about what PCA does. So first of all, it's a linear method. So it just projects the data from 10,000-dimensional space into whatever dimension you choose. It may be one dimensional, two dimensional, or three dimensional. So, the dimensions should be few so that you can still look at them.

And now the question is, what does it try to achieve when it tries to find such a linear embedding of such a projection?
There are different ways of seeing what it tries to do, and they're all equivalent.
So one way of trying to see what it does is that it actually tries to find the projection so that the data is still as spread out as possible.
Now, why is this important i.e why would you want to capture the directions of the largest variation in your dataset?
So think again about your customer base and think about the fact that maybe you're measuring a few variables. Age group could be one of the variables, and maybe another variable is the country they're living in.

Let us say, you are a company that is based in the US and in your customer pool, basically everyone is in the US. Then for that variable of where these people are living, there is nothing changing in that variable, since everyone is in the US, so this variable really doesn't tell you anything. If you still see differences in your customers in terms of their buying behavior still that variable cannot explain it since it doesn't change at all.

So these variables have very little variation in your data set. They don't matter as they don't tell you a lot about the dataset. So that's the intuition for removing all these variables that have very little variation, and only keeping the ones that actually have differences in the samples that you're looking at.

Also in the microscopy image that we saw previously, we don't have to keep the pixels that are always the same in every one of the cells. We want to concentrate on the pixels that are actually different in different cells because these can tell us something about whether a cell is indicative of being abnormal versus normal.

So that's the intuition for trying to find a projection that preserves the variables or the directions that show the biggest amount of variation.

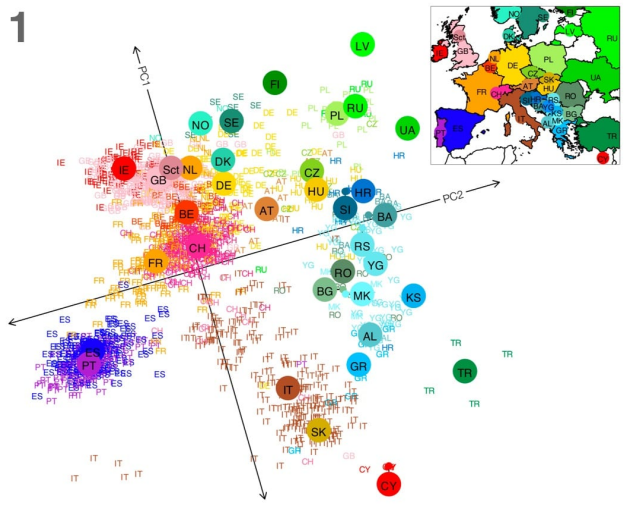
Let's look at an application of PCA :

There are different kinds of companies (for e.g. 23andMe is one such company) that offer to actually sequence your DNA. And so, you can get all the snippets of the values at all of the positions that differ between different people. So there are many sequence structures that are the same between you and me. They justify that we are humans, and they will all be the same for all of us. But, these positions don't matter much. They don't contain any information about our ancestry or about any of our phenotypic values. So what we really want to measure are the positions that actually change between different people. Those are the ones that carry information. So those are the ones that are usually sequenced.

And what they did in the experiment is they took people that got sequenced in different parts of Europe. So, they got a very very long 100,000-dimensional vector for every person of what is the value at that particular location in the DNA. And all they did is do a Principal Component Analysis.

You can see in the figure below the vertical axis **PC1 (Principal Component 1)** and the horizontal axis **PC2 (Principal Component 2).**
So this is just a projection along with the two directions that vary the most.

1



What we see is really quite amazing, where basically every person was mapped to the location they are coming from. The map that they get here represents quite well the map of Europe.

So without putting in any information about where this person comes from, all that was done here is to take DNA from every person, and then a Principal Component Analysis was done, and that outcome is basically the map of Europe. So just PCA, which is something so simple, was able to recreate and actually map every one of the dots back to where they're coming from just by choosing two directions in the data set which maximize the spread.

Principal Component Analysis - Covariance and Eigenvectors

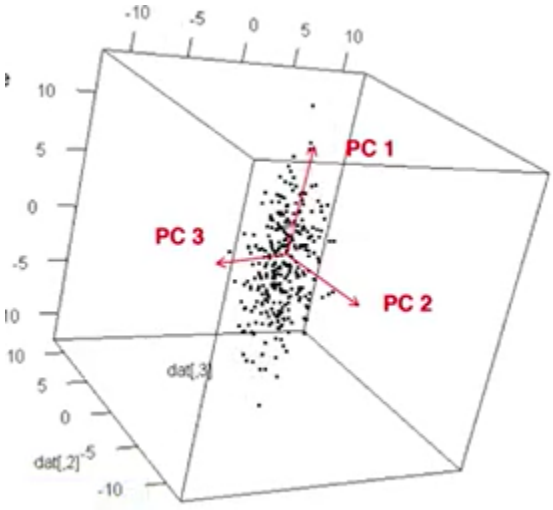
In this lecture, we will talk a bit further about what Principal Component Analysis (PCA) actually does.

So, there are different definitions of it and we have already discussed the intuition behind it in the last lecture.

Definition 1: **Maximize projection variance**

Start with centered data $X \in R^{n * p}$

- PC 1 is the direction of largest variance.
- PC 2 is
  - perpendicular to PC 1
  - again largest variance
- PC 3 is
  - perpendicular to PC 1, PC 2
  - again largest variance
- etc.

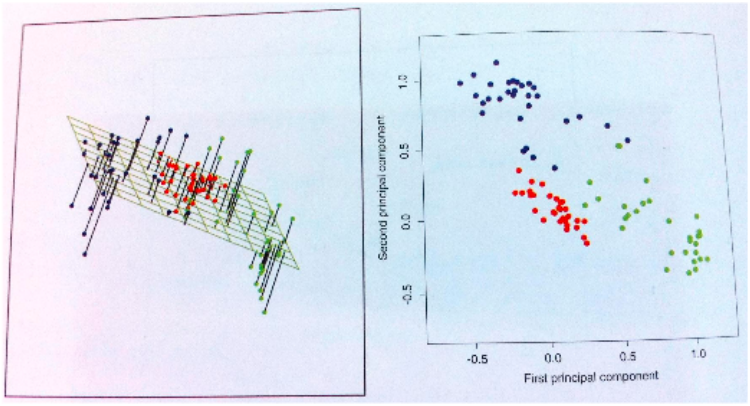


So first we have our samples. And they live in some high-dimensional space. We only showed three dimensions here in the image, but they actually live in high dimensional space. And what we are trying to find are the directions that vary the most. These directions that vary the most are the only ones we are going to keep, like what we saw in the previous lecture.

For example, if we want to get a two dimensional representation, we just keep the two directions that vary the most. The intuition is quite clear, we don't need to keep the various directions that don't vary because they don't really contain much information about what makes them different from each other and so that's the first way of seeing what PCA does.

Definition 2: Minimize projections residuals

- PC 1: Straight line with the smallest orthogonal distance to all points
- PC 1 & PC 2: Plane with the smallest orthogonal distance to all points.
- etc.



So, we will discuss the second way of seeing what PCA does. In fact, both the previous method and this method are quite the same, but intuitively they are slightly different. So, either we can keep the directions that vary the most, but actually, this is equivalent to removing the directions that carry the least signal.

So if we are trying to remove the ones that have the least signal it means that whatever the signal we are using (for example, we can see points in a three-dimensional plane in the above image) we want to find the projection of that, and eventually, all of these signals are going to be lost if we are projecting them all out into this two-dimensional space, and this signal we lost is the smallest one possible.
So that's what we mean by minimizing the projection residuals. So residuals are whatever is left out that we weren't able to capture by this projection and that signal that is being lost needs to be as small as possible.

In the previous method, we kept the biggest signal possible that we could keep when we were going from a 10,000-dimensional space to a two-dimensional space.
So both of these methods are very intuitive ways of two things that we may want to do but in fact, they are quite the same. So, we can try to find the projection that keeps the

directions of the largest variation, or we can try to find a projection that minimizes the amount of signal lost, and we can show that these two things are exactly the same,

Now the question is how do we even compute it i.e how do we find directions that maximize the amount of variation captured or find the directions that lets us lose the least amount of signal?
How do we actually do this? How does a computer do it?
It's really quite exciting that a computer can do it by just doing something that is known as a **Spectral decomposition**.

- Covariance matrix (or correlation matrix) $R = \frac{1}{n}X^TX$ is symmetric and positive semidefinite.
- **Spectral Decomposition Theorem:** Every real symmetric matrix R can be decomposed as

  $$R = V \wedge V^T$$

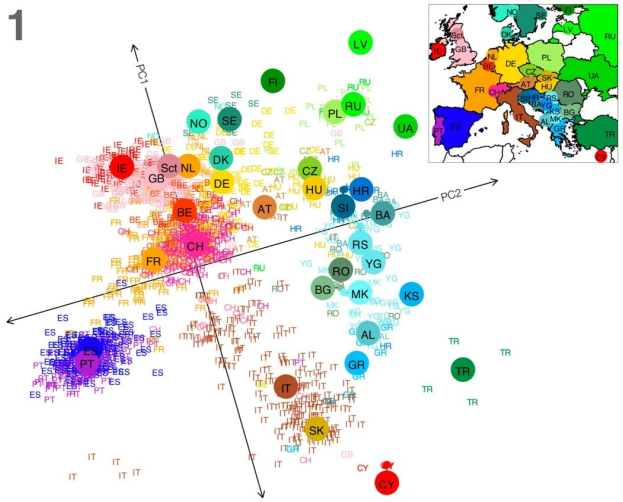
  Where $\wedge$ is diagonal and V is orthogonal.
- Columns of V (= Eigenvectors of R) are the PCs.
- Diagonal entries of $\wedge$ (= Eigenvectors of R) are variances along PCs.

So what it actually does is to take the covariance matrix of the data or the correlation matrix which says how are two different variables i.e buying behavior of one person and buying behavior of the other person, correlated with each other?

And in this correlation matrix, all we have to do is to just find the Eigenvectors corresponding to the largest Eigenvalues, and the computer can do this very easily.
In fact, a lot of what we do mathematically, when we're doing things with data analysis is finding these Eigenvectors and Eigenvalues, which is one of the most standard computations that we can do with a matrix. So it's just finding Eigenvectors and Eigenvalues.

In fact, we only have to find the directions here i.e the direction of the largest spreads, or the directions that minimize the amount of signal loss that corresponds to the Eigenvector corresponding to the largest Eigenvalue. So if we want to represent it in one dimension then we are done. But if we want to go to two dimensions, then we take the next direction i.e take the Eigenvector corresponding to the second largest

Eigenvalue. Similarly, if we want to represent data in three dimensions, we also can take the Eigenvector corresponding to the third-largest Eigenvalue. So, we just keep as many of these Eigenvectors as we want in our representation to be, and often we would only go up to two dimensions because we would like to find the representation in two dimensions.



That's exactly what was done in this particular example. So we just kept the Eigenvector corresponding to the largest Eigenvalue, that's PC1. And then the Eigenvector corresponding to the second largest Eigenvalue, that's PC2, and that's it i.e just representing the data in this space corresponding to these two Eigenvectors is exactly what was done to get this plot here.

So this is a very standard mathematical tool that people use. So when we have a matrix we do this Eigenvector decomposition and we just keep the largest Eigenvectors and we represent our data in that space and this is exactly what PCA does. We can see here that it's been very effective in this particular application.

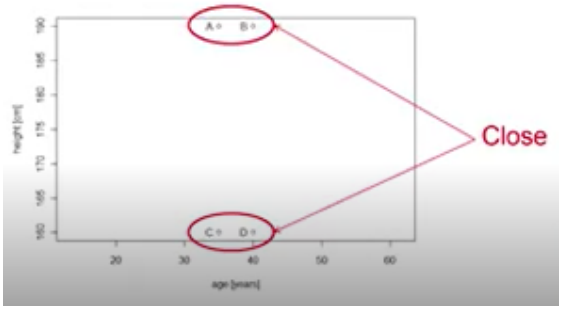
This course is about also giving you intuition for things that you have to be careful about. You also need to be careful about something here when we're talking about Principal Component Analysis.

So let's say we have here a very simple dataset, where we have people (for example we have only 4 people here), and what we're collecting is data on age and data on height.

Person	Age (years)	Height (cm)
A	35	190
B	40	190
C	35	160
D	40	160

So, of course, the age can be measured in years, in days, and similarly, in months. And for heights, here we have measured it in centimeters, but you can also measure it in feet, etc. So let's think through whether it actually matters which kind of units we are measuring things in when we're doing Principal Component Analysis.

It's really important to think through this because you don't really want that your picture changes when you're just changing the units. For example, when we are going in height from cm to feet, we would like the picture to remain the same because the data is the same. It's just that we changed the units.



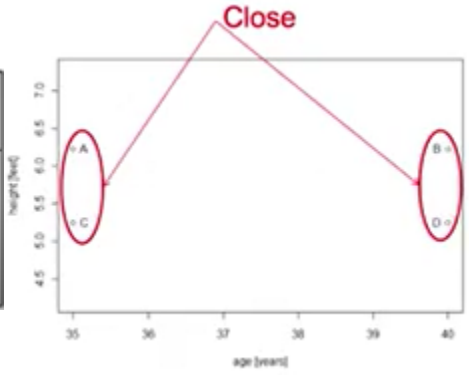
But think about what PCA does. It tries to find the directions of the largest spread. Well, if we change something and just multiply our numbers by 1000 or 100,000 in one direction, **then suddenly this direction becomes the direction of the largest amount of variability just by how we are scaling the data.**

So it's a problem for PCA that in its standard form, it actually does depend on the units that we choose.

So for example, here in this little data set, if we measure height in cm (centimeters), the direction of the largest spread is in a direction which is along the height.

However, if you change it to feet, the direction of the largest spread is in a direction here which is along the age.

Person	Age (years)	Height (feet)
A	35	6.232
B	40	6.232
C	35	5.248
D	40	5.248



So we see that in PCA, the picture that we're getting of our first principal component is in fact different depending on whether we're measuring height in feet or whether we're measuring height in centimeters.

It is just because centimeters have a larger spread as we are multiplying it with some value.

So that's really problematic in terms of PCA and when you have datasets which have very different units. For example, here we have age and we have height. They're very different units, since one is measured in years and one is measured in feet. So it's not clear how to normalize them against each other.

Then what we should do actually is instead of the covariance matrix, we should always use the **correlation matrix** when we have different units in different entries.

It's important to use the correlation matrix because the correlation matrix normalizes every one of the directions of variables so that all of them carry the same weight.
So here we have age carrying the same weight as height. We're normalizing all these different variables against each other.

This is really important. Because if we change the units it can just completely change our first principal components. So this is something that we really have to take into account when we're doing PCA.

If we have the same kinds of units, it doesn't matter as much. If all of the variables are measured in feet then the differences actually mean something. So then we can use the covariance matrix. But if we're measuring many different types of units in the matrix-like age, height, weight, etc. and these are all units that we have in our data set. Then it is really important to use the correlation matrix. Because depending on how we scale things, our plots are going to become very different.

So that was Principal Component Analysis.
Next, we'll talk about a nonlinear method, and then see how these two methods actually compare against each other.

t-distributed Stochastic Neighbor Embedding (t-SNE)

We have already discussed a linear method namely PCA. Now we will discuss a non-linear method and will see what are the advantages of non-linear methods versus linear methods for visualizing data. And here we will see a newer method that really often performs very well for identifying clusters in our data set, which is known as t-SNE or t-distributed Stochastic Neighbor Embedding.

t-distributed Stochastic Neighbor Embedding (t-SNE)

- Probabilistic approach to place samples from high-dimensional space into low-dimensional space so as to preserve the identity of neighbors.
- Find an embedding so that original high-dimensional sample distribution is approximated well by the resulting low-dimensional sample distribution (t-SNE uses Kullback-Leibler divergence to measure the "distance" between distributions and minimizes this objective function)
- It gives rise to a non-linear embedding where close-by points remain close by and far away points remain far away so that clusters are preserved.

So what is the idea behind all this?
The idea is actually quite interesting. So we have our samples i.e customers that live where we have these high dimensional measurements, say the buying behavior over a long timeframe. So we have our samples in this high-dimensional space and that defines a sample distribution.
We would like to represent that distribution in a low dimensional space, maybe in a two-dimensional space so that we can actually visualize the space. So we can think of all of our customers in this two-dimensional space and that again will define a sample distribution.

The idea behind Stochastic Neighbor Embedding is to view our samples as a sample distribution in a high dimensional space and a distribution in a low dimensional space. And we would like these two distributions to be as similar as possible to each other. Now there are different ways of measuring distances between distributions.

One of them which is used quite a lot in information theory for example is known as Kullback-Leibler divergence. And that's exactly what is used in t-SNE for actually defining a distance between these two distributions -that's what we want to minimize, so that's our objective function.

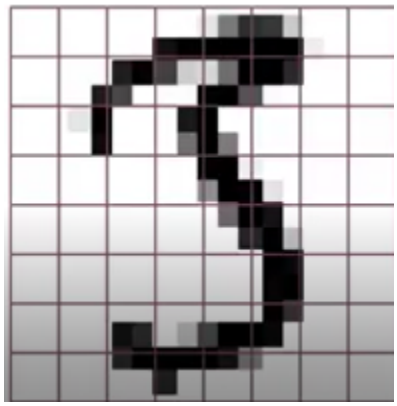
We want these two distributions to look as similar as possible. So what this method does is, it tries to minimize the differences between these two distributions, and that's quite a difficult optimization problem. We won't go into the intricacies of solving this optimization problem, but it's highly non-convex, meaning that it will depend on the initialization that we're giving it. So we should give a good initialization often when we actually use this PCA embedding, which we discussed before. It tries to find the good starting embedding and then from there it optimizes more and more for this difference between the two distributions to become smaller and smaller. So it's a difficult optimization problem and computationally this method is quite heavy. But we'll see some examples showing that it's really often worth doing something like this i.e something nonlinear to find a good embedding where we can really actually start to see clusters in our dataset. So, non-linearity helps us to do it.

We want the two distributions to still look similar to each other meaning where we have a high density of our samples, they should remain high density of samples. So that the clusters that we have remain exactly there i.e we keep the clusters preserved. And clusters that are further away from each other i.e for separated clusters we want them to still be separated in a low dimensional embedding and that's the goal. We preserve clusters and clusters that are separated from each other, we would like to preserve that as well, and that's possible with these nonlinear methods like stochastic neighbor embedding. It's very hard to do this actually with linear embeddings because if we just want to do a projection of high dimensional space, like a 10,000-dimensional space into a two-dimensional space, then that means that in 10,000-dimensional space there's so much more space than in a two-dimensional space. So, the problem of just doing something linear is that, if we're going from something with a lot of space to something with little space then there will be crowding. So usually what happens is that all of the points, even though in this high dimensional space they might still be kind of separated into different clusters, they now all become one cluster. So that's always a difficulty when we just use linear methods. Nonlinear methods like stochastic

neighbor embedding, allow us the added non-linearity to still spread out the data better than just a linear approach. And we'll see what actually happens i.e why and how this method can actually help us quite a bit as compared to PCA which is a linear method.

Let's go through an example. So here's a very simple dataset that is actually very often used for showing or demonstrating different kinds of approaches in machine learning. This is a digit recognition dataset called MNIST, and we have a whole lot of hand-written digits.

- MNIST contains about 1800 images of hand-written digits - 180 for each digit from 0 to 9.
- Each (centered) digit was put in an 8 * 8 grid (i.e D = 64)
- Measure the gray value in each part of the grid, i.e 64 gray values
- Input : x_1 ,..., $x_n \in R^D$ Output: y_1 ,..., $y_n \in R^d$, where d << D

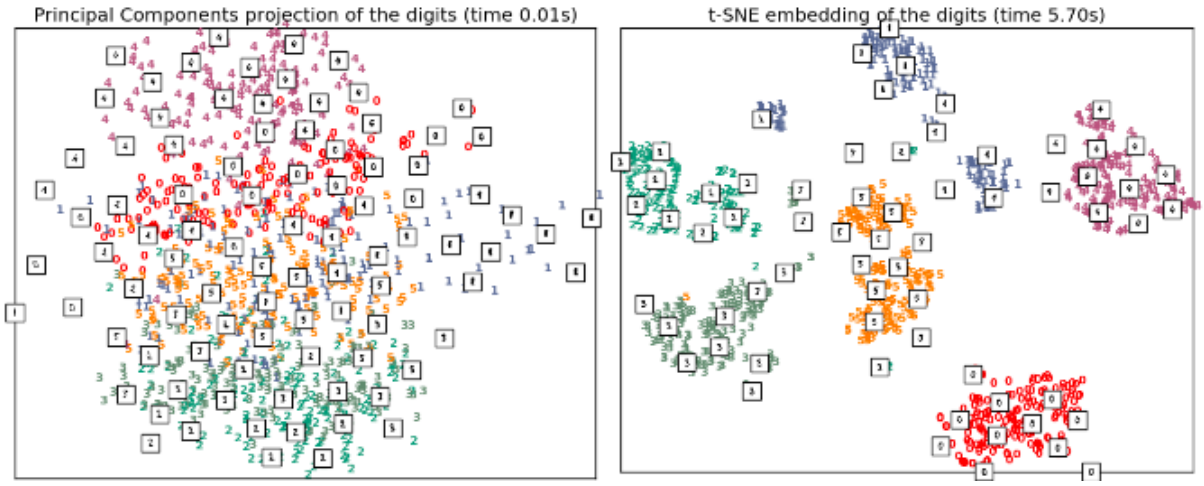


So this is how these hand-written digits look. Every sample here is one hand-written digit. They are summarized into a 64-dimensional vector and it's in grayscale format, so each one of these pixels has a value assigned to it for that kind of entry.

Now we would like to represent all these digits in a low dimensional space say two-dimensional space, so that we can actually look at it and we want to see what are the differences in the representation if we use PCA to represent our data as compared

to this nonlinear method, t-SNE, which just tries to somehow make the two sample distributions in the high dimensional space and low dimensional space as similar as possible.

So let's see what this looks like and look at some of the differences and similarities.



We choose digits from 0-5 to do this.
So this is what we get with PCA, we have the first principal component and similarly the second principal component. And, as we discussed, the first Eigenvector corresponds to the largest Eigenvalue and the second Eigenvector corresponds to the second largest Eigenvalue. This process is super fast (taking only 0.01 second) i.e doing a PCA plot is really really fast.

And the stochastic neighbor embedding is much slower. It took 5.7 seconds to get these embeddings here. Also, this depends on our initialization. So if we would run this again with a different initialization, we might actually get a different kind of picture.
So we really want to run this a couple of times, and then just choose an embedding that leads us to the smallest objective function. The objective function is the Kullback-Leibler divergence that the algorithm will actually output to us as well. So we can just use the results that have the smallest value of this Kullback-Leibler divergence.

So what do we see here as differences?
It's quite clear when we compare the above two pictures against each other.
For PCA we have one big blob, and for t-SNE what we see is actually quite nice, it seems to find meaningful clusters. So without telling the algorithm anything about that we have different numbers, and different digits, we just took the 64-dimensional vectors and all we did is just visualize it in two dimensions. It's actually able to identify that there are differences between these different digits. Here, all the 2's are clustered together, all the 3's are clustered together and similarly, all other digits are clustered together. So that's quite nice.

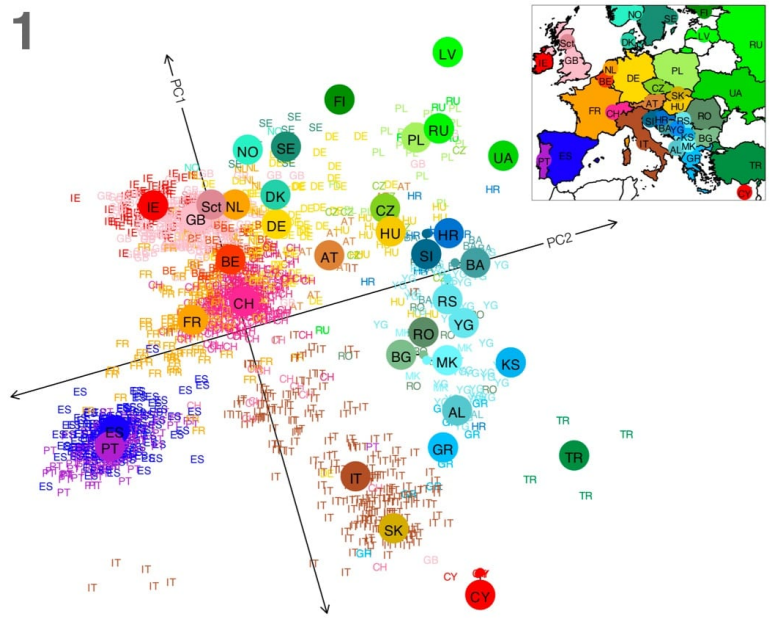
Now we can imagine that if we do this, using a large customer base, that might be for example the whole buying history from these customers. Then a plot i.e a t-SNE visualization like this will be able to group out and separate customers into very different segments or different groups. And then we can go in and actually try to analyze what they actually mean.

Whereas for PCA it's quite difficult to see. We see that along with the first principal component, we have mainly the number 1 that is really spread out. So that is the number with the largest variation because it goes all along with the first principal component, and then the second principal component was kind of able to distinguish between different digits and all the digits are separated along with the second principal component, but it's still not clear where different clusters are. As, because of this linear embedding, we're going from 64-dimensional space to two-dimensional space and there's so much less space that we have to expect that we basically get one blob, although maybe in the high dimensional spaces we did actually have multiple blobs.

So if our goal is just to identify outliers, for example, if there was one number that was completely an outlier. We actually often identify that quite easily in PCA, where we have some samples that are very far out. PCA is a very fast way of doing that, so it is very interesting from that perspective. So definitely, it is advisable to do PCA as a first step so that you can maybe see some patterns that you can just get out very quickly.

But if you want to start seeing different groups in your dataset, then you can go for something like t-SNE which takes longer but it really identifies something interesting in your data. It might also see patterns, and then you can go in and try to figure out why all these customers are grouped together? What is it that makes them similar to each other and different from the customers that you have in other groups?
So we discussed the advantages and disadvantages of these two methods.

Of course, we saw that for some applications, PCA can already work very well, such as the application that we had seen here at the beginning.



Here we also saw that some clusters still come out and are separated from each other, but it was basically put into one big blob in the hand-written digits example.
However the location still made a lot of sense.

But for most other applications, we may need something that is a bit more complex on the computational side, like t-SNE, which takes longer as it solves a more difficult optimization problem. But then it can actually identify some really interesting structure in the data just through a simple visualization.