LECTURE NOTES

Table of Contents

Clustering

- The process of grouping any kind of data based on the similarity in their features, automatically, without human expertise, is called **clustering**. It is a type of **unsupervised learning**.

- Intuitively, clustering is dividing a population into groups such that the points in one group are similar to each other. Each group is called a **cluster**.

- Since there is no ground truth data and nothing to compare with, we decide if a cluster is good or bad if it simply makes some **business sense.**

Some advanced applications of clustering:

➔ Validation of domain knowledge/human opinion.

➔ Search algorithms and recommendation systems.

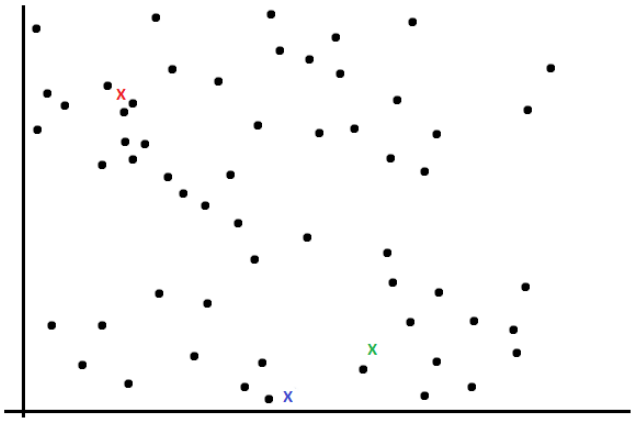
➔ Feature creation.

➔ Clustering with supervised learning to improve model accuracy.

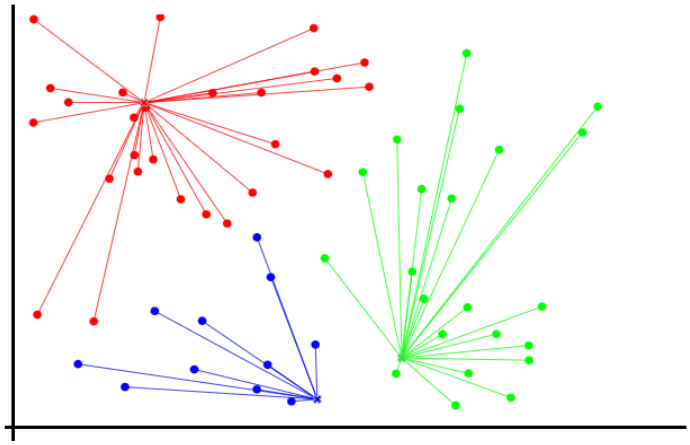
Lloyd's algorithm (K-means algorithm)

- This algorithm is used to cope with the problem of updating the centers.
- It has 5 basic steps:

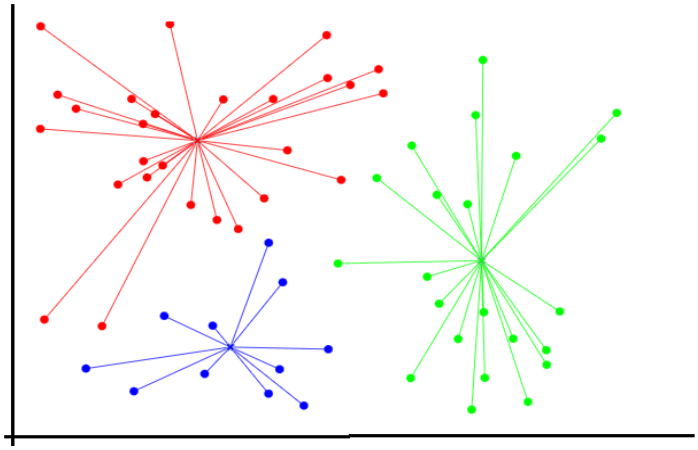
➔ Randomly initialize **k** centers.



➔ Assign points to these centers to get the clusters.



➔ Find the **centroids** of these clusters.



➔ Reassign the points to the centers to create the new clusters.

➔ Repeat until the center of new clusters stops changing their positions.

Within-cluster sum of squares (WCSS)

- The within-cluster sum of squares is a measure of the variability of the data points within each cluster. It is given as:

$$wcss = \sum_{i=1}^{k} \sum_{j=1}^{m_i} (x_{ij} - c_i)^2$$

where $\mathbf{x_{ij}}$ is the $\mathbf{j^{th}}$ point belonging to the $\mathbf{i^{th}}$ cluster and $\mathbf{m_i}$ is the number of points in the $\mathbf{i^{th}}$ cluster.

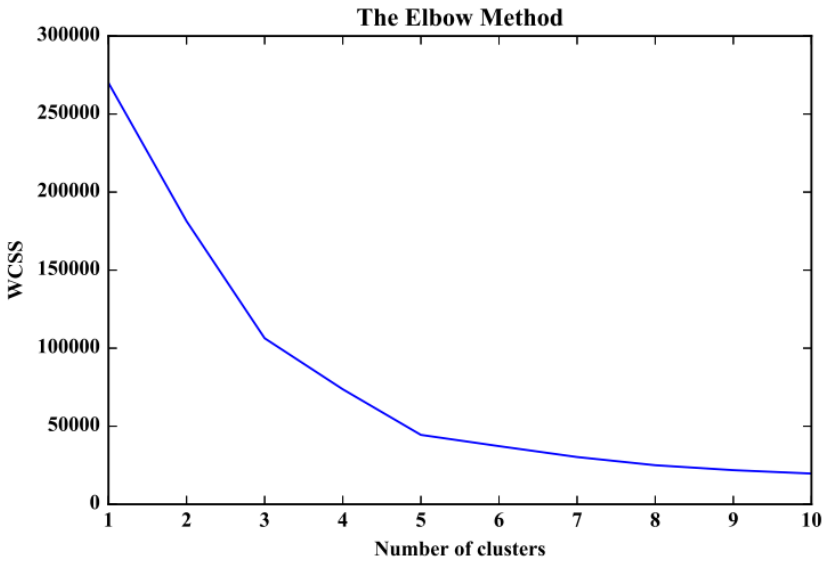
- A **variation** of the above formula can be as follows:

$$wcss = \sum_{i=1}^{k} \sum_{j=1}^{m_i} d(x_{ij}, c_i)$$

where, $\mathbf{d(x_{ij}, c_i)}$ is representing a distance function that is calculating the distance between the point $\mathbf{x_{ij}}$ and the centroid $\mathbf{c_i}$ of the cluster.

Elbow method

- It is a method to determine the optimal number of clusters (**k**) for k-means clustering.
- We perform the k-means clustering for a range of values of **k** and for each iteration, we calculate the value of the WCSS metric.
- When the value of WCSS is plotted against a range of **k** values, we get a plot looking like an elbow.



- We can clearly see that the WCSS value decreases as the number of clusters (k) increases.

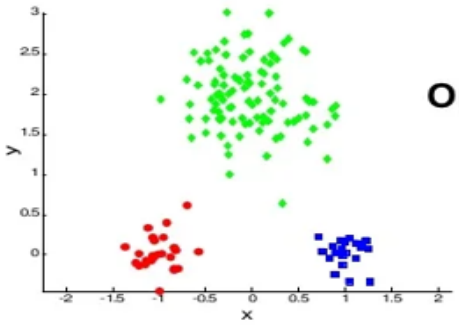
- At some point on the graph, there is a sharp change in the slope (k = 5) after which the change in slope is very small. The k value corresponding to this point is the optimal K value or an **optimal** number of clusters.

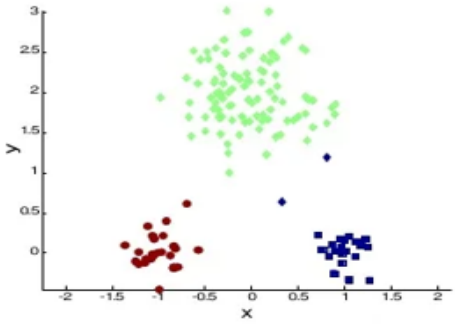
- If we do not get a sharp change in the slope of the elbow plot while using the WCSS metric on the y-axis, we can try using the **Silhouette score** to get significant results or to get confidence in our decision.

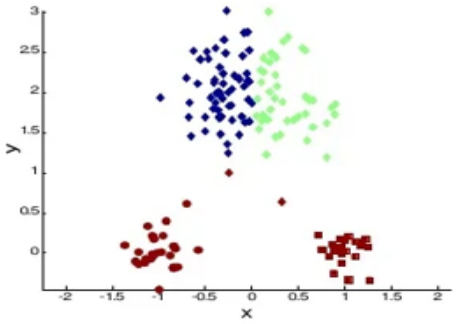
Drawbacks of the K-means

1. We may get bad results sometimes while using the k-means algorithm because of the **random initialization** of the centroids.

   For example




Optimal Clustering


Sub-optimal Clustering

2. The k-means algorithm may not give the best results for data where the clusters are of varying size or density.

3. The number of clusters (k) is not defined.
4. It does not work well with non-globular clusters.

K-means++ algorithm

- To overcome the drawback due to the random initialization of centroids in K-means clustering, we use K-means++. It is smarter to initialize the centroids in order to improve the clustering algorithm.

- The steps involved in the initialization of centroids are:

➔ Select the first centroid randomly from the data points.
➔ Choose the next center as the farthest point from the first center.
➔ The next center would be a data point farthest from both the first and second centers.

- Repeat steps 2 and 3 until **k** centroids have been sampled.

- If there are **outliers** in our data, then instead of choosing them as centroid, we can choose the farthest point as the centroid with a **probability proportional to the distance.** This is the implementation that sklearn follows by default..

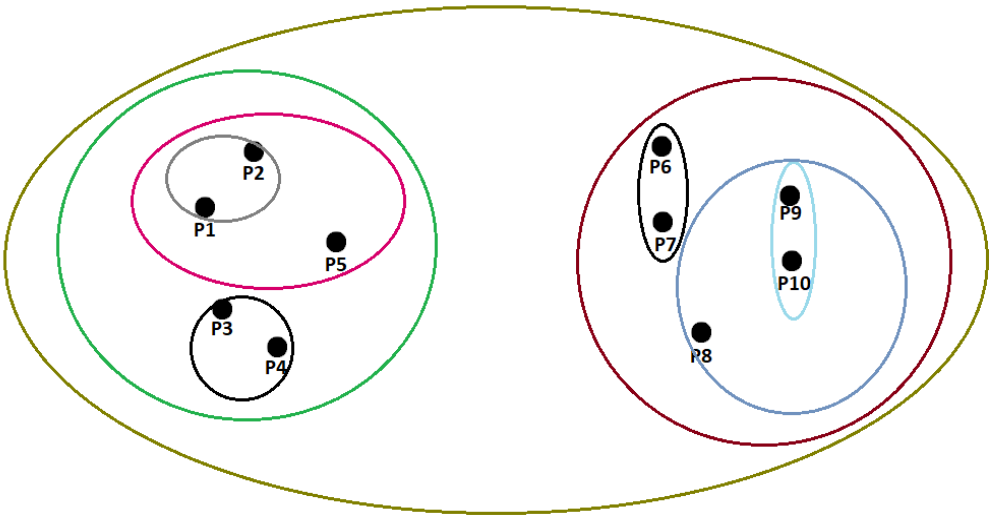
Hierarchical clustering

- It is an unsupervised clustering algorithm in which the hierarchy of clusters is developed in the form of a tree, and this tree-shaped structure is known as the **dendrogram.**

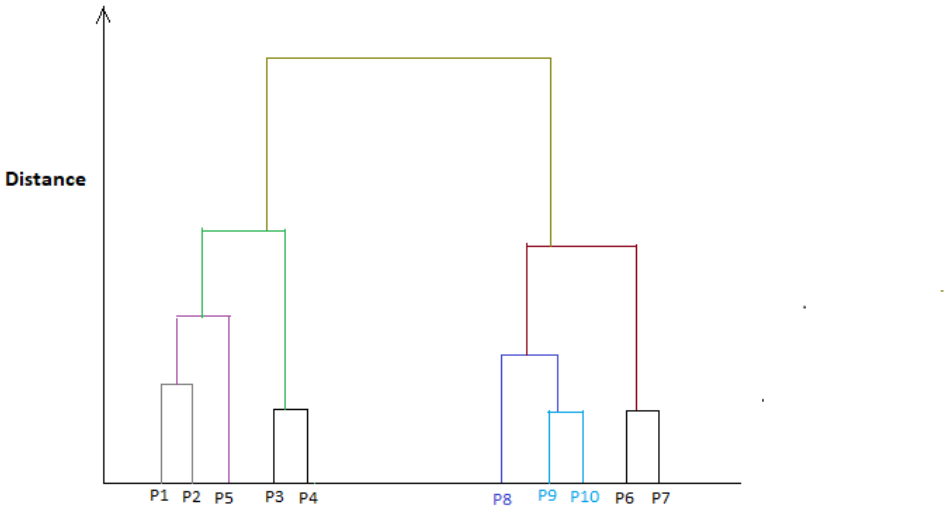
Steps of Hierarchical clustering:

➔ Compute the distance between all the data points. We can choose any distance measure from Euclidean, Manhattan, Hamming Distance, etc
➔ Merge the two points having minimum distance between them.
➔ Treat the sub-clusters as points and repeat till only one cluster remains.
➔ Finally, develop the dendrogram to divide the clusters as per the problem.

For example,



The associated **dendrogram** is:



Distance between two clusters:

- We observe that the distance between two clusters needs to be calculated in order to merge two clusters. Therefore, some linkage needs to be formed between these clusters.

- The common type of linkages that can be formed between the two clusters are:

1. Complete linkage: It is the **maximum** distance between the two farthest points of different clusters.
2. Single linkage: It is the **minimum** distance between the two closest points of different clusters.
3. Centroid linkage: The distance between the centroids of the clusters is calculated.
4. Ward's distance/linkage: The Ward's distance between two clusters c_1 and c_2 is given as:

$$W(c_1 + c_2) = WCSS(c_1 + c_2) - WCSS(c_1) - WCSS(c_2)$$

where WCSS is the **within-cluster sum of squares**.

Advantages of Hierarchical clustering:

➔ There is no need to pre-decide the k value. i.e. no**.** of clusters.
➔ Good for data visualization providing hierarchical relation between the clusters. The hierarchy may actually be present in the real world.

Disadvantages of Hierarchical clustering:

➔ This method of clustering is sensitive to the choice of linkage function.
➔ Hierarchical clustering can only be used for analysis. We cannot do inference using it. If we get a new data point, we need to re-run the entire algorithm with all the existing points and reanalyze.

DBSCAN (Density-based spatial clustering application with noise)

- The fundamental principle is that each point in a cluster must have at least a certain number of neighbors within a given radius.

- The density threshold is defined by two parameters: the radius of the neighborhood (**eps**) and the minimum number of neighbors (**minPts**) within the radius of the neighborhood.

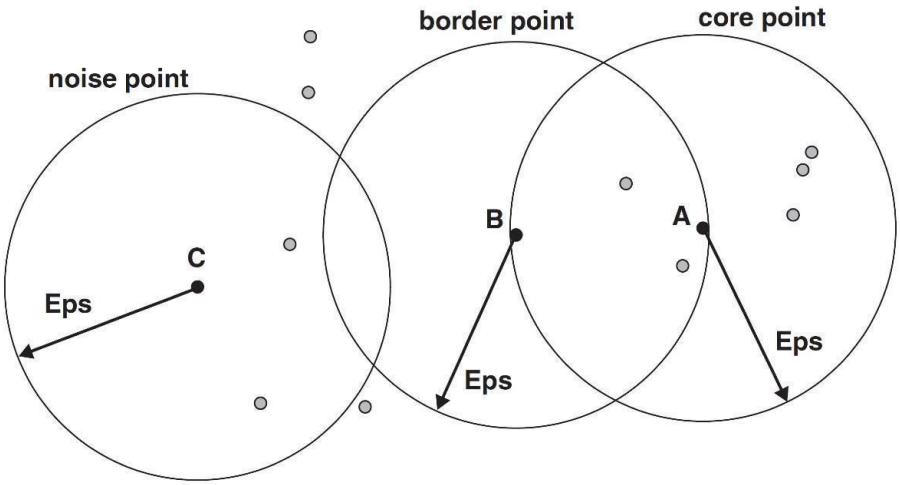
- There are three types of points in our data:

1. Core point: This is a point that has at least *minPts* points within distance *eps* from itself.

2. Border point: This is a point that has at least one Core point at a distance *eps and it is not a core point*.

3. Noise point: This is a point that is neither a Core nor a Border. And it has less than *minPts* points within distance *eps* from itself.

For e.g., with a value **minPts= 7**,



- The DBSCAN clustering algorithm can be summarized in the following steps:
  - ➔ Pick a random point from the dataset and if it is a **core point**, assign it to cluster 1 and mark it as visited.
  - ➔ If not, the point is marked as noise. Assign all the points within the neighborhood of the initial point to cluster 1. If these new points are also core points, assign their neighborhood point to cluster 1.
  - ➔ Next, randomly choose another point that has not been visited and repeat step 1 with it.
  - ➔ Stop when all the points are visited.

Deciding *epsilon*:

- One way to estimate the initial value of epsilon is:
  - ➔ Calculate the distance between each point.
  - ➔ Plot the histogram of those distances.
  - ➔ We may get two peaks, the initial value of epsilon may be one of the distance values between these two peaks.

Advantages of DBSCAN:

- ➔ It is able to find arbitrarily shaped clusters and clusters with noise (i.e. outliers).
- ➔ There is no need to decide ‘k’ (no. of clusters).

Disadvantages of DBSCAN:

- ➔ It does not work well with sparse points (high dimensional data).
- ➔ While DBSCAN is great at separating high-density clusters from low-density clusters, DBSCAN struggles with clusters of similar density.

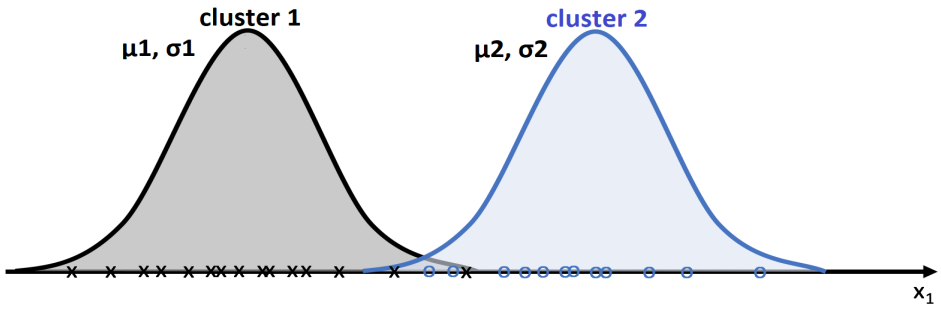
Soft clustering with Gaussian mixture model

- In soft clustering, a probability of each data point being in different clusters is assigned instead of putting each data point completely into a separate cluster.

- Gaussian Mixture Models are probabilistic models and use the soft clustering approach for distributing the points in different clusters.

- We can represent the clusters in our data using the **Gaussian distributions**.

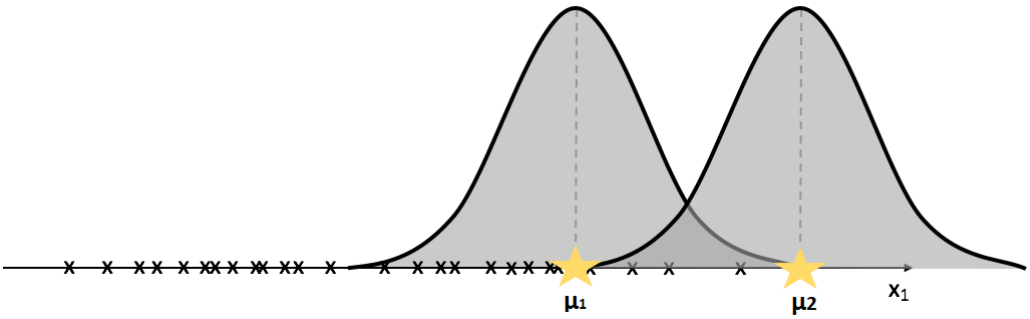
- For example, in the below given 1-Dimensional data, each cluster is represented using a Gaussian distribution.



1-D Example of GMM

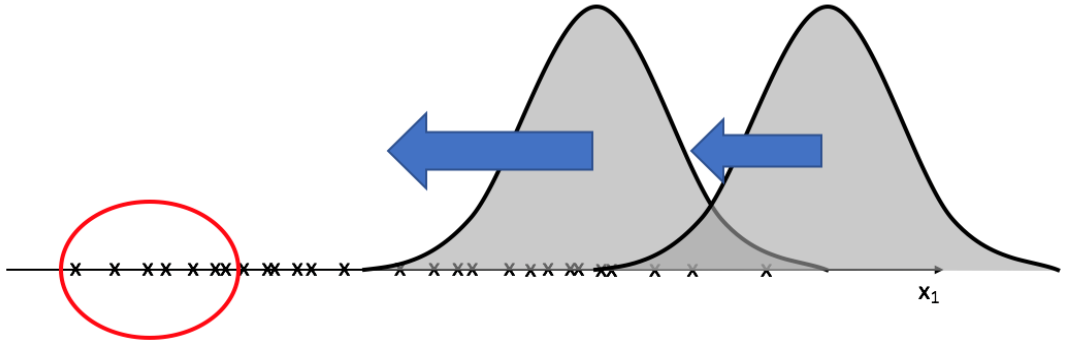
GMM algorithm:

➔ Randomly initialize μ and σ.



➔ Update μ and σ.

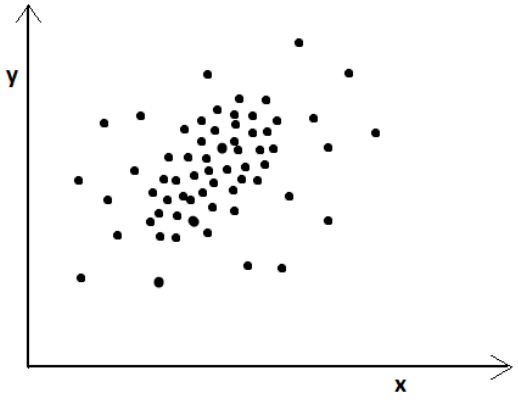
- Calc the probability of each point belonging to each gaussian.
- Assign all the points which have the highest probability to their respective Gaussians.
- Now, compute new μ for each gaussian as the mean of the points belonging to that gaussian (similar to Lloyd's algorithm)
- Similarly, update the variance.



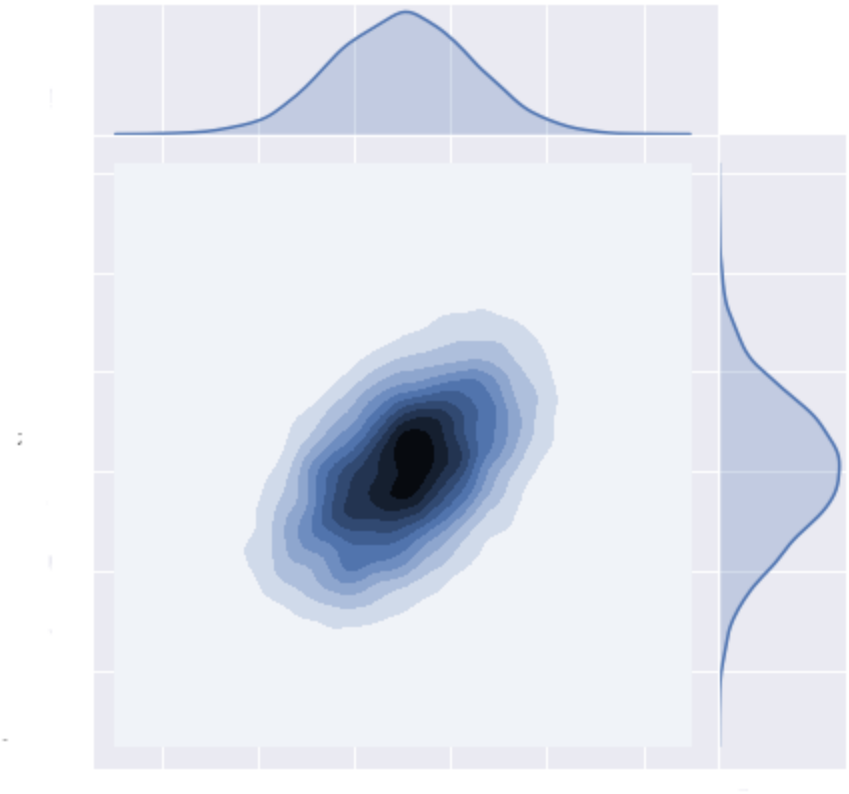
➔ After multiple updates, we will have tightly fitting Gaussian distributions representing different clusters.

2-Dimensional Gaussians:

- Let's consider two variables **x** and **y** with the following 2D scatter plot.



- The corresponding Multivariate Gaussian distribution would be:



- The part where the color is dark has more density of points and vice versa.

- The above multivariate Gaussian distribution is defined by five parameters:

    1. $\boldsymbol{\mu_x}$ - the mean of x-coordinate.
    2. $\boldsymbol{\mu_y}$ - mean of y-coordinate
    3. $\boldsymbol{\sigma^2_x}$ - variance in x direction.

    4. $\boldsymbol{\sigma^2_y}$ - variance in the y direction.
    5. **Cov(x,y)** - Covariance between x and y.

- The above ideas can be applied to 2D and ND gaussian, and this is how the GMM algorithm works.

Anomaly Detection

So far, you have learned many concepts that are there in Machine Learning

The purpose of this lecture is to let you know how you can take the concepts that you already know like Random Forests, SVMs, KNN, etc., and modify them for using them for another purpose rather than classification.

What is an Anomaly?

- Anomaly is synonymous with an outlier. These terms are often interchanged and may be called Novelty depending on the context.

What's the difference?

- Anomaly means something which is not a part of the normal behavior
- Novelty means something unique, or something that you haven't seen it before(novel)

Applications of Anomaly/Outlier/Novelty Detection:

- Credit Card Fraud Detection
- System Intrusion Detection
- Ecosystem Disturbances in Weather and Environment (alarming about Tsunami, Earthquake, etc.)
- You can read how some startups are thinking out of the box using these techniques from a blog here.

1. Distribution Based

- The simplest way for detecting an outlier would be to use distribution parameters (mean and standard deviation).
- Consider a feature X with some observations x_1, x_2,…., x_n with some outliers.
- We know that it will follow some distribution which will have parameters θ.
- Let follow a Gaussian distribution with some mean(μ) and standard deviation(σ)
- If we know the distribution of the data, we'll try to fit the distribution. However, the problem arises with the distribution parameters.
- While we know the distribution, the parameter estimates of the distribution are often corrupted by the noise/outlier
- Hence, we need to robustly estimate the parameters of the distribution. We do this using an algorithm called RANSAC which stands for Random Sample Consencus

2. Random Sample Consensus (RANSAC)

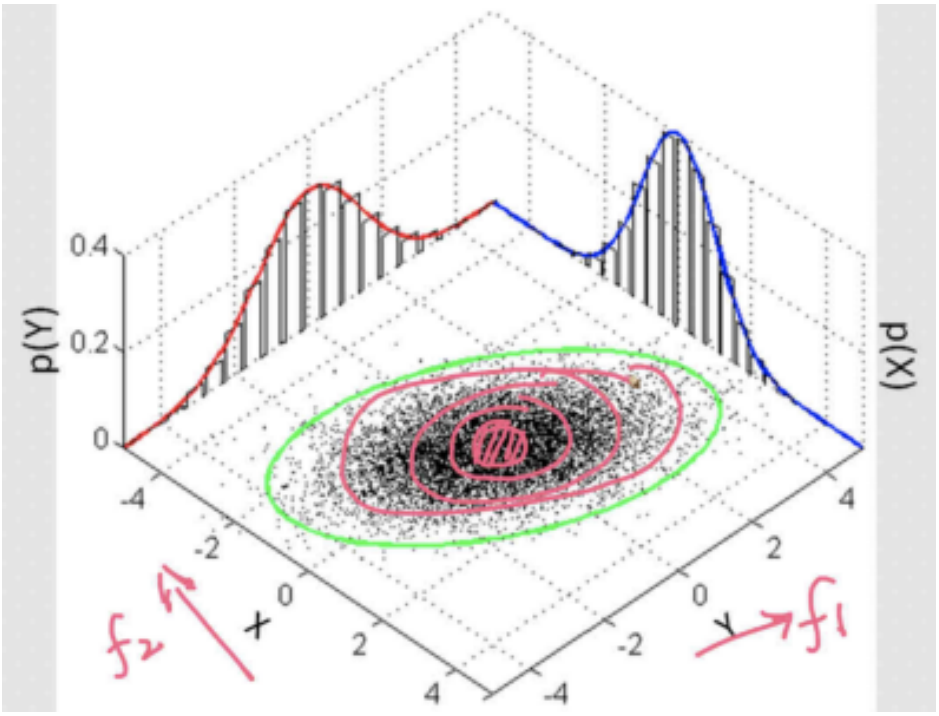
- RANSAC is a trial-and-error approach that works very well in real life.
- Imagine a dataset X with a number of points having parameters μ and σ^2. Let's call them collectively θ
- There are mainly three steps involved in RANSAC. They are
    - Sample a subset of points from the dataset (n'). We consider this point as an inlier.
    - Now, compute a model that estimates the parameters of the sampled points.
    - Score the model which indicates how many points will support the model.
- We repeat these three steps iteratively and then select the model best supported by the data, which then tells which points are inliers and which are outliers.

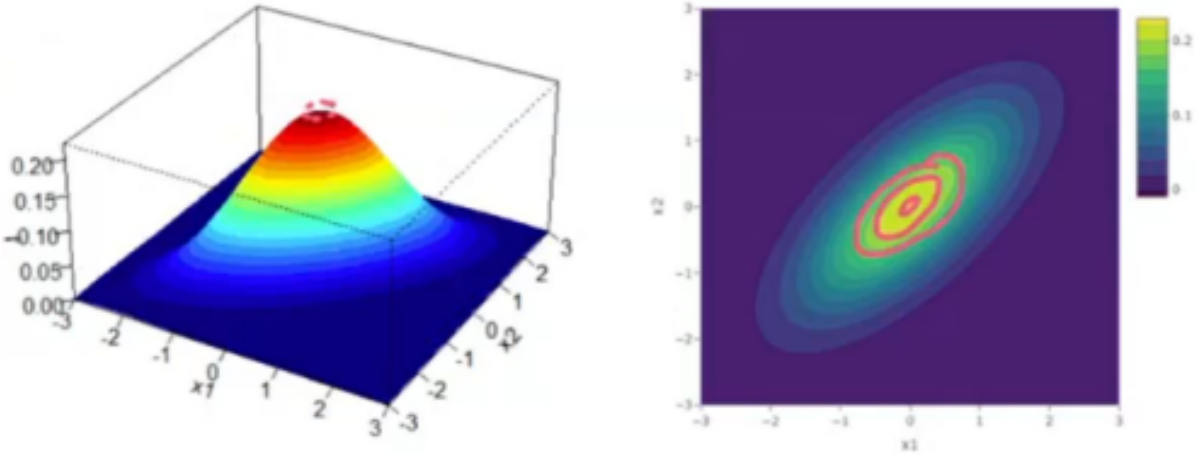
Extending the idea to higher dimensions

- Till now, we assumed that we have only a single feature X which follows Gaussian Distribution.
- Now, imagine we have d-dimensional data where each point x_i Є R^d and the data is not labeled.
- If we know the data points x_is follow multivariate Gaussian distribution(unimodal), then X follows normal distribution; X ~ ($\vec{u}$, Σ) , where is $\vec{u}$ mean vector and Σ is a covariance matrix
- Here we'll consider ($\vec{u}$, Σ) as θ
- If you remember from GMMS, in multidimensional space, the shape of gaussian was similar to a hill where the density of the points was highest in the middle contour, and it keeps getting low as we move away from the center
- In this case too, RANSAC can be applied. Farther away from centroid, we'll know that it is an outlier.
- Similar principle is used in another method called Elliptical Envelope.

3. Elliptic Envelope

- We know that a Unimodal Multivariate Gaussian Distribution on a single plane will look like ellipses if visualized on a plane. This idea can be extended to find out an outlier





- Given some data X where x_is Є R^d and X follows Normal Distribution being unimodal, Elliptical Envelope robustly estimate the parameters ($\vec{u}$, Σ).
- The term robustly means without getting impacted by outliers
- Next, then we remove the points that are outliers which are very far away from the centroid

What if the distribution is non-gaussian? Do we need to convert it into Gaussian distribution?

- While the elliptical envelope method assumes that the distribution is Gaussian, the strategy can be applied to any distribution.

- As long as we know any distribution and its parameters θ, we can extend our strategy to use RANSAC and estimate parameters θ.

- The can be other distributions such as multivariate Poissons, multivariate log normals, etc., but we don't use them as much.

- One other strategy is to convert them into Gaussians but we don't necessarily have to.

- As long as there is any distribution we can calculate the probability of x_i Є X using PMF/PDF.

Disadvantages:

- It cannot be used for non-unimodal data

- It is specifically for multivariate Gaussians

- If the data fails to meet the assumptions of unimodal and multivariate gaussian, the whole thing crashes.

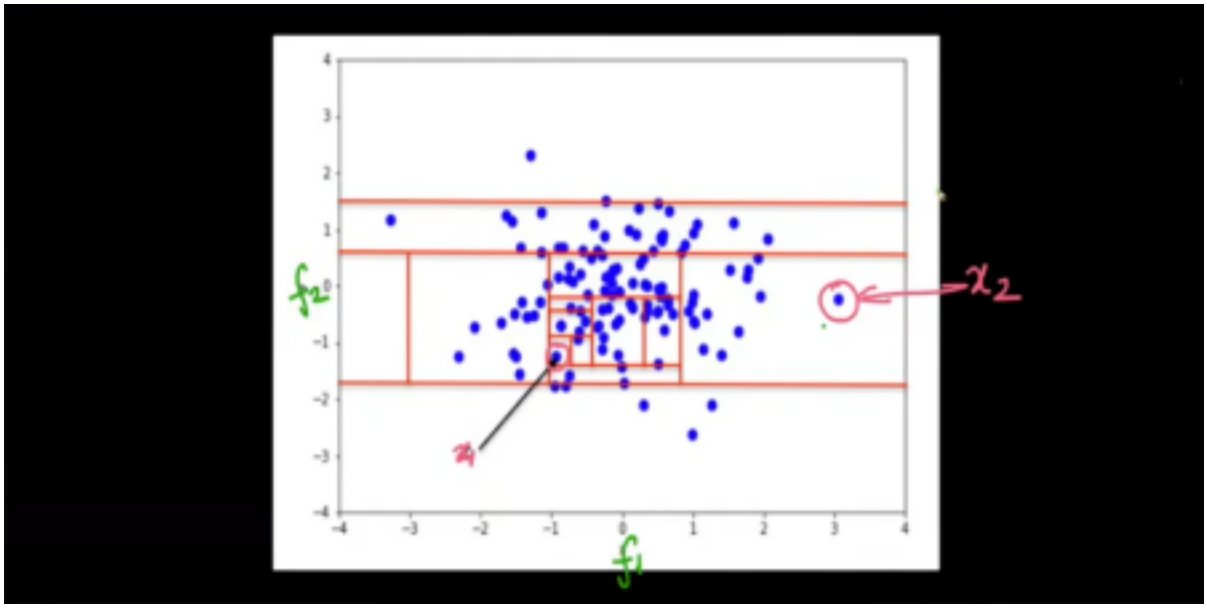
4. Isolation Forests

- Consider a dataset D which contains data points x_1, x_2,...., x_n. Just like Random forests, Isolation Forests build many trees.

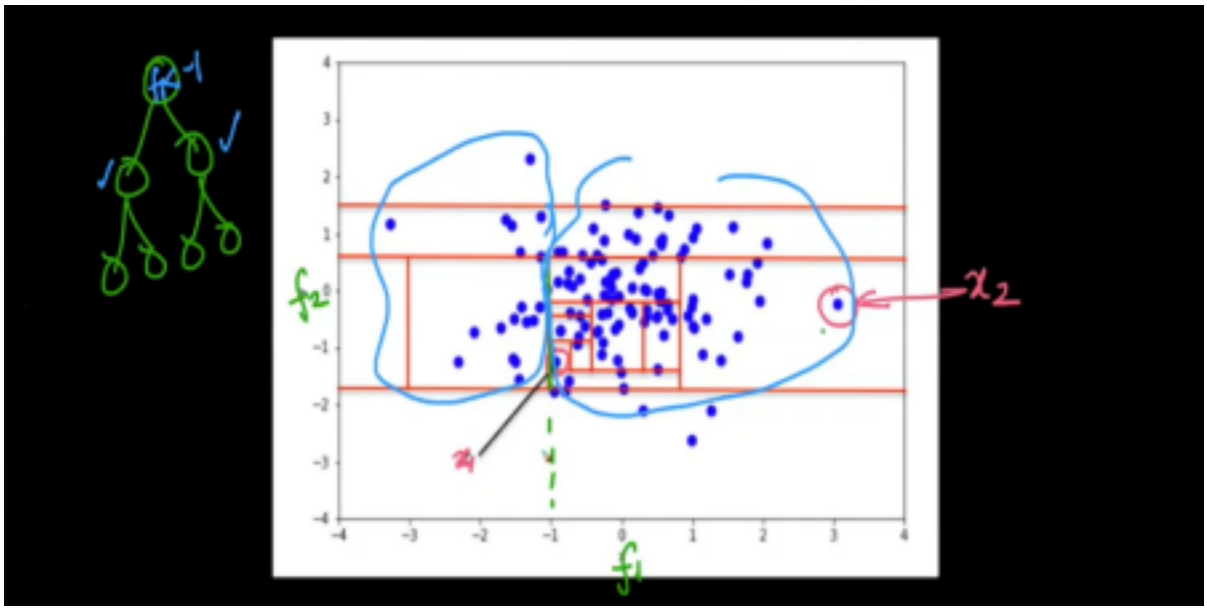
- These are the steps involved in Isolation Forest:
    - Build many trees like random forests
    - For each tree:
        - Randomly pick a feature
        - Randomly threshold that features
        - Build each tree until the leaf consists of only one datapoint

- Isolation Forests are also known as iForests

- Consider the plot along feature f_1 and f_2 given below:



- In isolation forests, we are building totally random trees. So if we pick feature f_1 and put a threshold there will be a vertical bar.
- Similarly, if we pick feature f_2 and put a threshold there will be a horizontal bar.
- For example, if we pick feature f_1 and we select threshold as $f_1 < 1$, then our first root node will be based on this condition



- Based on the diagram above,
  - The node containing x_1 will be at more depth.

    - Observe that the point x_1 is in dense region, and point x_2 is far away

    - That is because, to break the point x_1 from all the other points, more and more splits will be required and that will increase the depth of the node containing point x_1.

- So, to sum it up, the idea behind Isolation forest is:
    - On an average outliers have lower depth in the random trees
    - On an average, inliers have more depth in the random trees

Evaluation of Isolation Forest

- Imagine, we have build random trees. For each point x_i in the dataset, we can get an average depth.

- We use this average depth to convert into a metric.

- Apart from this, there are lot of different metrics, that people have came up with over the years

- But, the basic intuition is that lesser the average depth, higher likelihood is there that it is an outlier

Deciding average depth of a point

- There is no one metric specifically used for average depths in iForests. At the end, whichever metric you use, it is based on the threshold.

- There are a lot of metrics that researchers have came up with over the years.

- But, studying them in this lecture is out of scope.

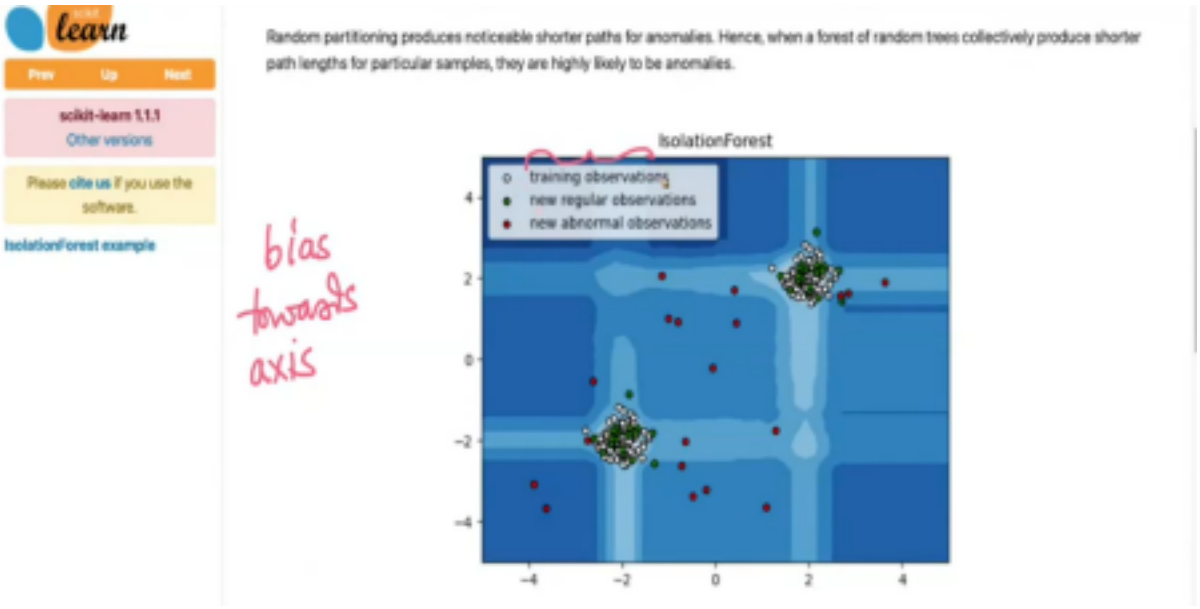
Disadvantages

- One of the major limitation of iForests is that they are biased towards axis parallel splits.

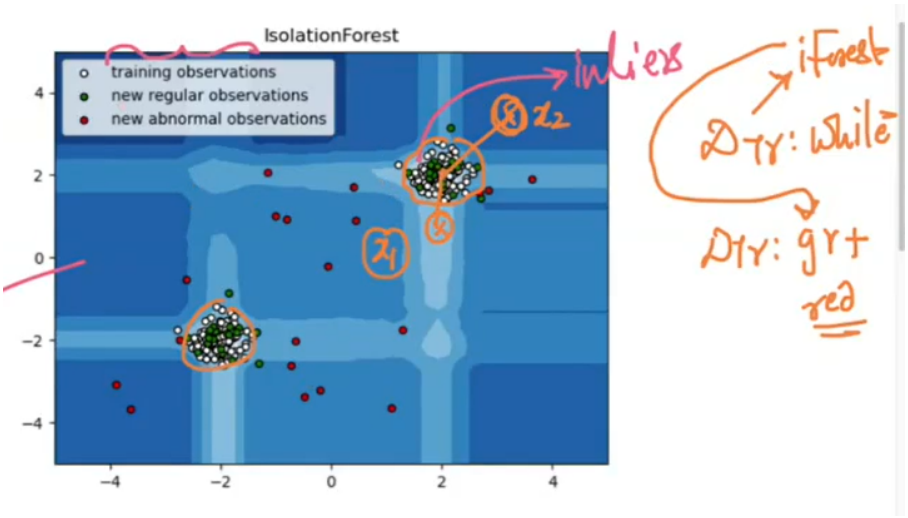
- iForests makes splits and these splits are always parallel to either of the axis.

- Because of this, the boundary will not be smoothed.
    - In the diagram given below, the different shades of blue represents the likelihood of a point to be an outlier. Darker the color, it is more likely that the point in that region will be an outlier

- We've trained the iForest model using training data(white points)
- It is tested on testing data(red + green) where red color indicates outliers



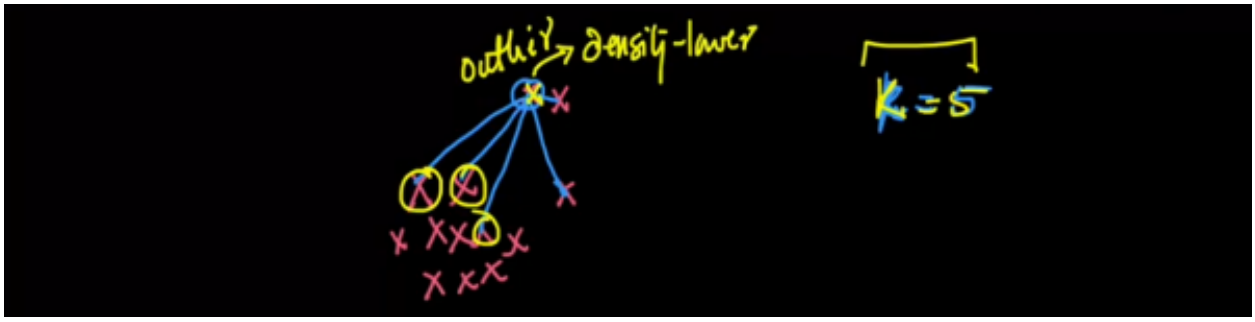
- Now imagine two points x_1 and x_2 as shown in the diagram given below.
- Both the points are almost equidistant from the nearest cluster. x_1 is on the axis and point x_2 is off axis.
- Because the model is biased towards axis parallel splits, it will classify point x_1 as an inlier and x_2 as an outlier
- This is also known as banding in signal processing



5. Local Outlier Factor (LOF)

- On a higher level, LOF is based on two ideas: KNN and density
- The core idea behind LOF is to compare the density of a point with its neighbors' density
- If the density of a point is less than the density of its neighbors, we flag that point as an outlier

- Imagine a bunch of datapoints as shown below



- We compute the density of a point based on average distance.
- If average distance between a point and its **K** nearest neighbors is large, it is more likely that the point will be an outlier
- Also, larger the value of **K**, more confident are the results.

Some concepts to understand the working of LOFs:

1(a) K-distance

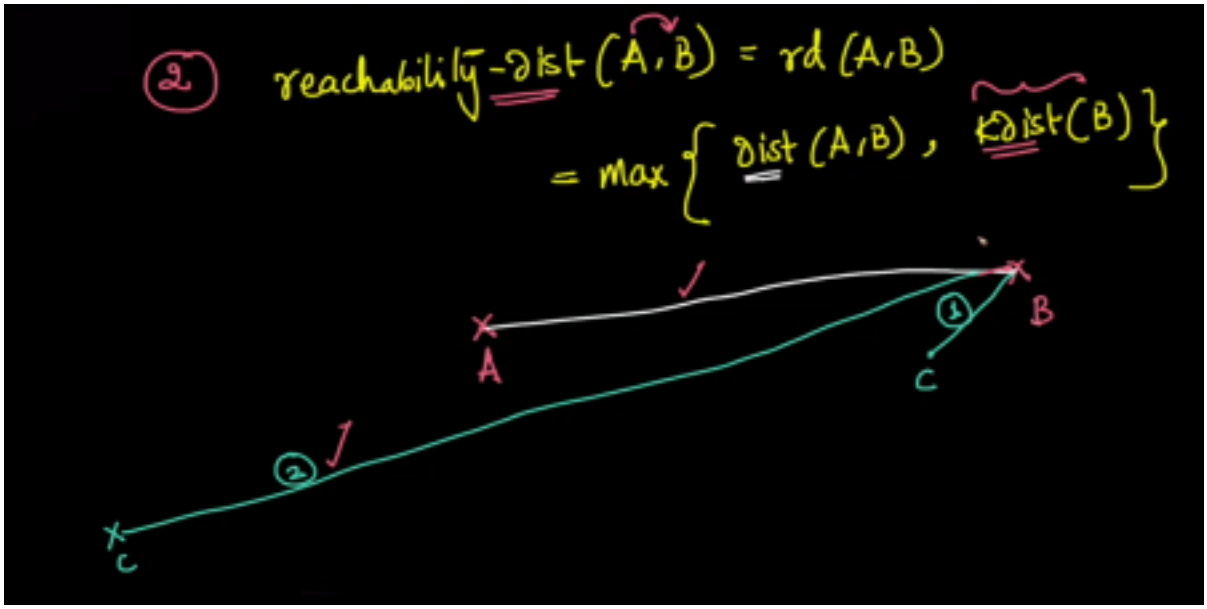
- We define K-distance of a point **A** as the distance of point **A** to its **Kth** nearest neighbor
- In general, larger the value of k-distance is, farther away the points is from other datapoints

1(b) Set: Nk (A)

- It is a set of k-nearest neighbors of point **A**.

2. Reachability distance

- From point **A** to point **B** , we define reachability distance as maximum of the distance from point **A** to point **B** and the maximum k-distance of point **B**
- Consider point **B** with some **k** nearest neighbors shown in the diagram below.



- There is a possibility that some neighbors might be close(condition 1) and some neighbors might be very far away(condition 2)
- In this case, there is a neighbor of point **B** whose k-distance is greater than the distance between point **A** and **B**, and hence, it is considered as its reachability distance.

3. Local Reachability Density

- It is often represented as **$lrd_k(A)$** , which tells local reachability density of a point **A**
- It is defined as the average reachability distance between point **A** and **k** neighbors

$$\text{So, } lrd_k(A) = \frac{\sum_{B \in N_k(A)} rd_k(A,B)\$}{N_k(A)}$$

-
- The summation in above equation represents the sum of reachability distances from a point **A** and set of neighbors **B** as **B** Є **$N_k(A)$**

- We define Local Outlier Factor of point as follows:

$$LOF_k(A) = \frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)|.ldr_k(A)}$$

  - **$ldrk_k(A)$** is the density of point **A**
  - The expression $\frac{\sum_{B \in N_k(A)} lrd_k(B)}{|N_k(A)|}$ is the average neighborhood density
  - So, LOF of point **A** is nothing but the average neighborhood density(lrd) of point **A** divided by the density of **A**

Interpretation of LOF

- If **LOF(A) = 1**, then we can say that the point has same density(lrd) as its **k** nearest neighbors
- If **LOF(A) > 1**,, then the **k** neighbors of point **A** have higher density than point **A**.
  - That does not mean point **A** is an outlier. It may or may not be.
  - But if **LOF(A) >>> 1**, then the point is definitely an outlier.
- If **LOF(A) < 1**, then the point has more density than its nearest neighbors.

Disadvantages of LOF

- Finding optimal K
- Finding threshold.
  - If LOF(A) >> 1, what is the threshold??

- Cannot handle high dimensional data efficiently
- High Time Complexity

****

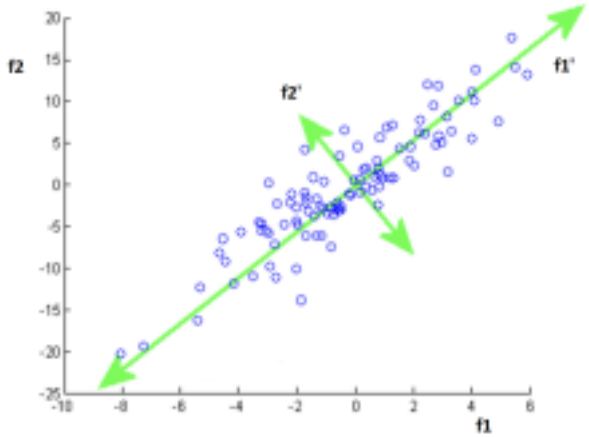
High Dimensional Visualisation

1. Principal Component Analysis (PCA)

- Dimensionality reduction techniques help to convert high dimensional data to fewer dimensions which can be then visualized using simpler plots or it can be used when we want to preserve the information of our feature columns.

- Principal component analysis, or PCA, is a dimensionality-reduction method that is often used to reduce the dimensionality of large data sets.
- PCA is an act of finding a new axis to represent the data so that a few principal components may contain the most information.
- The main objective here is whenever we are going from a higher dimension(d) to a lower dimension(d') we want to preserve those dimensions which have high variance (or high / information.)

  For example: Imagine we have two features f1 and f2, and the data is spread as shown in the image below. We want to project the given 2-D data to 1-D.



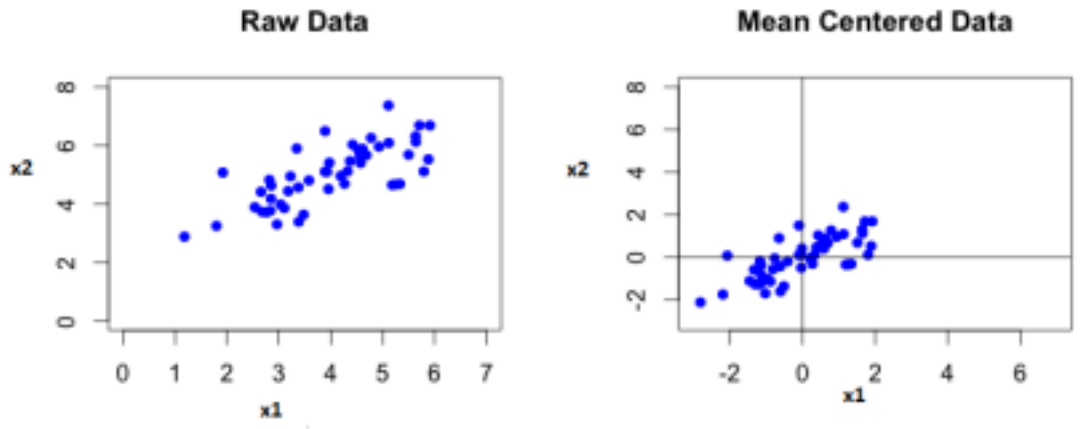
- One thing we can observe here is that both axes have a good amount of variance. It will be difficult to decide which feature to drop.

- Therefore, we rotated the entire coordinate axes. It is shown in green.

- Let's call these new dimensions as $\mathbf{f_1}'$ and $\mathbf{f_2}'$. Now we can see that variability on $\mathbf{f_1}'$ is more than $\mathbf{f_2}'$ and we can now drop the $\mathbf{f_2}'$ axis.

Mathematics of PCA:

Step 1: Feature wise Centering: We compute the mean of each feature and subtract it from each value of that feature to center the mean of the data at origin.



Step 2: Standardize: It is done in order to keep all the features of our data in same scale. Best is to use standard scaling because it is less affected by the outliers.

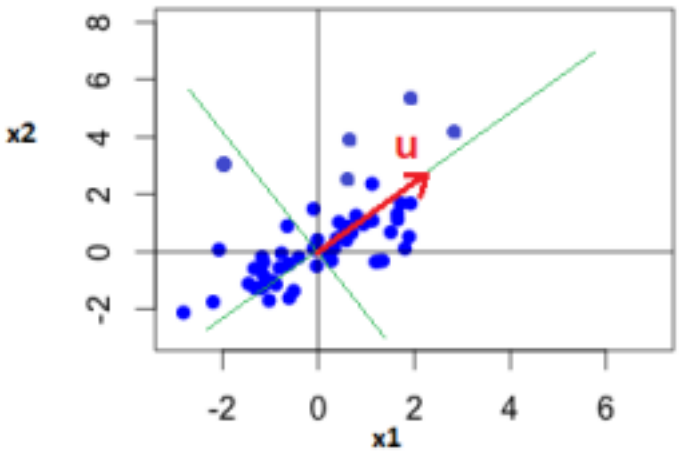
Combining step1 and step 2, we get the value of our feature vector x as:

$$x' = \frac{x - x.mean()}{x.std()}$$

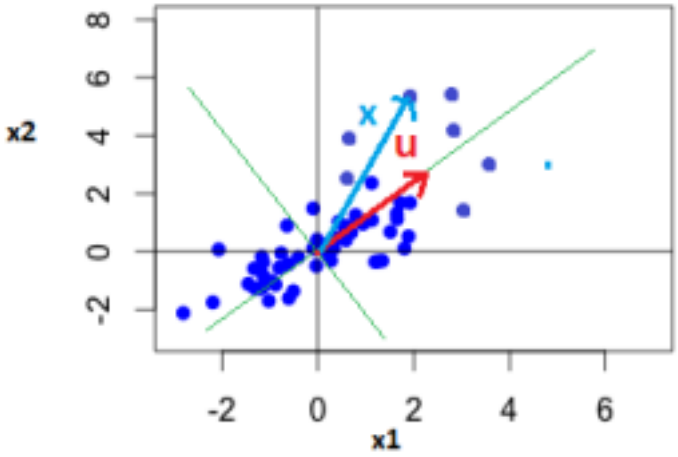
Step 3: Rotate the coordinate axes:

- There can be many angles of rotation possible, we'll consider a random axis and optimize it to get the best axis.
- Let's consider a unit vector in the direction of the new axis. $\vec{u}$

- Our optimization problem is finding unit vectors u such that the variance of all $\mathbf{x_i}'$ (features) are maximum.



- We can also think of it in terms of the length of projections.
- Consider a vector x in the space representing one of the points in our data.



- The projection of a vector x on vector u is given as: $\frac{\vec{u} \cdot \vec{x}}{||\vec{u}||}$
- The best u will be where the summation of the length of projections of all such points(xi) on the vector u is maximum.

i.e. $$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^{n} \frac{\vec{u}.\vec{x_i}}{||\vec{u}||}$$

- However, the projections may be a negative value. Therefore, we take the magnitude of the numerator:

i.e. $$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^{n} \frac{|\vec{u}.\vec{x_i}|}{||\vec{u}||}$$

- Since the above objective function is not differentiable, we rewrite it as:

$$\max_{\vec{u}} \frac{1}{n} \sum_{i=1}^{n} \frac{(\vec{u}.\vec{x_i})^2}{||\vec{u}||}$$

- Now, since u is a unit vector, we introduce a constraint: $||\mathbf{u}|| = 1$

- Using Lagrange's multiplier, our loss function becomes:

$$max_{\vec{u},\lambda} \quad \frac{(X.\vec{u})^2}{n} + \lambda(||\vec{u}|| - 1)$$

We can optimize the above Lagrangian loss function using Gradient descent.

Optimization without Gradient descent:

- We know that, $\vec{u}.\vec{x_i} = \vec{x_i}.\vec{u}$

therefore, our new loss function is:

$$L = \frac{(X.\vec{u})^2}{n} + \lambda(||\vec{u}|| - 1)$$

- Now, since we can write the constraint $||\vec{u}|| = 1$ as $||\vec{u}||^2 = 1$, therefore,

$$L = \frac{(X.\vec{u})^2}{n} + \lambda(||\vec{u}||^2 - 1)$$

- Now we know that If a matrix A is an invertible square matrix, then:

$$A^2 = A^T.A = ||A||^2$$

Thus,

$$L = \frac{(X.\vec{u})^T(X.\vec{u})}{n} + \lambda((u^T.u) - 1)$$

- Using the identity, $(A.B)^T = B^T.A^T$

We get,

$$L = \frac{(u^T.X^T)(X.\vec{u})}{n} + \lambda((u^T.u) - 1)$$

Or

$$L = u^T \frac{X^T X}{n} u + \lambda(u^T u - 1)$$

Let $\frac{X^T X}{n} = V$

- Hence, $L = u^T V u + \lambda(u^T u - 1)$

V is the pairwise covariance matrix of all the features of our feature matrix X.

- Taking partial derivatives w.r.t μ and λ,

$$\frac{\partial L}{\partial u} = 2Vu + 2\lambda u \quad \text{and} \quad \frac{\partial L}{\partial \lambda} = u^T u - 1$$

- After putting both the partial derivatives equal to zero, we get,

$$Vu = \lambda u \quad \text{— (i)}$$

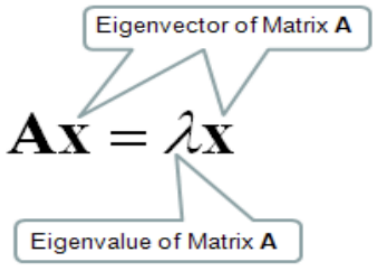
$$u^T u = 1$$

- By carefully observing the equation (i), we can conclude that u is the eigenvector and λ is the eigenvalue of matrix **V**.

Eigenvector and Eigenvalue:

- For any matrix A, there exists a vector **x** such that when this vector is multiplied with the matrix A, we get a new vector in the same direction having a different magnitude.

  - The vector **x** is called the Eigenvector and the length is called as

Eigenvalue. i.e. 

- There can be multiple eigenvectors, which are always orthogonal to

each other.

Conclusion:

- To find the best value of vector $\mathbf{u}$, which is the direction of maximum variance (or maximum information) and along which we should rotate our existing coordinates, we follow the below-given steps:

  Step 1: Find the covariance matrix of the given feature matrix X.

  Step 2: Then calculate the eigenvectors and eigenvalues of the covariance matrix. The eigenvector is the direction of best $\mathbf{u}$ and the eigenvalue is the importance of that vector.

- The eigenvector associated with the largest eigenvalue indicates the direction in which the data has the most variance.

- Therefore, we can select our principal components in the direction of the eigenvectors having large eigenvalues and drop the principal components having relatively small eigenvalues.

2. t-SNE (t-Distributed Stochastic Neighbor Embedding)

- The core idea of t-SNE is to preserve the pairwise distance in a neighborhood when the datapoint in a higher dimensional space is projected into a lower dimensional space.

- Therefore, t-SNE tries to create an embedding that preserves the neighborhood using some probabilistic methods.

Ideas for the probabilistic model

- We need to define the probability that a point x_i is a neighbor of x_j.

- One way is to define probability as inversely proportional to the distance between points.

i.e.

$$p_{ij} = \frac{1}{||x_i - x_j||}$$

- But there are some problems with the above probability function:
  - The function gives an infinite value when the distance between the two points is zero.
  - Probability reduces too fast.
  - The function depends on an absolute function.

- Therefore, we take inspiration from Gaussian distributions for a better PDF and a new probability function is defined as:

$$p_{ij} = \frac{\exp(-\left\|x_i - x_j\right\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\left\|x_k - x_l\right\|^2 / 2\sigma_i^2)}$$

- The standard deviation with respect to each x_i creates a scaled distance effect in the neighborhood of x_i and the denominator ensures that the sum of probabilities is 1.

Math for t-SNE

- Our objective is to project data points $x_i \in R^d$ to y_i using t-SNE, where $y_i \in R^2$
- We compute the pairwise similarities as probabilities.
- We compute P_{ij} for d-dimensions and Q_{ij} for d'-dimensions where d>d.

i.e.

$$p_{ij} = \frac{\exp(-\|x_i - x_j\|^2 / 2\sigma_i^2)}{\sum_{k \neq l} \exp(-\|x_k - x_l\|^2 / 2\sigma_i^2)}$$

where, $\mathbf{p_{ij}}$ is the probability that the points x_i and x_j are neighbors in *d*-dimensional space, and $\mathbf{q_{ij}}$ is the probability that the points x_i and x_j are neighbors in *d'*-dimensional space.

- We define $\mathbf{q_{ij}}$ with the same formulation as $\mathbf{p_{ij}}$ since every x_i and x_j would have corresponding y_i and y_j in d' dimensional space.

$$q_{ij} = \frac{exp(-||y_i - y_j||^2)}{\sum_{k \neq l} exp(-||y_k - y_l||^2)}$$

- Now, to get a useful d' transformation we need $p_{ij} \approx q_{ij}$. Thus, we need a loss function to optimize.

KL divergence

- It is a loss function that measures the dissimilarity between two distributions.
- The KL-Divergence between two distributions P and Q is written as:

$$KL - div(P_{ij}, Q_{ij}) = \sum_i \sum_j [P_{ij} \ . \ log(\frac{P_{ij}}{Q_{ij}})]$$

- Some interpretations using the KL-divergence formula are:
  - If P_{ij} and Q_{ij} are the same, then KL-divergence will be equal to 0.
  - If Pij is very small and P_{ij} and Q_{ij} are different, then KL-divergence will have a small value.
  - Lastly, since KL-divergence is a measure of dissimilarity, it is always **greater than or equal to 0.**

- Here, P is working as a weightage because if P_{ij} is small we don't really care as points x_i and x_j will be far away from each other in d-dimension space.

- Thus, our **optimization problem** would be to find all the y_is that **minimize** KL-divergence(P,Q).

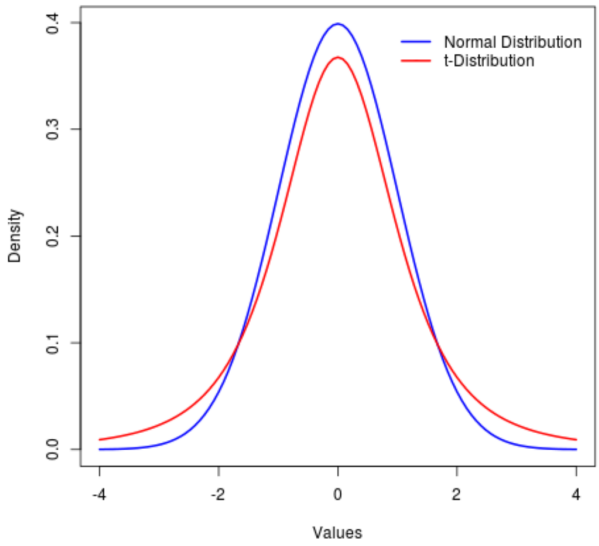
T-distribution (Cauchy Distribution)

- Researchers proposed the use of t-distribution with a degree of freedom equal to 1 instead of gaussian distribution for dimension d'.

- New q_{ij} is defined as:

$$q_{ij} = \frac{(1 + ||y_i - y_j||^2)^{-1}}{\sum_{k \neq l}(1 + ||y_k - y_l||^2)^{-1}}$$

- t-distributions work better than gaussian distributions with SNE because:
  - t-distributions with a degree of freedom equal to 1 have a longer tail than Gaussian distributions
  - Gaussian distribution falls exponentially while t-distributions sort of inversely
  - So, for our q_{ij}s , if we use t-distribution, two points can go farther away and still get pairwise distance preserved of the sort. Therefore, it is able to capture distant neighbors better.



- Therefore, we use Gaussian distribution for p_{ij}s and t-distribution for q_{ij}s because of which if two points are far away in lower dimensional space, the probabilities will still remain the same.

Perplexity

- Perplexity can be interpreted as the effective number of neighbors whose distance we want to preserve
- It is one of the parameters that we might want to configure when using t-SNE.
- Its value is kept in the range [5, 50]

****