CHAPTER 2

INTRODUCTION TO MACHINE LEARNING (Part 2)

2.1. Unsupervised Learning

Unsupervised learning is also known as Knowledge Discovery . It is a machine learning technique in which model are not providing any type of training to learn from data. It mainly deals with the unlabeled data and try to discover itself similarities and differences on the bases of patterns ,trend and in their relationship. Unlike in Supervised learning where , model on the bases of trained algorithms known about both input and its corresponding output .Where as in Unsupervised Learning ,input are known by model and model has to discover output.

Unsupervised Machine Learning is the process of training /teaching a model to use unlabeled,

unclassified data and enabling the algorithm to analysis pattern, feature, trend and relationship on that data without supervision.

For example:

In unlabeled dataset ,there is an image of both cow and goat and model has to recognize it.

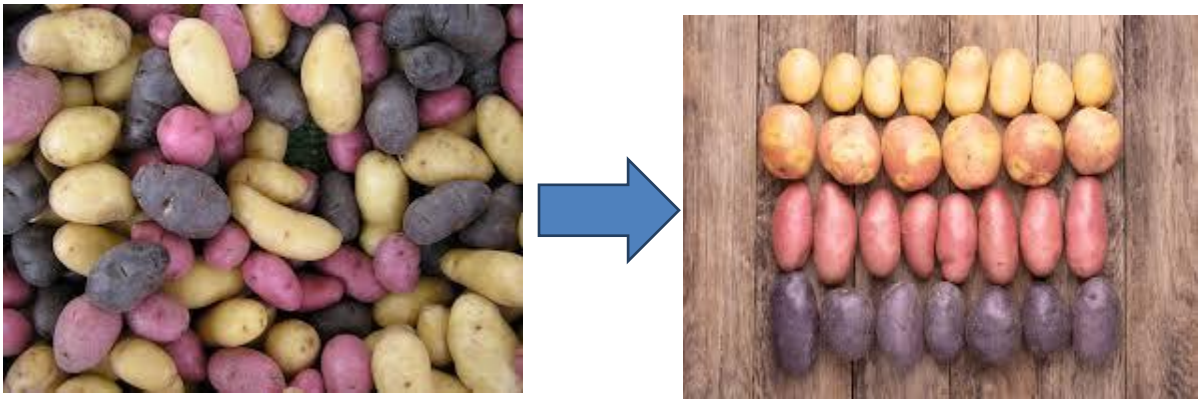
Figure1: Image of Both Cow and Goat

Hence , model first start it by discovering hidden pattern and trend ,and classify to categorize them in two classes, in one class it put images of cow and in second class it put images of goat.

2.1.1 Types of unsupervised learning techniques:

1. **Clustering:**

Clustering algorithm form cluster based on data point by discovering patterns, features ,variation or trend, which share common characteristics feature or attribute from a collection of uncategorized data in it.



Sample Cluster / Group

Figure 2: Clustering

From the above figure it is clear that ,there is sample which is a collection of uncategorized data that is., collection of various variety of potatoes which is clustered into similar variety of potatoes.

Common clustering algorithms include:

A. K-means:

It is a partition-based clustering method that aims to divide data points of unlabeled data set into K distinct clusters /group, where K is a user-defined parameter representing the number of

clusters / group. Its goal is to determine the similarities and dissimilarities within the given cluster/group.

Figure 3: K-means Clustering

The above figure represent a cluster of apple, this cluster is further cluster into K cluster on the bases of quality like color ,shape and size. So here, the value of K is 4.

A. Hierarchical clustering:

It is also known as hierarchical cluster analysis or HCA , this is tree-shaped structure known as the 'Dendrogram'. In dendrogram, individual data points are located at the bottom of it, while the largest clusters, which include all the data points, are located at the top of it.

There are two types of hierarchical Clustering:

A. Agglomerative Clustering :

It is also known as the bottom-up approach or hierarchical agglomerative clustering (HAC) where each data are treated as a singleton cluster at the outset and then successively a large cluster of many different things which are collected or brought

together are further cluster into pairs of clusters until all clusters have been merged into a single cluster that contains all data.

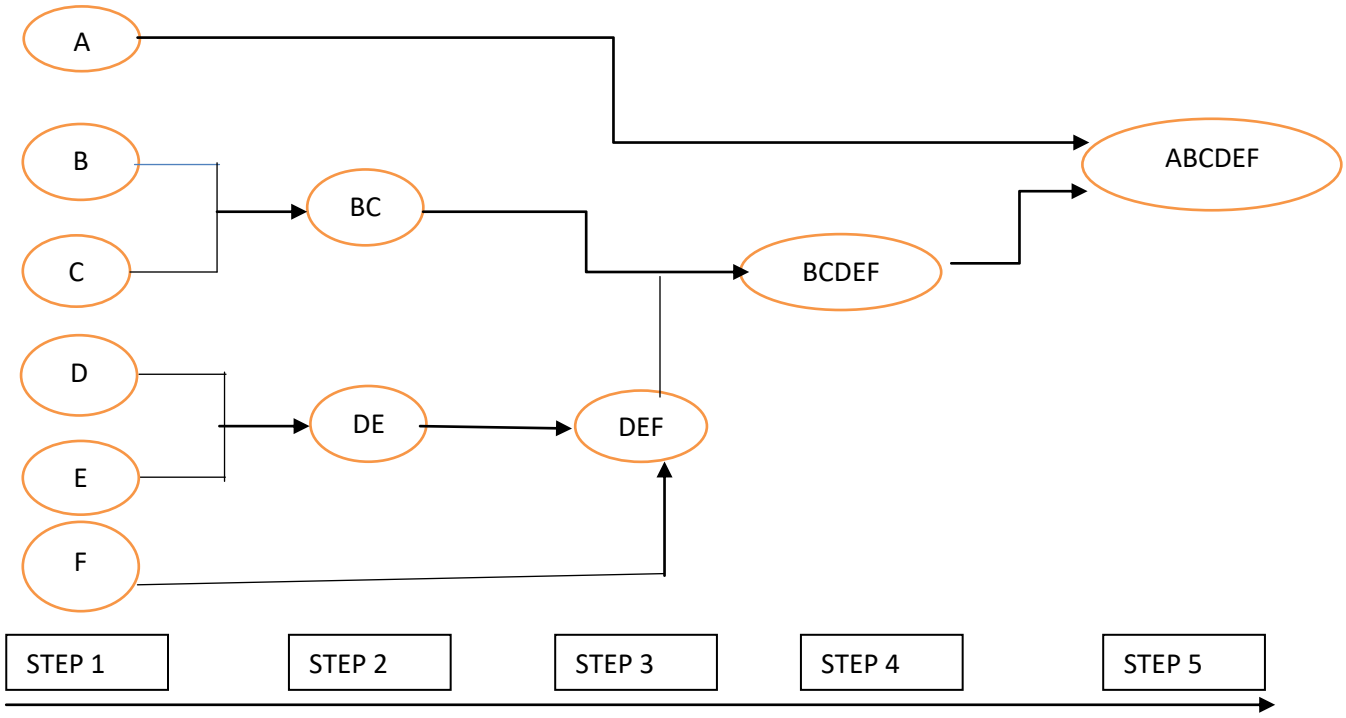


Figure 4: Agglomerative Clustering

I. **Divisive clustering** :

It is also known as a top-down approach. In this method a cluster that contains the whole data are farther split into clusters recursively until individual data have been split into singleton clusters.

Figure 5: Divisive clustering

B. Density-based spatial clustering of applications with noise (DBSCAN):

In 1996 .it was introduced by Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu .It is used for clustering data points based on their density distribution in a feature space. Mainly it can identify clusters of arbitrary shapes and enable to handle noisy data effectively . DBSCAN works by defining two key parameters: ε (epsilon) and MinPts.

- ε (epsilon) : It is the radius that determines the neighboring distance around each data point.

- MinPts : It is the minimum number of data points which is present within ε distance.

The DBSCAN algorithm can classify data points into three categories:

- Core Points: Data points that are within ε distance of neighboring points and have at least MinPts .
- Border Points: Data points that are within ε distance of a core point but do not have enough neighbors to be core points themselves.
- Noise Points: Data points that are not within ε distance of any core point and do not have enough neighbors to form a cluster.

DBSCAN has various applications in data analysis, including spatial data analysis, anomaly detection, image processing, and customer segmentation.

C. Mean Shift :

It is also known as the Mode-seeking algorithm .Mean-shift clustering is a non-parametric, density-based clustering algorithm that can be used to group similar data points together into clusters without the need for labeled data. It is particularly useful for datasets where the clusters are in distinct shapes and are not well-separated by linear boundaries, or when similar data points of unlabeled data set are not cluster in same cluster and the number of clusters is not known beforehand.

Steps in Mean Shift clustering algorithm are as follows:

Kernel and Bandwidth Selection:

The algorithm begins with the selection of a kernel function and a bandwidth parameter whereas the kernel function determines the weight or influence of neighboring data points in density estimation, and the bandwidth sets the size of the neighborhood used to calculate the density.

Data Point Initialization:

It initialize all data points as potential cluster centers.

Mean Shift Vector Computation:

For each data point, the algorithm computes a mean shift vector that points towards the region of highest density. This vector is calculated by taking average weight of the data point which connects to its neighbors, while the weights are determined by the kernel function and bandwidth parameters.

Update Data Points:

Shifting the data points in the direction of their corresponding mean shift vectors.

Convergence:

The mean shift vector computation are repeated and updates data point until convergence is achieved.

Clustering:

The final clusters represent the groups of common data points based on density.

E. Gaussian Mixture Model (GMM):

It is a probabilistic model used for clustering data into multiple groups or clusters.

Where in each cluster, the data points are generated from a Gaussian distribution,

and the overall data distribution is a combination of these Gaussian distributions. GMM find the parameters that include the means, variances, and mixing coefficients of the Gaussian components

by using the Expectation-Maximization (EM) algorithm.

In a one dimensional space, the probability density function of a Gaussian distribution is given by:

$$f(x \mid \mu, \sigma^2) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

where μ is the mean and σ2 is the variance.But this would only be true for a single variable.

In the case of two variables, the probability density function would be given by:

$$f(x \mid \mu, \Sigma) = \frac{1}{\sqrt{2\pi|\Sigma|}} \exp\left[-\frac{1}{2}(\boldsymbol{x}-\boldsymbol{\mu})^t \Sigma^{-1}(\boldsymbol{x}-\boldsymbol{\mu})\right]$$

where x is the input vector, μ is the 2D mean vector, and Σ is the 2×2 covariance matrix.

The GMM clustering process works in the following steps:

1. Initialization: Choose initial values for the means, variances, and mixing coefficients of the Gaussian components.

2. **Expectation-Maximization (EM) algorithm**:The EM algorithm consists of two steps:

a**. Expectation step (E-step)**: It calculates the probability of each data point belonging to each cluster based on the current parameter estimates.

b**. Maximization step (M-step):** It updates the parameters of the Gaussian distributions to maximize the likelihood of the data given the estimated probabilities from the E-step.

Iteration: Repeat the E-step and M-step until convergence, it means that when the model parameters stop changing significantly.

Assign clusters: After convergence, assign each data point to the cluster with the highest probability.

GMM can handle complex and overlapping clusters and can also estimate the probability that a data point belongs to a particular cluster or not. GMM is determining the optimal number of clusters (components), using model selection techniques such as the Bayesian Information

Criterion (BIC) or the Akaike Information Criterion (AIC) to compare models with different numbers of components and choose the one that best fits the data.

G. Spectral Clustering :

It first convert the data points into a graph representation and then perform clustering on this graph. The graph is constructed based on pair-wise similarity or distance measures between data points. In graph it representation nodes as the data point and the similarity between the data points are represented by an edge.

The steps involved in Spectral Clustering are as follows:

1. **Construct the Affinity Matrix:** To constructed it take a given a dataset with n data points, represent the pairwise similarity between the data points. The similarity can be measured using various distance metrics (e.g., Euclidean distance, cosine similarity, or Gaussian kernel) based on the characteristics of the data in the particular domain.

2**. Graph Representation:** The above matrix is then treated as the adjacency matrix of a weighted graph. In the graph each data point corresponds to a node and the edge weights represent the distance between the similar points.

3. **Graph Partitioning**: The graph partition is into k disjoint clusters or subgraphs by optimization. This is often done by finding the k smallest eigenvectorsof the Laplacian matrix of the graph.

4**. Forming Clusters**: The k eigenvectors obtained from the above graph are arranged as rows in a new matrix.This are treated as new data points in a higher-dimensional space. Finally, traditional k-means clustering techniques are applied to these new data points to form the final clusters.

2.Dimensionality Reduction:

Dimensionality reduction is a fundamental technique in unsupervised learning that aims to reduce the number of features or variables in a dataset without the lost of its relevant information . The primary goal is to simplify the data representation, making it more manageable and easier to analyze, visualize, or process, and enable to reveal patterns, structures, or clusters within the data, making it easier to understand and explore.

There are two main approaches to dimensionality reduction in unsupervised learning:

Feature Selection: It select a subset of the relevant original features from the dataset while discarding the rest or unused features. Common methods which include is information gain, mutual information, and statistical tests like chi-squared test or t-test.

Feature Extraction: Unlike feature selection, it transforms the original features into a lower-dimensional space using various mathematical techniques. Now the new set of features are known as latent variables or components, which are more informative and compact representation.

PCA and Auto – encoder are the popular techniques for feature extraction :

I). Principal Component Analysis(PCA): This technique was introduced by the mathematician Karl Pearson in 1901. It mapped the data of higher dimensional space into the data of lower dimension space and the variance of the data in the lower dimensional space should be maximum. (PCA) is a statistical procedure that uses an orthogonal transformation that converts a set of correlated variables into a set of uncorrelated variables. It is used to examine the interrelations among a set of variables. It is also known as a General Factor. analysis where regression represent a line of best fit.

(PCA) is used to reduce the dimensionality of a data set by finding a new set of variables that should be smaller than the original set of variables, without the lost of its relevant information , and useful for the regression and classification of data.

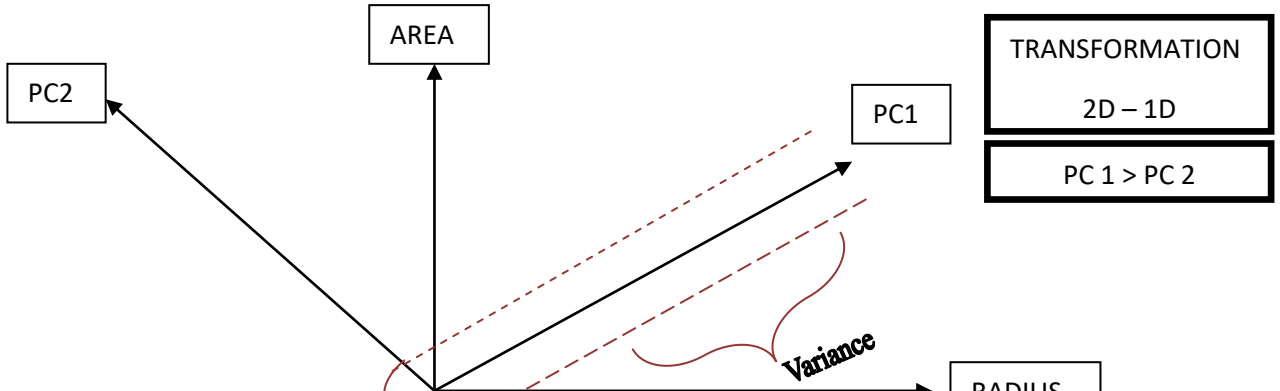


Figure 6: Principal Component Analysis(PCA)

The following steps of PCA (Principal Component Analysis) are:

Step 1: Standardization

First, we need to standardize our dataset to ensure that each variable has a mean of 0 and a standard deviation of 1.

$$\mathbf{Z = X-\mu / \sigma}$$

Here,

$\mu = \{\mu_1, \mu_2, \ldots\ldots\ldots, \mu_m\}$

$\boldsymbol{\mu}$ = The mean of independent features .

$\boldsymbol{\sigma}$ = The standard deviation of independent features .

$\boldsymbol{\sigma = \{ \sigma_1 , \sigma_2 , \ldots\ldots, \sigma_m\}}$

Step 2: Covariance Matrix Computation

It measure the strength of joint variability between two or more variables, representing how much they change in relation to each other. To find the covariance we can use the formula:

$$\mathbf{Cov\ (x1\ ,x2) = \sum_{i}^{n}{}_{=1}\ (x1i\ \ x1)\ (x2i\ \ \ x2)\ /\ n\text{-}1.}$$

The value of covariance can be positive, negative, or zeros.

Positive: As the x1 increases then x2 also increases.

Negative: As the x1 increases then x2 also decreases.

Zeros: No direct relation for zero.

Step 3: Compute Eigenvalues and Eigenvectors of Covariance Matrix to Identify Principal Components

Let A be a square nXn matrix and X be a non-zero vector for which λ, for some scalar values λ then λ is known as the eigenvalue of matrix A and X is known as the eigenvector of matrix A for the corresponding eigenvalue.

$$\mathbf{A\ X = \lambda\ X}$$

It can also be written as :

$$\mathbf{AX - \lambda\ X\ \ = 0}$$

$$\mathbf{(A - \lambda\ I)\ X = 0}$$

where I is the identity matrix of the same shape as matrix A. And the above conditions will be true only if **(A – λ I)** will be non-invertible (i.e. singular matrix). That means,

$$\mathbf{|(A - \lambda\ I)| = 0}$$

From the above equation , we can find the Eigen values and then corresponding eigenvector can be found.

II). .Auto-encoders:

Auto-encoders belong to the family of neural network architectures that determines how the nodes are inter-connected, how the information flows within the network, and how the network performs various specific tasks in ML, that includes classification, regression ,trends, feature or pattern recognition .The network are designed to learns from receive input data, include an efficient feature or pattern from data, extract its essential characteristics and transform it into compressed version of given original input data ,while Principle Component Analysis (PCA) finds the directions along which you can project the data that cover maximum data points or variance.

Steps in Auto-encoders are as follows:

1. **Encoding level:** Input data is fed into encoder network ,which usually has several hidden layers. Each layer reduces the rest of the input data, preserving higher characteristics and patterns.
2. **Decoding level:**

The coded representation is then passed through the decoding network, , the purpose of which is to create original input data. The architecture of the decoder is usually symmetric or mirror image with respect to the encoder.

3. **Loss of function:** The difference between the original input and the reconstructed input is measured by a loss function such as mean square error (MSE) or binary cross entropy. The purpose of the network is to minimize this loss, to learn to create a patterns, but to provide an accurate representation of the input data.

   The types of auto-encoders in unsupervised learning are:

   A. **Vanilla Auto-encoder** : This is the basic form of Auto Encoder, which consists of an encoder and a decoder. Its purpose is to study data entry negotiation. The encoder reduces the dimensionality of the input and the decoder reconstructs the original input. It is trained to minimize restructuring**.**

   B. **Sparse Auto-encoders**: In sparse auto-encoders, the encoder is encouraged to learn different notations. This means that only a small number of neurons in the hidden layer are activated simultaneously, can make information encoding more efficient .

Sparseness can be accomplished by various techniques such as normal operation or fines.

C. **Noise Canceling Auto-encoders**: Noise canceling auto-encoders are trained to reproduce clean data from noisy devices. During the training, the network took corrupted versions of the login and training to recreate the original, clean data. This helps auto-encoders learn dynamic features and filter out noise.

D. **Contractive Auto-encoder:**

The Contractive auto-encoder adds a loss penalty that prevents the encoder from detecting minor changes. This empowers learning agent to capture more important features and reduce the impact of random noise.

E. Variational Auto-encoder (VAE): VAE is a design that models data distribution as well as learning the fit of data. They use a probabilistic encoder and decoder to learn the hidden area where data can be sampled and reproduced.

3. Anomaly Detection:

In the context of machine learning, anomaly detection refers to the process of identifying patterns or data items in data that differ from expected behavior or patterns. These differences are often called anomalies, outliers, innovations, or exceptions. Vulnerability detection has many applications in many fields such as finance, cybersecurity, manufacturing, healthcare, and more.

Anomaly detection in machine learning works like this:

Data collection and preprocessing: The first step is to collect and prepare the data for analysis. This includes cleaning the data, handling incomplete results, and converting the data into a suitable format for further processing.

Feature selection and engineering: Selection or engineering of related features that capture data features. This step is important because the quality of features can affect the performance of malicious detection algorithms.

Defining Normal Behavior: Diagnosing abnormal behavior requires understanding what is causing the behavior. This can be done in many ways, including statistical methods, expert knowledge, and even learning from historical data.

Choose an anomaly detection method: There are several methods for detecting anomalies, each with advantages and disadvantages.
Some of the methods are:

Statistical method: This method involves measuring the distance between points of points or other central measurements. Examples include z-scores, modified z-scores, and Mahalanobis distance.

Machine Learning Techniques: Supervised and unsupervised machine learning algorithms can be used. Unsupervised techniques such as split forest, single class SVM, and k-means clustering can be effective.

Deep Learning: Deep neural networks can be used for abnormal detection through training models to reproduce original models.
The content of data containing construction errors is considered unreliable.

Time Series Analysis: Special techniques such as autoregressive integral moving average (ARIMA) or exponential smoothing can be used for time dependent data.

Consolidation: Combining multiple detection vulnerabilities can improve detection overall.

Training and evaluation: If machine learning is used, the algorithm will be trained on ordinary data. Use labeled or unlabeled data to evaluate the performance of algorithms based on whether the issue is audited.
Personal Choices: Set a threshold to determine what's wrong. Data points or patterns below this threshold are considered normal and data points or patterns above this threshold are

declared abnormal. Thresholds can be set manually or selected according to measurement parameters.

Reporting and Monitoring: Once a vulnerability detection system is established, it should be regularly monitored and updated to reflect changing data patterns. At the same time, new vulnerabilities may arise and the system must be able to catch them.
Dealing with False Positives and False Negatives: A false positive test can be false positive (data always marked negative) or negative (no false positives). It is important to balance these variations and is often used privately.

In a nutshell, anomaly detection in machine learning involves identifying different patterns or themes in data. The choice of technique depends on the nature of the data, the availability of data to be recorded, and the desired balance between good and bad. It's important to tailor the process to a specific problem and refine it as data evolves.

4. Association Rule Mining:

Association rule mining is a technique used in machine learning and data mining to find relationships or relationships between items in big data. This technique is particularly useful in market basket research, where the goal is to identify patterns in consumer behavior.

The association mining policy works like this:

Data collection and progression: The first step is to collect transaction data, which usually includes a list of products purchased on each exchange. For example, in a retail environment, every transaction will be a customer's shopping cart with many items.

Item Frequency Calculation: Calculate the frequency of each item in the database.
This includes counting how often each product appears on the market.

Support Threshold Setting: Set the support threshold that represents the minimum change frequency or percentage this is important. This threshold helps filter out less popular items.
Support is an important part of the collective mining policy that determines which items will be considered "parties" and therefore used to establish organizational rules. Support for a product is defined as the proportion of the market in which the product is available. Initial support helps evaluate less useful items as useful or interesting.

Support configuration in mining policy attribution works like this:

Contribution Ratio: For each product (portfolio), Support is calculated by dividing the number of products with products from all industries in the dataset.

Support (item X) = (number of changes with X) / (total number of changes)

Starting point: Choose a support point based on experience names, data attributes, and the desired balance between sensitivity and specificity.
This threshold represents the minimum support level at which an item should be considered multiple times.

A lower value causes more objects to be considered more often, which leads to more convergence.

A higher setting will result in fewer items being evaluated more often and therefore fewer rules, but these rules will be more important and more enforceable.

Filter rare items: All items with an incentive value higher than the selection are considered active items. Substances with stimulus values below the threshold were considered rare and excluded from further analysis.
Create join rules: Create join rules from active items based on metrics such as confidence and increment. These codes represent relationships between items that appear frequently in the dataset.

It is important to remember that the choice of support depends on the specific purpose of the analysis, the nature of the data, and the insight you seek. There is no one-size-fits-all approach to setting thresholds; It is an iterative process, often with testing and evaluation.

If the threshold is set too high, you will not see interesting but rare patterns in your data. On the other hand, if the threshold is set too low, you may encounter too many active items and associated rules; some of these may not be important or interesting.

You can consider the following steps to help build an appropriate backing:

Start with a very high starting point to get a small amount of important product and code.
Gradually lower the threshold and monitor changes in products and policies
Use process visualization to understand the distribution of support benefits and their relationship to very active products.
Consider registration information and the benefits of policy making.
After all, finding the right balance between support and understanding is the key to successful joint mining policy.

Create Timeline Items: Check for items that meet the support criteria. These items are called active items.
Favorite items represent combinations of items that often occur together in a document.

Create Association Rule: Create association rules from active objects. Association rules are expressions of the form "If X then Y"; where X is a set of objects (called antecedents) and Y is another object (called antecedents). These rules capture the relationship between what tends to be shared in a business.
Let's look at a simple example of creating join rules using the Apriori algorithm. In this example we will consider sales data that includes commercial buyers. The aim is to identify common rules that define the relationship between different objects.

Dataset, discount, document code, transaction ID, Items to buy

---

1 bread, milk

2 breads, diapers, Beer, eggs

3 Milk, diapers, beer, coke

4 Bread, milk, diapers, beer

Active items: {bread, diapers}, {diaper, beer}, {diaper, coke}

Create association rules:

We create association rules from active elements.

Association rule: {bread} -> {diaper}

Association rule: {diaper} -> {beer}

Association rule: {diaper} -> {cola}

Evaluation and pruning rule:

Every trust We calculate the rule based on association rules and pruning with a minimum confidence threshold.

trust({bread} -> {diaper}) = support({bread, diaper}) / support({bread}) = 60% / 80% = 75%

trust

({diaper} -> { beer }) = support({diaper, beer}) / support({diaper}) = 40% / 80% = 50%

confidence({diaper} -> {cola}) = support({diaper, Coke}) / Support ({ Diapers }) = 40% / 80% = 50%

Trimmed Association rule: {Bread} -> { Diapers}

Result:

In this example, we use the Apriori algorithm to create organization rules . We found that the {Bread} -> {Diaper} rule is reliable enough and useful in decision making.

Note that this example is simple and real-world data can contain many additional objects and relationships.
It also includes additional steps such as the organization's policy design, evaluation of additional indicators (eg leverage, leverage) and identification of top-level items.

Policy Analysis: Evaluates shared policies against a variety of metrics, including:

Support: Percentage of businesses with both prior revenues and results.
Reliability: The percentage of transactions that contain both the history and the result.
Lift: Specifies whether a prefix can form a prefix. Removal > 1 indicates a good relationship.
Pruning and Selection: Codes that meet certain conditions (such as minimum trust or support thresholds) are chosen logically and usefully.

Interpretation and Practice: Interpretation of the rules set by the organization can provide business perspective.
For example, in the retail environment these rules can drive sales or implement marketing strategies.

Visualization and Reporting: The results of corporate policy mining can be visualized using tools such as graphs or tables, making it easier for stakeholders to understand and follow the findings.

Continuous Analysis: Attribution mining is an iterative process. As new data becomes available, the analysis can be repeated to reveal changing patterns and trends.

It's worth noting that enterprise policy mining can suffer from the "curse" when dealing with large files or large objects.

To solve this problem, techniques such as Apriori algorithm, FP-growth algorithm, and Eclat algorithm have been developed to efficiently search for active objects and generate relevant rules.

Association rule mining in general is an important technique for discovering hidden patterns and relationships in business data and has applications in many fields beyond market sales, including professional, medical and fraudulent.

5. Generative Models:

Generative models are a class of machine learning models that aim to learn and model the consequences of the distribution of given data. This model can create a new data model that resembles the original data and captures the existing structure and structure in the data. The design can be used for image design, text design, data augmentation, detection, etc. It has many uses, including

There are many types of designs, and each has its own methods and features. Here are some similarities:

A.Auto-encoder: The Auto-encoder is a type of neural network consisting of an encoder and a decoder.

The encoder maps the input data to the lower level data source, while the decoder maps the hidden data source back to the original data source. Variable auto-encoder (VAE) is a special type of auto-encoder that combines distributions to create data that resembles the distribution of the original data.

B. Variable Auto-encoders (VAE):

VAEs are modular designs that combine auto-encoders with modular designs. They learn to encode information into a distribution in the hidden space and allow new information to be generated by sampling from this distribution. VAEs provide a way to generate new knowledge while exploring gaps in distributed knowledge.

C. Generative Contention Networks (GAN):

GANs have two neural networks: a generator and a splitter. The generator creates a new data model while the operator tries to separate the actual data from the data generated by the generator. The two networks are trained inconsistently, causing machine learning to produce data that is difficult for a human to distinguish from real data.

D. Pixel CNN and Pixel RNN:

These models focus on generating data point by point, pixel by pixel. Pixel CNN (Conditional Neural Network) and Pixel RNN (Recurrent Neural Network) are models that generate images to match the order of pixels.

They learn to estimate the value of each pixel based on previously created pixels, making the image consistent and realistic.

Pixel CNN (Pixel Conditional Neural Network) and Pixel RNN (Pixel Recurrent Neural Network) are well-known algorithms for image processing. They are part of a family of models called autoregressive models that sequentially generate data one item at a time when conditioned to prebuilt items. Pixel CNN and Pixel RNN have the unique ability to create pixel-by-pixel images in an integrated and structured way while preserving fine details.

Pixel CNN and Pixel RNN work by modeling the distribution of each pixel based on previously generated pixels. This allows them to capture the progress of neighboring pixels and create images with local consistency.

Here is a brief summary of Pixel CNN and Pixel RNN:

Pixel CNN:

Pixel CNN is introduced as a model that predicts the distribution of the current pixel according to the values of its left neighbors and top. It uses a mask to ensure that the model is based only on previous pixels.

Pixel CNN main features:

- **Masked Convolution:** At each layer of the model, a convolution filter is applied to the input image. However, because the filters are masked, the estimate is based on previous pixels only. This avoids looking at future pixels that haven't been rendered yet.

autoregressive rendering: Pixel CNN renders the image autoregressively, starting from the left corner, line by line and pixel by pixel.

individual pixel values: Pixel CNN typically models the distribution of discrete pixel values using the soft-max activation of the output layer for each pixel.

- **Pixel RNN**: The Pixel RNN takes the autoregressive concept one step further by using a convolutional neural network (RNN) to model the progression of pixels. Pixel RNN uses RNN layers to capture the distribution instead of using layers like Pixel CNN.

- **Pixel RNN key features:**

Linear LSTM and Crossed BiLSTM: Pixel RNN uses two types of RNN: linear LSTM and diagonal BiLSTM (BiLSTM).

Line LSTMs reproduce pixels line by line, while crossover BiLSTMs capture long-range dependence of crossover images.

Autoregressive Generation: Similar to Pixel CNN, Pixel RNN automatically generates images regressively, allowing each pixel to be estimated based on previous pixels. The

Pixel CNN and Pixel RNN do well in rendering tasks, creating consistent and visually appealing images. However, they also have limitations such as being slow due to their poor flexibility and inability to design universal models as well as other designs such as productive competitor networks (GANs) or variable auto-encoders (VAEs).

These models paved the way for advances in design and inspired later models that combined their strengths with other techniques to create beautiful images.

- **Solved Pixel CNN and Pixel RNN example in Machine Learning**

Of course! Let's see how Pixel CNN and Pixel RNN create images with simple examples. In this example we will use a 3x3 grayscale image with pixel values ranging from 0 to 255.

Dataset:

Consider a small grayscale image:

Image:

----- - -- ----

100 150 200

50 120C 14080 4 Pixels 4 44

Masked Convolutions: In Pixel CNN, masking is used to ensure that each pixel estimate is based only on previous pixels. Let's say we have a 3x3 convolution filter.

For pixel position (120), the filter will cover:

Filter:

----

* * * *

* X *

* * *

Prediction: Model, Left and upper neighbors (Conditional distribution of pixel position 120 for values 100 and 150). Determine the distribution approximated using the Soft-max technique and the sample pixel values of this distribution.

Pixel RNN:

Linear LSTM: Pixel RNN uses RNNs to capture pixel dependencies. We use linear LSTMs for simplicity. The pattern is created line by line, starting from the left pixel.

Autoregressive Generation: The LSTM row for each row limits the prediction of each pixel to the pixels previously generated in that row. Let's say we create a second row:

The first pixel (50) is approaching the top pixel (100).

To the second pixel 120, the top and left pixels 100 and 50 are approximated.

The third pixel 180 approaches the top and left pixels 150 and 120.

Diagonal BiLSTM: Pixel RNN also uses Diagonal BiLSTM to detect diagonal lines.

These LSTMs capture long-term trends in images.

Creation process:

Pixel CNN:

Start with a blank image. Using the
estimated distribution, construct the middle pixel (120) from the top and left pixels (100 and 150). Section
Continue rendering other pixels

Pixel RNN:

Create first pixel row from pixels: 50, 120, 180.
Create the second row using the LSTM cold row of pixels created in the first row: 80, 110, 220.

Note: In practice, Pixel CNN and Pixel RNN models have more layers and use more methods to solve many problems. This example shows the basics of how this model can be used to create autoregressive graphs.

Pixel CNN and Pixel RNN both generate sequences of images, preserving the local population. They can render interconnected graphs, but due to their autoregressive nature, they may be slower than other designs and are less capable of capturing spherical models.

E. **Normalization Flow:** The Normalization Flow is a method designed to transform simple distributions, such as the Gaussian distribution, into a smoother distribution that matches the data distribution. They contain a set of dynamic variables that allow them to model the distribution of complex data and build good models.

Transformer-based models: Models such as the GPT (Generative Pre-trained Transformer) family are designed for language processing but can also be used for text. They use self-monitoring techniques to capture long-term prospects in the data and establish communication and key points.

F. Boltzmann Machines:

Boltzmann machines are a class of probabilistic generative models used to model binary data and learn the fundamental distribution of data. Constrained Boltzmann Machines (RBMs) are a popular alternative for learning representation and data modeling.

Structured models are especially useful where data generation, augmentation, or simulation is required.

They can generate new data that can be used for education, testing or research. They also play an important role in tasks such as transformations, data connections and even creative ideas.

Boltzmann machines (BM) are a class of structural models that model combinations of binary variables. They are introduced as power-based models that aim to capture the basic structure of the data and examine the evolution of variables. Boltzmann machines can be used for many tasks such as minimization, custom learning and coordinate filtering.

The following are important terms and concepts related to Boltzmann Machines:

- **Nodes (Neurons):**

Boltzmann Machines consist of a group of nodes where each node represents a binary variable. Nodes are divided into visible volumes and hidden volumes.

Visible classes correspond to visible data, while hidden classes capture hidden properties.

- **Links (Edges) and Weight**: There are links between

nodes and each link has a weight. Weight determines the strength of interaction between nodes and affects the probability of the model's distribution.

- **Strong Power:** The Boltzmann Machine uses a power-based model approach. The power function provides a power value for each visible and hidden configuration.

The energy function defines the relationship of a configuration to model parameters.

- **Probability Distributions:** The probability distribution of the Boltzmann machine is defined using a power function. The probability of a given configuration is proportional to the negative of its energy. The Boltzmann distribution is often used to generate useful values.

- **Education and Training:**

Learning in Boltzmann mechanics involves adjusting the weights to fit the training data. However, making the most of training data directly is difficult due to the interaction between hidden and visible units. Markov Chain Monte Carlo (MCMC) methods such as Gibbs sampling are often used for training.

G. Gibbs Sampling:

Gibbs sampling is a technique used in Boltzmann machine learning. It includes updating the status of visible and hidden units while keeping other units stable. This method helps to estimate the distribution of the sample for the data distribution.
Gibbs sampling is a Markov Chain Monte Carlo (MCMC) technique often used in machine learning and statistics to estimate random variables, especially when direct sampling is difficult or difficult. It is commonly used when you want to create a model from a multivariate distribution or calculate the expectation of that distribution.

Gibbs Sampling works like this:

- **Markov Chain Introduction:**

Markov Chain is a sequence of variables in which each variable depends only on the previous variable. Gibbs sampling works in the framework of Markov chains.

- **Destination Distribution:**

Suppose you have more than one distribution (usually in common form) and you want to sample from it. However, sampling directly from this distribution may not be possible or difficult.

- **Initialization:**

Initializes with an initial value for each variable in the distribution. These initial results may be arbitrary or based on an opinion.

- **Iterative Sampling:**

With each iteration, you update the value of one variable while keeping the value of another variable. Systematically select the variables to be updated and sample their new values from the distributions of the other variables against their current values.

- **Conditional Sampling:** The main idea of Gibbs sampling is to adjust one variable each time based on the current value of another variable. This includes adjusting the values of other variables and sampling from the distribution of variables obtained using Bayes' theorem.

- **Stationary Distribution:**

From time to time, as the iterations progress, the Markov chain approaches the stationary distribution. In the context of Gibbs sampling, the target distribution you want to predict is a fixed distribution.

- **Multiple Chains and Aging:**

Multiple independent Gibbs sampling chains are usually run from different initializations to achieve convergence. In addition, an "aging" period is often used to allow the chain to stabilize before samples are collected for analysis.

- **Collected models and estimates:**

Models aggregated from the Markov chain after addition can be used to estimate from the marginal many variables of interest, such as expectations, variances or distributions.

Gibbs sampling is particularly useful when sampling from shared distributions with high spatial distributions, complex dependencies, or difficult normalization. It is a versatile

technique that can be used for many things, including Bayesian statistics, image modeling, machine learning, and more.

It is important to note, however, that Gibbs sampling may have limitations for similar relationships, such as slow correlations and sensitivity to order changing with updating.
As such, it often requires careful attention and care to make sure it works properly.
Let's see Gibbs Sampling example in Machine Learning. We will consider a situation where we want to sample from both sides of the joint using Gibbs sampling.

- **Example problem**: Sampling from a Bivariate Gaussian Distribution

Suppose we have a Bivariate Gaussian Distribution with the following parameters:

- **Mean Vector:**

μ = [2, 3] : Co-variable [0.5, 2]]

Our aim is to sample from this distribution using Gibbs sampling.

Gibbs Sampling Steps:

- **Initialization:**

We start with the initial values of two variables, let's say x = 0 and y = 0. another variable. We are changing the x and y variables.

. Update x, given y:
We start with the conditional distribution P(x)
Yes). This distribution is a univariate Gaussian distribution with **mean μ x y** and **variance σ^2xy;** where :

μ x y = μ x + Σ x y * (y – μ y) / Σ y y

and **σ^2xy = Σ x x - Σxy^2 / Σ y y.**

Given x update:

We sample new y values from the conditional distribution P (y x). This distribution is also inverse Gaussian with mean μ y x and variance σ^2y.

x, where

μ y x = μ y + Σ y x * (x – μ x) / Σ x x

and **σ^2 y x = Σ y y – Σ y x^2 / Σ x x.**

Iterations:

For some iterations we repeat the iterative process. The samples obtained from this process are transferred to the common target.

Example procedure:

Let's run a few Gibbs sampling examples:

Initial values: x = 0, y = 0

Iteration 1:

x update to y: Example from P (G x y = au) section). Update y to

x: Sample y (Gaussian distribution) from P (y x = 0).

. Update x to Y: model x from P(x)

y from the previous iteration). Section

Update y, given x: example y from P (y x from previous iteration).

Repeat for more.

Result:

After running Gibbs Sampling for a sufficient number of iterations, the sample collection converges to a bivariate Gaussian distribution defined by the mean vector and covariance matrix.

This example shows the basic steps for Gibbs sampling from a simple distribution. In practice, Gibbs sampling can be applied to various distributions and dimensions to estimate their covariance and to estimate the range of interest.

H. **Restricted Boltzmann Machine (RBM):** The RBM is a variant of the Boltzmann machine where there is no connection (visible-visible or hidden connection) between nodes in the same layer. This limitation facilitates training and enables better learning.

Applications:

Boltzmann machines and RBMs have been used in a variety of tasks, including collaborative filtering, recommendation, graph learning, and regression. They can capture interactions in data and learn meaningful representations. It is worth noting that training

Boltzmann machines can be expensive, especially for large datasets and deep models.
For this reason, they are often combined with other techniques such as pre-training, optimization, and hybrid models.

In general, Boltzmann machines and their variants provide a probabilistic framework for data distribution models and learning properties of complex data. Although somewhat influenced by recent developments such as deep neural networks, they are still valid in some cases and provide insight into the fundamentals of design patterns and unsupervised learning.

6. Embedding:

In machine learning, "embedding" refers to the process of representing categorical or discrete variables (such as words or objects) as a continuous field. This change allows machine learning algorithms to process these changes more efficiently by capturing the relationships and emotions between them. Embeds are widely used in many tasks, particularly in natural language processing (NLP) and recommendation systems, where they have proven very effective.

Here are some important terms related to embedding in machine learning:

- **Word embedding:**

In NLP, the word embedding is used to represent words as density vectors in a vector space. Words with similar meaning or meaning are adjusted to cover vectors in that space.
Popular word insertion methods include Word2Vec, GloVe, and Fast-Text.

- **Asset Embeds:** Similar to the term Embeds, entity embeds represent entities (eg, users, objects) in a persistent environment. These embeds can capture relationships between organizations and can be used in the approval process and information applications.
- **Embedding layer:**

In neural networks, the embedding layer is used to learn embedding directly from data during training. These layers specify logical parameters (for example, word indexes) for continuous vector representation. Layers are often used in network models such as convolutional neural networks (RNNs) and Transformers.

- **Dimensions:**

The dimensionality of the embedding is an important parameter. Too low dimensionality causes data loss, too high dimensionality causes overfitting. Choosing the right size is often done by trial.

- **Pre-Trained Placements:**

Pre-Trained Placements are trained on large corps and are useful for transferring knowledge from one task or field to another. For example, embeds can be customized for a specific task.

- **Embedding Similarity:**

Embedding similarity can be measured using cosine similarity, Euclidean distance, or other distance metrics. Similar burials should have a small distance between them.

- **Visualization:**

  The can use size reduction techniques such as t-SNE or PCA to visualize the image in a lower dimension. This can help to understand the relationships captured by the insertions.

- **Interpolation and Analogy:**

Insertions often reveal interesting things.

For example, you can create new insertions that capture semantic transformations by interpolating between two word insertions. An example can be made by adding/removing vector variables.

- **Infrequent Introduction to Dense Vectors:**

Embedding are particularly useful for handling categorical variables of great importance. They transform sparse single-bit encoded inputs into dense vectors.

- **Transfer Learning:**

Integrating learning from one project to another is beneficial, allowing learning to be transferred with minimal documentation and improved performance.

Embedding play an important role in representing and understanding complex data and improve the performance of machine learning models in many applications.

7. Density Estimation:

Density estimation is an important concept in engineering and statistics that involves estimating the length (PDF) of different models or datasets. The PDF explains the result of the difference between the two variables. Density estimation is used to understand data distribution, analyze data, make predictions, and create new data models. It has applications in many areas such as anomaly detection, clustering, classification and modeling.

There are many methods for density estimation, and each has its own methods and properties:

i. **Histogram:**

The histogram divides the data into boxes and counts the number of data text content in each section.

The height of each bin represents an estimate of the intensity of the range. When the histograms are flat, the size should be chosen carefully to avoid smoothing or over-smoothing.

ii. Kernel Density Estimate (KDE):

KDE uses a kernel function (eg Gaussian) between each data point to generate a density estimate.

Write the nuclei to construct the population rate function. KDE provides better representation of data distributions than histograms.

Let's look at an example of Kernel Density Estimation (KDE) in machine learning. KDE is a method for estimating the probability density function of a continuous variable based on a set of data points. In this example we will use KDE to estimate the probability distribution for one-dimensional data.

Example problem: Probability estimation using KDE

Suppose we have data on the test scores of students in a class.

We want to use KDE to estimate the distribution of test scores.

Dataset:

Exam Scores: [70, 75, 85, 90, 78, 82, 88, 92, 80, 85, 95, 70, 744 KDE4 process:

Select kernel : . Function that can be used to generate a smoothed estimate of possible speed. An alternative is the Gaussian (normal) kernel.

Select Bandwidth:

Specify the Bandwidth parameter, which determines the width of the kernel function and controls the level of smoothing. Smaller bandwidths result in more detailed but less noisy data, while larger bandwidths result in better but less accurate predictions.

Core Placement:

For each data center, place the core in the middle of the point based on core options and bandwidth.

Kernel Contribution:

Calculates the contribution of each kernel to the density index of different points in the range of data values.

Total Contributions:

Adds the contributions of all cores to get the final KDE estimate of the possible rate function.

KDE Example: Let's estimate the probability distribution of test scores using a Gaussian kernel with a bandwidth of 5.

Calculation:

Suppose we want to estimate the density of a point x = 80.

We will insert a Gaussian kernel into each data point and calculate their contribution at x = 80.

Contribution to the kernel: weighted by Gaussian between data points, where bandwidth is affected and Gaussian width is affected.

calculates the contribution of all cores to get the estimated speed at x = 80.

Result:

KDE estimate of probable speed at

x = 80 The mathematical value represents the estimated probability based on a score around 80 of the given data.

Note that this is a simple example.

In practice, you will use libraries or programs that use KDE to perform calculations efficiently and accurately. KDE is especially important when you want to see the distribution of data, especially when the data does not follow the distribution.

iii. Parametric Methods:

Parametric methods assume a certain functional form of the fundamental distribution (eg Gaussian, Exponential).

The parameters of the selected distribution were estimated from the data to fit the proposed distribution. Examples include Gaussian mixing models (GMMs) and exponential families.

iv. **Nonparametric method:**

The nonparametric method does not assume a specific distribution and provides more flexibility. In addition to KDE, other non-parametric methods include Parzen Windows, nearest neighbor method, and local polynomial regression.

v. Mixture Models: Mixture models combine several simple distributions (mixtures) to model complex data distributions.

An example is GMM, where all components are Gaussian. Expectation maximization (EM) algorithms are often used for parameter estimation in mixed models.

vi. Neural Density Estimation:

Deep learning like normalized flow and variable auto-encoders learns complex maps from simple distributions like Gaussian distributions to Desired data distributions.

These methods allow for faster prediction and can also be used for general models.

a. Bayesian Methods:

Bayesian methods take preconceptions about a distribution, combine them with observed data, and estimate the posterior distribution. Bayesian density estimation is particularly important when given prior knowledge.

Density estimation is used in several ways:

a. **Impaired detection**: Low-probability erroneous data is generally considered improbable.
b. **Clusters:** High-density areas may indicate clusters or clusters in the data.
c. **Classification:** Density estimation can aid classification tasks by comparing the densities of different units.
d. **Generative Modeling:** Density estimation methods can be used to model underlying data distributions and create new data models.

The selection of the appropriate density estimation method depends on the characteristics of the data, the level of smoothness required, and the complexity of the classification model.

Let's use Bayesian linear regression to understand a simple example of the Bayesian approach to machine learning.

In this example, we will use Bayesian inference to estimate the parameters of the linear regression model and their uncertainties.

Example Problems: Bayesian Linear Regression

Suppose we have a data pair (x, y) where x represents the input and y represents the target variable. We want to fit a linear regression model to the data and estimate the parameters of the regression line (slope and intersection) using Bayesian methods.

Model:

A linear regression model is defined as:

$y = w * x + b + \varepsilon$

where:

y is the target variable.

w is the slope (weight) parameter.

ε is a loud sound.

Previous classification:

We will use Gaussian as w and b parameters:

$w \sim N(0, 1)$

$b \sim N(0, 1)$

Suppose Well: 4th noise ε Gaussian follows the distribution:

$\varepsilon \sim N(0, \sigma^2)$

Purpose:

Our goal is to estimate the last of the w and b parameters, which gives the observed data distribution.

Bayesian Inference Steps:

i. **Previous distribution:**

Provide a Gaussian priority for parameters w and b.

ii. **Probability:**

Defines probability based on Gaussian noise assumption.

iii. **Back Distributions:**

Use Bayes' theorem to calculate the final distributions of w and b based on data and priority.

iv. **Parameter Estimation**: Estimate the posterior distribution using methods such as

Markov Chain Monte Carlo (MCMC) or regression.

v. **Sampling operation:**

Let's use a simple dataset with three definitions:

Dataset: [(1, 2), (2, 4), (3, 5)]

Noise ^ variance 2 Let's assume that = 0.1.

We will use MCMC to sample from the next post.

vi. **Result:** After running the MCMC sampling procedure, we get samples from the final distribution for w and b. This model provides information about the uncertainty of measurement.

For example we will get the following distribution:

Back distribution for code

w: N(1.7, 0.3).

In this example, we use the Bayesian approach to estimate the parameters of the linear regression model. Bayesian inference allows us to combine prior knowledge, measure uncertainty, and provide a better understanding of the measured parameters.

Note that this example is simple and real applications will contain more models and data.